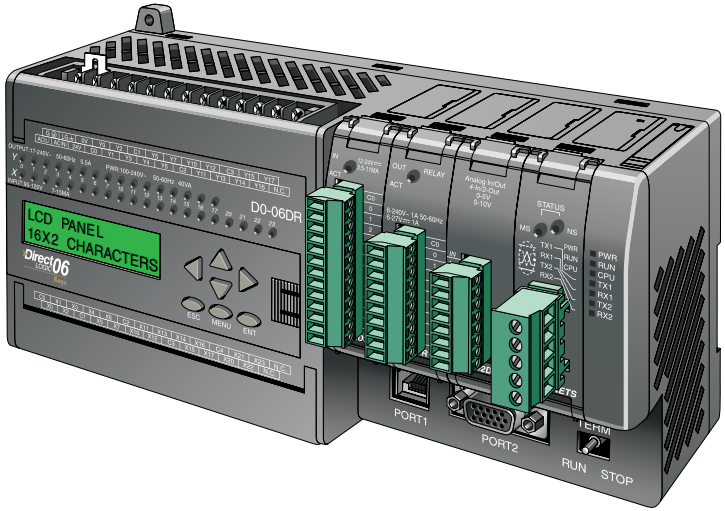
AUTOMATIONDIRECT.com

DL06 Micro PLC User Manual

Manual Number: D0-06USER-M



PLC Direct

~ WARNING ~

Thank you for purchasing automation equipment from **AutomationDirect.com®**, doing business as, **AutomationDirect**. We want your new automation equipment to operate safely. Anyone who installs or uses this equipment should read this publication (and any other relevant publications) before installing or operating the equipment.

To minimize the risk of potential safety problems, you should follow all applicable local and national codes that regulate the installation and operation of your equipment. These codes vary from area to area and usually change with time. It is your responsibility to determine which codes should be followed, and to verify that the equipment, installation, and operation is in compliance with the latest revision of these codes.

At a minimum, you should follow all applicable sections of the National Fire Code, National Electrical Code, and the codes of the National Electrical Manufacturer's Association (NEMA). There may be local regulatory or government offices that can also help determine which codes and standards are necessary for safe installation and operation.

Equipment damage or serious injury to personnel can result from the failure to follow all applicable codes and standards. We do not guarantee the products described in this publication are suitable for your particular application, nor do we assume any responsibility for your product design, installation, or operation.

Our products are not fault-tolerant and are not designed, manufactured or intended for use or resale as on-line control equipment in hazardous environments requiring fail-safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, direct life support machines, or weapons systems, in which the failure of the product could lead directly to death, personal injury, or severe physical or environmental damage ("High Risk Activities"). **AutomationDirect** specifically disclaims any expressed or implied warranty of fitness for High Risk Activities.

For additional warranty and safety information, see the Terms and Conditions section of our catalog. If you have any questions concerning the installation or operation of this equipment, or if you need additional information, please call us at 770-844-4200.

This publication is based on information that was available at the time it was printed. At **AutomationDirect** we constantly strive to improve our products and services, so we reserve the right to make changes to the products and/or publications at any time without notice and without any obligation. This publication may also discuss features that may not be available in certain revisions of the product.

Trademarks

This publication may contain references to products produced and/or offered by other companies. The product and company names may be trademarked and are the sole property of their respective owners. **AutomationDirect** disclaims any proprietary interest in the marks and names of others.

ADVERTENCIA

Gracias por comprar equipo de automatización de **Automationdirect.com®**. Deseamos que su nuevo equipo de automatización opere de manera segura. Cualquier persona que instale o use este equipo debe leer esta publicación (y cualquier otra publicación pertinente) antes de instalar u operar el equipo.

Para reducir al mínimo el riesgo debido a problemas de seguridad, debe seguir todos los códigos de seguridad locales o nacionales aplicables que regulan la instalación y operación de su equipo. Estos códigos varian de área en área y usualmente cambian con el tiempo. Es su responsabilidad determinar cuales códigos deben ser seguidos y verificar que el equipo, instalación y operación estén en cumplimiento con la revisión mas reciente de estos códigos.

Como mínimo, debe seguir las secciones aplicables del Código Nacional de Incendio, Código Nacional Eléctrico, y los códigos de (NEMA) la Asociación Nacional de Fabricantes Eléctricos de USA. Puede haber oficinas de normas locales o del gobierno que pueden ayudar a determinar cuales códigos y normas son necesarios para una instalación y operación segura.

Si no se siguen todos los códigos y normas aplicables, puede resultar en daños al equipo o lesiones serias a personas. No garantizamos los productos descritos en esta publicación para ser adecuados para su aplicación en particular, ni asumimos ninguna responsabilidad por el diseño de su producto, la instalación u operación.

Nuestros productos no son tolerantes a fallas y no han sido diseñados, fabricados o intencionados para uso o reventa como equipo de control en línea en ambientes peligrosos que requieren una ejecución sin fallas, tales como operación en instalaciones nucleares, sistemas de navegación aérea, o de comunicación, control de tráfico aéreo, máquinas de soporte de vida o sistemas de armamentos en las cuales la falla del producto puede resultar directamente en muerte, heridas personales, o daños físicos o ambientales severos ("Actividades de Alto Riesgo"). **Automationdirect.com** específicamente rechaza cualquier garantía ya sea expresada o implicada para actividades de alto riesgo.

Para información adicional acerca de garantía e información de seguridad, vea la sección de Términos y Condiciones de nuestro catálogo. Si tiene alguna pregunta sobre instalación u operación de este equipo, o si necesita información adicional, por favor llámenos al número 770-844-4200 en Estados Unidos.

Esta publicación está basada en la información disponible al momento de impresión. En **Automationdirect.com** nos esforzamos constantemente para mejorar nuestros productos y servicios, así que nos reservamos el derecho de hacer cambios al producto y/o a las publicaciones en cualquier momento sin notificación y sin ninguna obligación. Esta publicación también puede discutir características que no estén disponibles en ciertas revisiones del producto.

Marcas Registradas

Esta publicación puede contener referencias a productos producidos y/u ofrecidos por otras compañías. Los nombres de las compañías y productos pueden tener marcas registradas y son propiedad única de sus respectivos dueños. Automationdirect.com, renuncia cualquier interés propietario en las marcas y nombres de otros.

~ AVERTISSEMENT ~

Nous vous remercions d'avoir acheté l'équipement d'automatisation de **Automationdirect.com**[MC], en faisant des affaires comme, **AutomationDirect**. Nous tenons à ce que votre nouvel équipement d'automatisation fonctionne en toute sécurité. Toute personne qui installe ou utilise cet équipement doit lire la présente publication (et toutes les autres publications pertinentes) avant de l'installer ou de l'utiliser.

Afin de réduire au minimum le risque d'éventuels problèmes de sécurité, vous devez respecter tous les codes locaux et nationaux applicables régissant l'installation et le fonctionnement de votre équipement. Ces codes diffèrent d'une région à l'autre et, habituellement, évoluent au fil du temps. Il vous incombe de déterminer les codes à respecter et de vous assurer que l'équipement, l'installation et le fonctionnement sont conformes aux exigences de la version la plus récente de ces codes.

Vous devez, à tout le moins, respecter toutes les sections applicables du Code national de prévention des incendies, du Code national de l'électricité et des codes de la National Electrical Manufacturer's Association (NEMA). Des organismes de réglementation ou des services gouvernementaux locaux peuvent également vous aider à déterminer les codes ainsi que les normes à respecter pour assurer une installation et un fonctionnement sûrs.

L'omission de respecter la totalité des codes et des normes applicables peut entraîner des dommages à l'équipement ou causer de graves blessures au personnel. Nous ne garantissons pas que les produits décrits dans cette publication conviennent à votre application particulière et nous n'assumons aucune responsabilité à l'égard de la conception, de l'installation ou du fonctionnement de votre produit.

Nos produits ne sont pas insensibles aux défaillances et ne sont ni conçus ni fabriqués pour l'utilisation ou la revente en tant qu'équipement de commande en ligne dans des environnements dangereux nécessitant une sécurité absolue, par exemple, l'exploitation d'installations nucléaires, les systèmes de navigation aérienne ou de communication, le contrôle de la circulation aérienne, les équipements de survie ou les systèmes d'armes, pour lesquels la défaillance du produit peut provoquer la mort, des blessures corporelles ou de graves dommages matériels ou environnementaux («activités à risque élevé»). La société **AutomationDirect** nie toute garantie expresse ou implicite d'aptitude à l'emploi en ce qui a trait aux activités à risque élevé.

Pour des renseignements additionnels touchant la garantie et la sécurité, veuillez consulter la section Modalités et conditions de notre documentation. Si vous avez des questions au sujet de l'installation ou du fonctionnement de cet équipement, ou encore si vous avez besoin de renseignements supplémentaires, n'hésitez pas à nous téléphoner au 770-844-4200.

Cette publication s'appuie sur l'information qui était disponible au moment de l'impression. À la société **AutomationDirect**, nous nous efforçons constamment d'améliorer nos produits et services. C'est pourquoi nous nous réservons le droit d'apporter des modifications aux produits ou aux publications en tout temps, sans préavis ni quelque obligation que ce soit. La présente publication peut aussi porter sur des caractéristiques susceptibles de ne pas être offertes dans certaines versions révisées du produit.

Marques de commerce

La présente publication peut contenir des références à des produits fabriqués ou offerts par d'autres entreprises. Les désignations des produits et des entreprises peuvent être des marques de commerce et appartiennent exclusivement à leurs propriétaires respectifs. **AutomationDirect** nie tout intérêt dans les autres marques et désignations.

DL06 Micro PLC User Manual

AUTOMATIONDIRECT.com

Please include the Manual Number and the Manual Issue, both shown below, when communicating with Technical Support regarding this publication.

Manual Number: D0-06USER-M
Issue: 3rd Edition, Rev. I
Issue Date: 10/24

Publication History		
Issue	**Date**	**Description of Changes**
First Edition	7/02	Original
Rev. A	10/02	Updated drawing images and made minor corrections.
Rev. B	6/03	Added new PLC and made numerous corrections.
2nd Edition	3/04	Added two appendices, removed discrete module data and made numerous corrections.
3rd Edition	3/07	Corrected all tables, many corrections to Chapters 2, 3, 4, 5, 6, and 7; Chapter 3 (HSIO) was moved to the Appendices and Chapter 4 was divided into Chapters 3 & 4; added DS5 Intelligent Boxes to Chapter 5; added Ramp/Soak example to Chapter 8; Numbering Systems and Serial Communications were added to Appendices; many minor corrections were made throughout manual.
Rev. A	5/07	Minor corrections and updates.
Rev. B	6/11	Updated Chapter 5 with current ***Direct***SOFT dialog views, corrected number of registers needed to use the print message instruction, removed fuses and corrected I/O wiring drawings, and other minor corrections and updates.
Rev. C	2/13	Added H0-CTRIO2 references. Minor corrections and updates. Added transient suppression for inductive loads.
Rev. D	6/16	Corrections and updates.
Rev. E	8/18	Corrections, updates, removed mention of the DA-EU-M from Appendix J.
Rev. F	3/19	Corrections and updates
Rev. G	3/20	Corrections and updates.
Rev. H	5/22	Corrected image text ch2-22, 2-23
Rev. I	10/24	Removed Type 1 fonts. Added footer hyperlinks.

Table of Contents

Chapter 1: Getting Started

Chapter 2: Installation, Wiring, and Specifications

Chapter 3: CPU Specifications and Operation

Chapter 4: System Design and Configuration

Chapter 5: Standard RLL Instructions

Chapter 6: Drum Instruction Programming

Chapter 7: RLL^{PLUS} Stage Programming

Appendix A: Auxiliary Functions

Appendix J: European Union Directives (CE)

Appendix K: Introduction to Serial Communications

GETTING STARTED

In This Chapter...

Introduction

The Purpose of this Manual

Thank you for purchasing a DL06 Micro PLC. This manual shows you how to install, program, and maintain all PLCs in the DL06 family. It also helps you understand how to interface them to other devices in a control system. This manual contains important information for personnel who will install DL06 PLCs and for the PLC programmer. This user manual will provide the information you need to get and keep your system up and running.

Supplemental Manuals

The D0–OPTIONS–M manual contains technical information about the option cards available for the DL06 PLCs. This information includes specifications and wiring diagrams that will be indispensable if you use any of the optional I/O or communications cards. If you have purchased one of our operator interface panels or ***Direct***SOFT™ programming software, you will want to refer to the manuals that are written for these products.

Technical Support

We strive to make our manuals the best in the industry. We rely on your feedback to let us know if we are reaching our goal. If you cannot find the solution to your particular application, or, if for any reason you need technical assistance, please call us at

770–844–4200

Our technical support group will work with you to answer your questions. They are available Monday through Friday from 9:00 A.M. to 6:00 P.M. Eastern Time. We also encourage you to visit our web site where you can find technical and non-technical information about our products and our company.

http://www.automationdirect.com

If you have a comment, question or suggestion about any of our products, services, or manuals, please fill out and return the **Suggestions** card included with this manual.

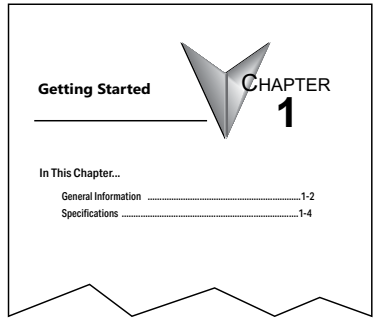
Conventions Used

When you see the notepad icon in the left-hand margin, the paragraph to its immediate right will be a **special note**. Notes represent information that may make your work quicker or more efficient.
The word **NOTE** in boldface type will mark the beginning of the text.

When you see the exclamation point icon in the left-hand margin, the paragraph to its immediate right will be a **warning**. This information could prevent injury, loss of property, or even death in extreme cases. Any warning in this manual should be regarded as critical information that should be read in its entirety.
The word **WARNING** in boldface type will mark the beginning of the text.

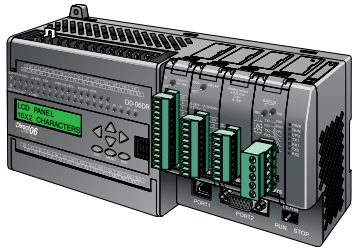
Key Topics for Each Chapter

The beginning of each chapter will list the key topics that can be found in that chapter.


Getting Started
CHAPTER 1
In This Chapter...
General Information 1-2
Specifications 1-4

DL06 Micro PLC Overview

The DL06 micro PLC family is a versatile product line that combines powerful features and a very compact footprint. The DL06 PLCs offer expandable I/O, high-speed counter, floating point, PID, etc. There are a number of communication options and an optional LCD display.



The DL06 PLC Features

The DL06 Micro PLC family includes nine different versions. All have the same appearance and CPU performance. The CPU offers an instruction set very similar to our powerful new DL260 CPU including new easy to use ASCII and MODBUS instructions. All DL06 PLCs have two built-in communications ports that can be used for programming, operator interface, networking, etc.

Units with DC inputs have selectable high-speed input features on four input points. Units with DC outputs offer selectable pulse output capability on the first and second output points. Details of these features and more are covered in Chapter 3, CPU Specifications and Operation. There are nine versions of the DL06 PLC. The most common industrial I/O types and power supply voltages are available. Consult the following table to find the model number of the PLC that best fits your application.

DL06 Micro PLC Family					
DL06 Part Number	**Discrete Input Type**	**Discrete Output Type**	**External Power**	**High-Speed Input**	**Pulse Output**
D0-06AA	AC	AC	95–240 VAC	No	No
D0-06AR	AC	Relay	95–240 VAC	No	No
D0-06DA	DC	AC	95–240 VAC	Yes	No
D0-06DD1	DC	DC Sinking	95–240 VAC	Yes	Yes
D0-06DD2	DC	DC Sourcing	95–240 VAC	Yes	Yes
D0-06DR	DC	Relay	95–240 VAC	Yes	No
D0-06DD1-D	DC	DC Sinking	12–24 VDC	Yes	Yes
D0-06DD2-D	DC	DC Sourcing	12–24 VDC	Yes	Yes
D0-06DR-D	DC	Relay	12–24 VDC	Yes	No

DirectSOFT Programming for Windows™

The DL06 Micro PLC can be programmed with *Direct*SOFT, a Windows-based software package that supports familiar features such as cut-and-paste between applications, point-and-click editing, viewing and editing multiple application programs at the same time, floating views, intelligent boxes, etc. Firmware version 2.10 is needed in order to use the intelligent boxes.

*Direct*SOFT (part number PC-DSOFTx) supports the *Direct*LOGIC CPU families. You can use *Direct*SOFT 5 to program the DL05, DL06, DL105, DL205, DL305, and DL405 CPUs. A separate manual discusses *Direct*SOFT programming software. Earlier programming software versions such as *Direct*SOFT32, version 4.0 can also be used to program the DL06.

Handheld Programmer

All DL06 Micro PLCs have a built-in programming port for use with the handheld programmer (D2–HPP), the same programmer used with the DL05, DL105 and DL205 families. The handheld programmer can be used to create, modify and debug your application program. A separate manual discusses the Handheld Programmer. Only D2–HPPs with firmware version 2.0 or later will program the DL06.

NOTE: *Not all instructions are available to use with the HPP - the real number instructions, for example.* ***Direct****SOFT will be needed to program instructions such as these.*

I/O Quick Selection Guide

The nine versions of the DL06 have input/output circuits which can interface to a wide variety of field devices. In several instances a particular input or output circuit can interface to either DC or AC voltages, or both sinking and sourcing circuit arrangements. Check this guide to find the proper DL06 Micro PLC to interface to the field devices in your application.

I/O Selection Guide						
DL06 Part Number	INPUTS			OUTPUTS		
	I/O type/ commons	Sink/Source	Voltage Ranges	I/O type/ commons	Sink/Source	Voltage/ Current Ratings*
D0–06AA	AC / 5	–	90 – 120 VAC	AC / 4	–	17 – 240 VAC, 50/60 Hz 0.5A
D0–06AR	AC / 5	–	90 – 120 VAC	Relay / 4	Sink or Source	6 – 27VDC, 2A 6 – 240 VAC, 2A
D0–06DA	DC / 5	Sink or Source	12 – 24 VDC	AC / 4	–	17 – 240 VAC, 50/60 Hz 0.5A
D0–06DD1	DC / 5	Sink or Source	12 – 24 VDC	DC / 4	Sink	6 – 27 VDC, 0.5A (Y0–Y1) 6 – 27 VDC, 1.0A (Y2–Y17)
D0–06DD2	DC / 5	Sink or Source	12 – 24 VDC	DC / 4	Source	12 – 24 VDC, 0.5A (Y0–Y1) 12 – 24 VDC, 1.0A (Y2–Y17)
D0–06DR	DC / 5	Sink or Source	12 – 24 VDC	Relay / 4	Sink or Source	6 – 27VDC, 2A 6 – 240 VAC, 2A
D0–06DD1–D	DC / 5	Sink or Source	12 – 24 VDC	DC / 4	Sink	6 – 27 VDC, 0.5A (Y0–Y1) 6 – 27 VDC, 1.0A (Y2–Y17)
D0–06DD2–D	DC / 5	Sink or Source	12 – 24 VDC	DC / 4	Source	12 – 24 VDC, 0.5A (Y0–Y1) 12 – 24 VDC, 1.0A (Y2–Y17)
D0–06DR–D	DC / 5	Sink or Source	12 – 24 VDC	Relay / 4	Sink or Source	6 – 27 VDC, 2A 6 – 240 VAC, 2A

* See Chapter 2, Specifications for more information about a particular DL06 version.

Quick Start

This example is not intended to tell you everything you need to know about programming and starting up a complex control system. It is only intended to give you an opportunity to demonstrate to yourself and others the basic steps necessary to power up the PLC and confirm its operation. Please look for warnings and notes throughout this manual for important information you will not want to overlook.

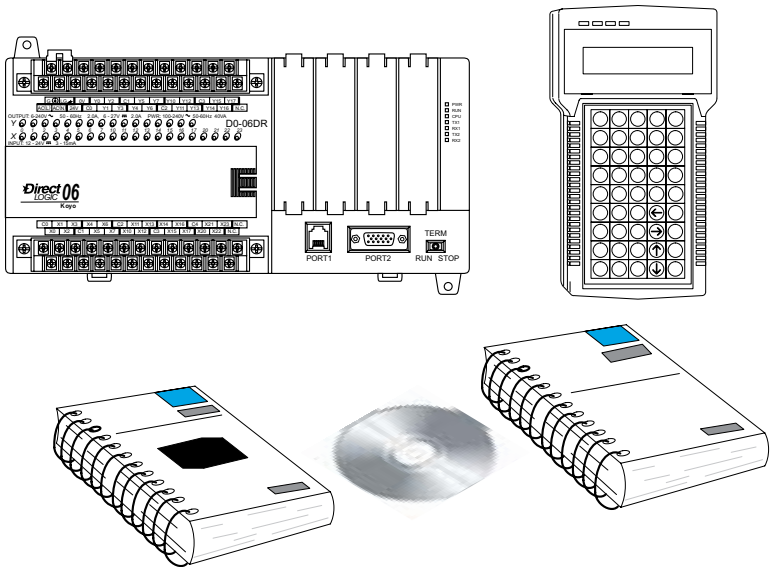
Step 1: Unpack the DL06 Equipment

Unpack the DL06 and gather the parts necessary to build this demonstration system. The recommended components are:

- DL06 Micro PLC
- AC power cord or DC power supply
- Toggle switches (see Step 2 on next page)
- Hook-up wire, 16-22 AWG
- DL06 User Manual (this manual)
- A small screwdriver, 5/8" flat or #1 Philips type

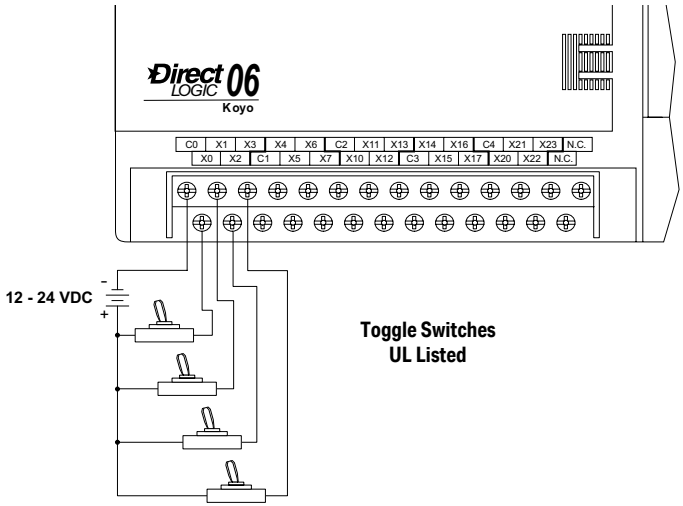
You will need at least one of the following programming options:

- ***Direct***SOFT Programming Software V5.0 or later (PC-DSOFTx), ***Direct***SOFT Programming Software Manual (included with the software), and a programming cable (D2-DSCBL connects the DL06 to a personal computer).
- D2-HPP Handheld Programmer, firmware version 2.0 or later, (comes with programming cable). Please purchase Handheld Programmer Manual D2-HPP-M separately.



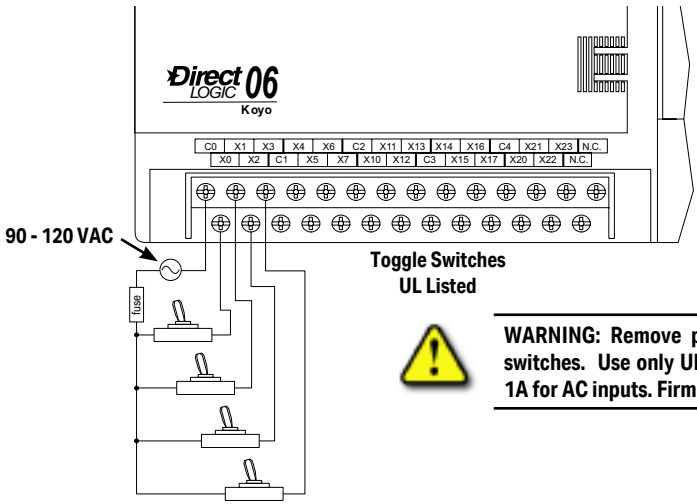
Step 2: Connect Switches to Input Terminals

To proceed with this quick-start exercise or to follow other examples in this manual, you will need to connect one or more input switches as shown below. If you have DC inputs on an AC-supply DL06, you can use the auxiliary 24VDC supply on the output terminal block or other external 12-24VDC power supply. Be sure to follow the instructions in the accompanying **WARNING** on this page.



D0-06DA, D0-06DD1, D0-06DD2, D0-06DR, D0-DD1-D, and D0-06DR1-D DC Input

Toggle Switches UL Listed



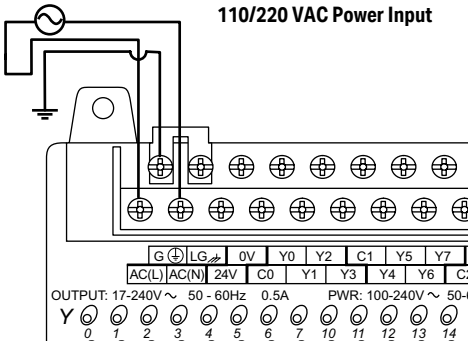
D0-06AA and D0-06AR AC input only

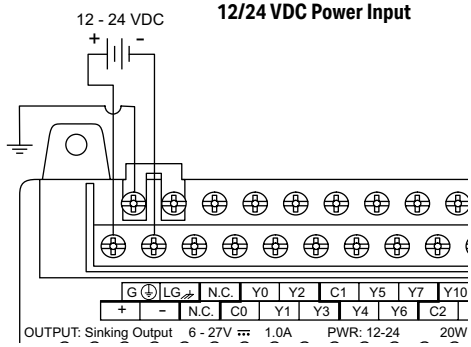
Toggle Switches UL Listed

WARNING: Remove power and unplug the DL06 when wiring the switches. Use only UL-approved switches rated for at least 250VAC, 1A for AC inputs. Firmly mount the switches before using.

Step 3: Connect the Power Wiring

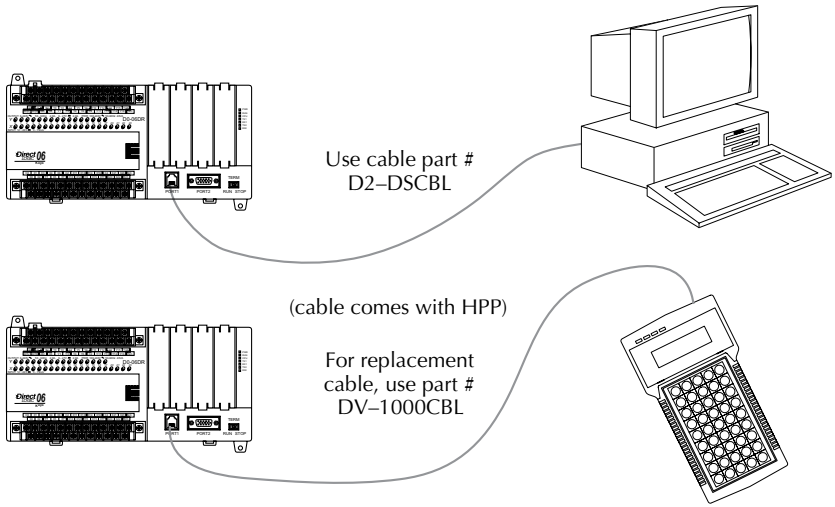
Connect the power input wiring for the DL06. Observe all precautions stated earlier in this manual. For more details on wiring, see Chapter 2 on Installation, Wiring, and Specifications. When the wiring is complete, close the connector covers. Do not apply power at this time.





Step 4: Connect the Programming Device

Most programmers will use *Direct*SOFT programming software, installed on a personal computer. An alternative, if you need a compact portable programming device, is the Handheld Programmer (firmware version 2.20 or later). Both devices will connect to COM port 1 of the DL06 via the appropriate cable.



***NOTE**: The Handheld Programmer cannot create or access LCD, ASCII or MODBUS instructions.*

Step 5: Switch on the System Power

Apply power to the system and ensure the PWR indicator on the DL06 is on. If not, remove power from the system and check all wiring and refer to the troubleshooting section in Chapter 9 for assistance.

Step 6: Initialize Scratchpad Memory

It's a good precaution to always clear the system memory (scratchpad memory) on a new DL06. There are two ways to clear the system memory:

- In *Direct*SOFT, select the PLC menu, then Setup and Initialize Scratch Pad. Initializing Scratch Pad will return secondary comm port settings and retentive range settings to default. If you have made any changes to these, you will need to note these changes and re-enter them after initializing Scratchpad.
- For the Handheld Programmer, use the AUX key and execute AUX 54.

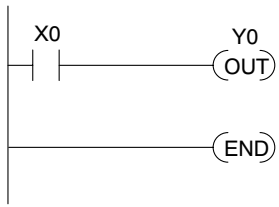
See the Handheld Programmer Manual for additional information.

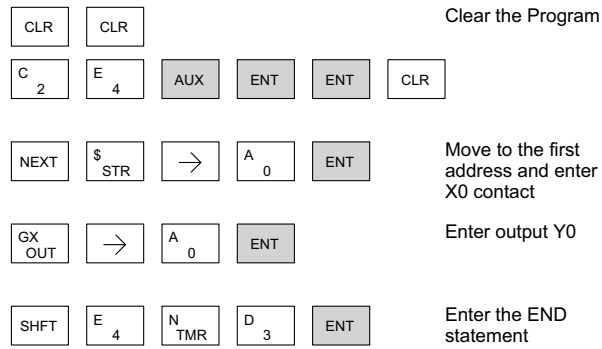
Step 7: Enter a Ladder Program

At this point, *Direct*SOFT programmers need to refer to Chapter 2 (Quick Start) in the *Direct*SOFT Programming Software Manual. There you will learn how to establish a communications link with the DL06 PLC, change CPU modes to Run or Program, and enter a program.

If you are learning how to program with the Handheld Programmer, make sure the CPU is in Program Mode (the RUN LED on the front of the DL06 should be off). If the RUN LED is on, use the MODE key on the Handheld Programmer to put the PLC in Program Mode, then switch to TERM.

Equivalent *Direct*SOFT display





Enter the following keystrokes on the Handheld Programmer.

After entering the simple example program, put the PLC in Run mode by using the Mode key on the Handheld Programmer.

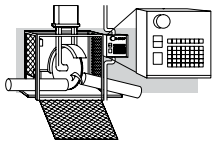
The RUN indicator on the PLC will illuminate, indicating the CPU has entered the Run mode. If not, repeat this step, ensuring the program is entered properly or refer to the troubleshooting guide in chapter 9.

After the CPU enters the run mode, the output status indicator for Y0 should follow the switch status on input channel X0. When the switch is on, the output will be on.

Steps to Designing a Successful System

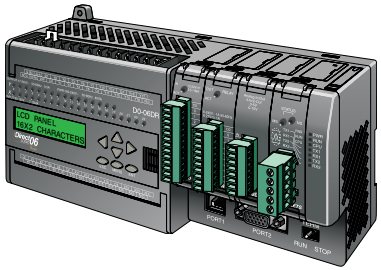
Step 1: Review the Installation Guidelines

Always make safety the first priority in any system design. Chapter 2 provides several guidelines that will help you design a safer, more reliable system. This chapter also includes wiring guidelines for the various versions of the DL06 PLC.



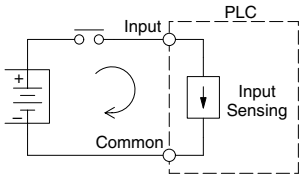
Step 2: Understand the PLC Setup Procedures

The PLC is the heart of your automation system. Make sure you take time to understand the various features and setup requirements.



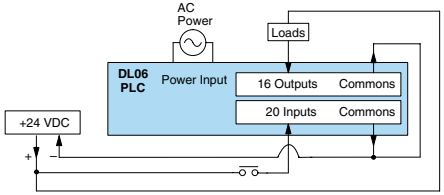
Step 3: Review the I/O Selection Criteria

There are many considerations involved when you select your I/O type and field devices. Take time to understand how the various types of sensors and loads can affect your choice of I/O type.



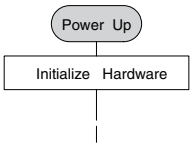
Step 4: Choose a System Wiring Strategy

It is important to understand the various system design options that are available before wiring field devices and field-side power supplies to the Micro PLC.



Step 5: Understand the System Operation

Before you begin to enter a program, it is very helpful to understand how the DL06 system processes information. This involves not only program execution steps, but also involves the various modes of operation and memory layout characteristics.

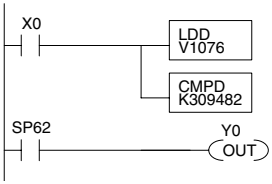


Step 6: Review the Programming Concepts

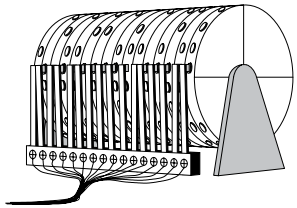
The DL06 PLC instruction set provides for three main approaches to solving the application program, depicted in the figure below.

- RLL diagram-style programming is the best tool for solving boolean logic and general CPU register/accumulator manipulation. It includes dozens of instructions, which will also be needed to augment drums and stages.
- The Timer/Event Drum Sequencer features up to 16 steps and offers both time and/or event-based step transitions. The DRUM instruction is best for a repetitive process based on a single series of steps.
- Stage programming (also called RLLplus) is based on state-transition diagrams. Stages divide the ladder program into sections which correspond to the states in a flow chart you draw for your process.

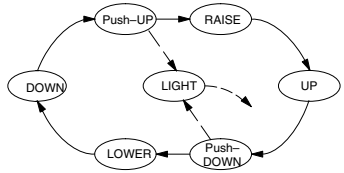
Standard RLL Programming
(see Chapter 5)



Timer/Event Drum Sequencer
(see Chapter 6)



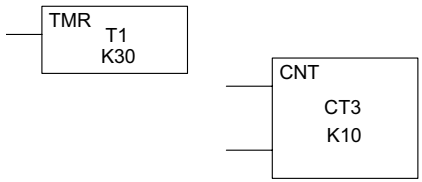
Stage Programming
(see Chapter 7)



After reviewing the programming concepts above, you'll be equipped with a variety of tools to write your application program.

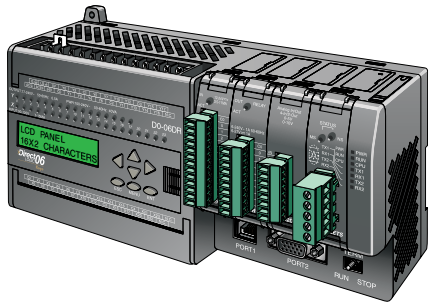
Step 7: Choose the Instructions

Once you have installed the Micro PLC and understand the main programming concepts, you can begin writing your application program. At that time you will begin to use one of the most powerful instruction sets available in a small PLC.



Step 8: Understand the Maintenance and Troubleshooting Procedures

Sometimes equipment failures occur when we least expect it. Switches fail, loads short and need to be replaced, etc. In most cases, the majority of the troubleshooting and maintenance time is spent trying to locate the problem. The DL06 Micro PLC has many built-in features, such as error codes, that can help you quickly identify problems.



Questions and Answers about DL06 Micro PLCs

Q. What is the instruction set like?

A. The instruction set is very close to that of our DL260 CPU. The DL06 instructions include the drum sequencing instruction, networking, ASCII, MODBUS, LCD, intelligent boxes and High-Speed I/O capabilities. High-Speed inputs are available on units with DC inputs only; high-speed outputs are available on units with DC outputs only.

Q. Do I have to buy the full DirectSOFT programming package to program the DL06?

A. Yes. The part number for *Direct*SOFT (PC-DSOFT6) is now used for all PLCs in the *Direct*LOGIC family, and the price is very affordable.

Q. Is the DL06 expandable?

A. Yes, the DL06 series function as stand-alone PLCs. However, option card slots allow you to expand the system without changing the footprint.

Q. Does the DL06 have motion control capability?

A. Yes, the DL06 has limited motion control capabilities. The High-Speed I/O features offer either encoder inputs with high-speed counting and presets with interrupt, or a pulse/direction output for stepper control. Three types of motion profiles are available, which are explained in Appendix E. The H0-CTRIO(2) option module can also be used to provide more motion functionality.

Q. Are the ladder programs stored in a removable EEPROM?

A. No. The DL06 contains a non-removable FLASH memory for program storage, which may be written and erased thousands of times. You may transfer programs to/from *Direct*SOFT on a PC.

Q. Does the DL06 contain fuses for its outputs?

A. There are no output circuit fuses. Therefore, we recommend fusing each channel, or fusing each common. See Chapter 2 for I/O wiring guidelines.

Q. Is the DL06 Micro PLC UL approved?

A. The Micro PLC has met the requirements of UL (Underwriters' Laboratories, Inc.) and CUL (Canadian Underwriters' Laboratories, Inc.). See our website, ***www.Automationdirect.com***, for complete details.

Q. Does the DL06 Micro PLC comply with European Union (EU) Directives?

A. The Micro PLC has met the requirements of the European Union Directives (CE). See our website, ***www.Automationdirect.com***, for complete details.

Q. Which devices can I connect to the communication ports of the DL06?

A. Port 1: The port is RS-232C, fixed at 9600 baud, odd parity, address 1, and uses the proprietary K-sequence protocol. The DL06 can also connect to MODBUS RTU and DirectNET networks as a slave device through port 1. The port communicates with the following devices:

- DV-1000 Data Access Unit, C-more, DirectTouch, LookoutDirect, DSData or Optimation Operator interface panels
- ***Direct***SOFT (running on a personal computer)
- D2-HPP handheld programmer
- Other devices which communicate via K-sequence, DirectNET, MODBUS RTU protocols should work with the DL06 Micro PLC. Contact the vendor for details.

A. Port 2: This is a multi-function port. It supports RS-232C, RS422, or RS485, with selective baud rates (300–38400 bps), address and parity. It also supports the proprietary K-sequence protocol, as well as ***Direct***Net and MODBUS RTU, ASCII In/Out, and non-sequence/print protocols.

Q. Can the DL06 accept 5VDC inputs?

A. No. 5 volts is lower than the DC input ON threshold. However, many TTL logic circuits can drive the inputs if they are wired as open collector (sinking) inputs. See Chapter 2 for I/O wiring guidelines.

INSTALLATION, WIRING, AND SPECIFICATIONS

In This Chapter...

Safety Guidelines

NOTE: Products with CE marks *perform their required functions safely and adhere to relevant standards as specified by CE directives, provided they are used according to their intended purpose, and the instructions in this manual are strictly followed. The protection provided by the equipment may be impaired if this equipment is used in a manner not specified in this manual. A listing of our international affiliates is available on our Web site: http://www.automationdirect.com*

WARNING: Providing a safe operating environment for personnel and equipment is your responsibility and should be your primary goal during system planning and installation. Automation systems can fail and may result in situations that can cause serious injury to personnel and/or damage equipment. Do not rely on the automation system alone to provide a safe operating environment. Sufficient emergency circuits should be provided to stop the operation of the PLC or the controlled machine or process, either partially or totally. These circuits should be routed outside the PLC in the event of controller failure, so that independent and rapid shutdown are available. Devices, such as mushroom switches or end of travel limit switches, should operate motor starter, solenoids, or other devices without being processed by the PLC. These emergency circuits should be designed using simple logic with a minimum number of highly reliable electromechanical components. Every automation application is different, so there may be special requirements for your particular application. Make sure all national, state, and local government requirements are followed for the proper installation and use of your equipment.

Plan for Safety

The best way to provide a safe operating environment is to make personnel and equipment safety part of the planning process. You should examine every aspect of the system to determine which areas are critical to operator or machine safety. If you are not familiar with PLC system installation practices, or your company does not have established installation guidelines, you should obtain additional information from the following sources.

- NEMA — The National Electrical Manufacturers Association, located in Washington, D.C., publishes many different documents that discuss standards for industrial control systems. You can order these publications directly from NEMA. Some of these include:
  ICS 1, General Standards for Industrial Control and Systems
  ICS 3, Industrial Systems
  ICS 6, Enclosures for Industrial Control Systems
- NEC — The National Electrical Code provides regulations concerning the installation and use of various types of electrical equipment. Copies of the NEC Handbook can often be obtained from your local electrical equipment distributor or your local library.
- Local and State Agencies — many local governments and state governments have additional requirements above and beyond those described in the NEC Handbook. Check with your local Electrical Inspector or Fire Marshall office for information.

Three Levels of Protection

The publications mentioned provide many ideas and requirements for system safety. At a minimum, you should follow these regulations. Also, you should use the following techniques, which provide three levels of system control.

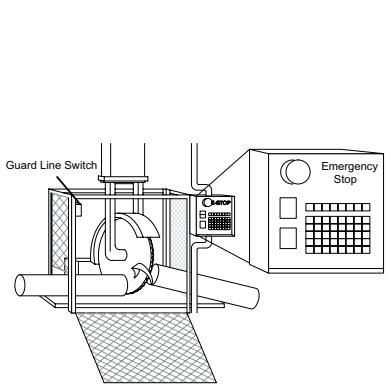
- Emergency stop switch for disconnecting system power
- Mechanical disconnect for output module power
- Orderly system shutdown sequence in the PLC control program

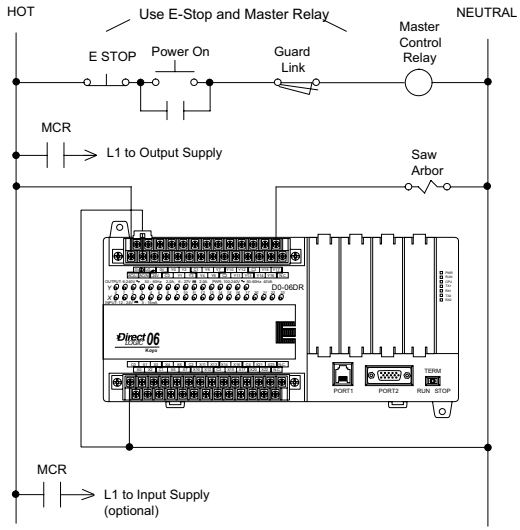
Emergency Stops

It is recommended that emergency stop circuits be incorporated into the system for every machine controlled by a PLC. For maximum safety in a PLC system, these circuits must not be wired into the controller, but should be hardwired external to the PLC. The emergency stop switches should be easily accessed by the operator and are generally wired into a master control relay (MCR) or a safety control relay (SCR) that will remove power from the PLC I/O system in an emergency.

MCRs and SCRs provide a convenient means for removing power from the I/O system during an emergency situation. By de-energizing an MCR (or SCR) coil, power to the input (optional) and output devices is removed. This event occurs when any emergency stop switch opens. However, the PLC continues to receive power and operate even though all its inputs and outputs are disabled.

The MCR circuit could be extended by placing a PLC fault relay (closed during normal PLC operation) in series with any other emergency stop conditions. This would cause the MCR circuit to drop the PLC I/O power in case of a PLC failure (memory error, I/O communications error, etc.).





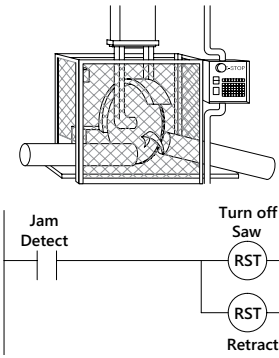
Emergency Power Disconnect

A properly rated emergency power disconnect should be used to power the PLC controlled system as a means of removing the power from the entire control system. It may be necessary to install a capacitor across the disconnect to protect against a condition known as **outrush**. This condition occurs when the output Triacs are turned off by powering off the disconnect, thus causing the energy stored in the inductive loads to seek the shortest distance to ground, which is often through the Triacs.

After an emergency shutdown or any other type of power interruption, there may be requirements that must be met before the PLC control program can be restarted. For example, there may be specific register values that must be established (or maintained from the state prior to the shutdown) before operations can resume. In this case, you may want to use retentive memory locations, or include constants in the control program to insure a known starting point.

Orderly System Shutdown

Ideally, the first level of fault detection is the PLC control program, which can identify machine problems. Certain shutdown sequences should be performed. The types of problems are usually things such as jammed parts, etc., that do not pose a risk of personal injury or equipment damage



WARNING: The control program must not be the only form of protection for any problems that may result in a risk of personal injury or equipment damage.

Class 1, Division 2 Approval

This equipment is suitable for use in Class 1, Zone 2, Division 2, groups A, B, C and D or non-hazardous locations only.

WARNING: Explosion Hazard! Substitution of components may impair suitability for Class 1, Division 2.

WARNING: Explosion Hazard! Do not disconnect equipment unless power has been switched off or the area is known to be non-hazardous.

WARNING: All models used with connector accessories must use R/C (ECBT2) mating plug for all applicable models. All mating plugs shall have suitable ratings for device.

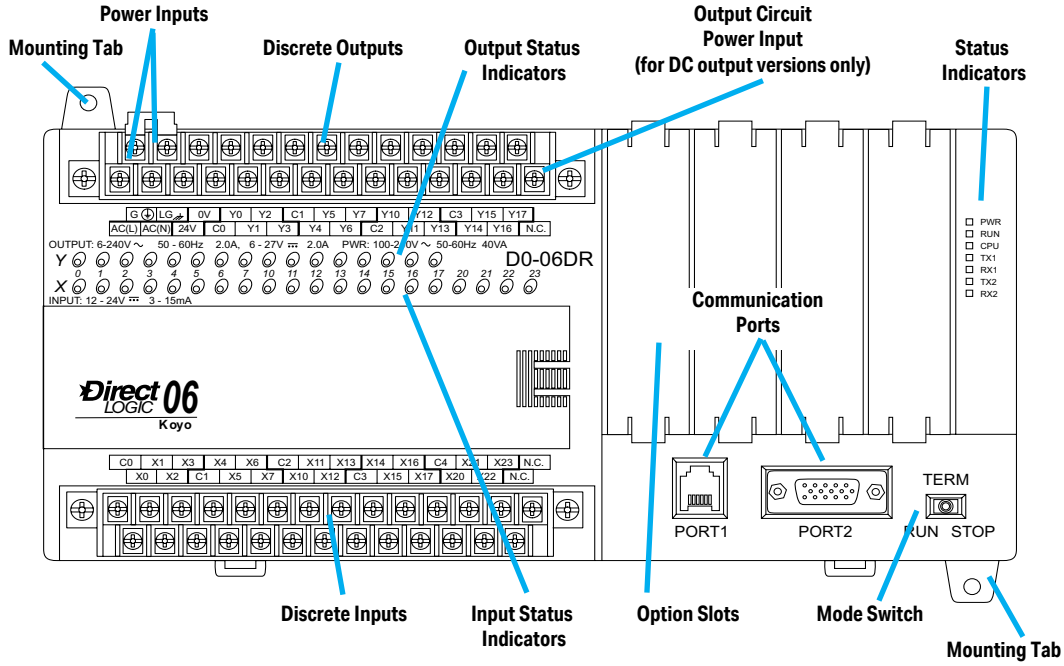
WARNING: This equipment is designed for use in Pollution Degree 2 environments (installed within an enclosure rated at least IP54).

WARNING: Transient suppression must be provided to prevent the rated voltage from being exceeded by 140%.

Orientation to DL06 Front Panel

Most connections, indicators and labels on the DL06 Micro PLCs are located on its front panel. The communication ports are located on front of the PLC, as are the option card slots and the mode selector switch. Please refer to the drawing below.

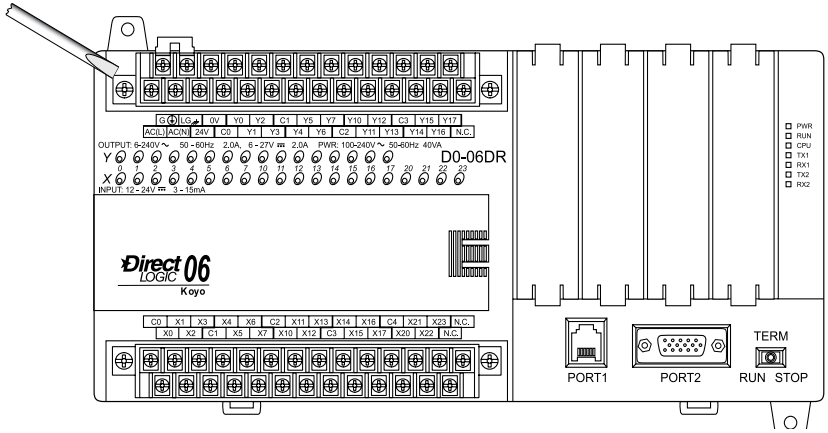
The output and power connector accepts external power and logic and chassis ground connections on the indicated terminals. The remaining terminals are for connecting commons and output connections Y0 through Y17. The sixteen output terminals are numbered in octal, Y0-Y7 and Y10-Y17. On DC output units, the end terminal on the right accepts power for the output stage. The input side connector provides the location for connecting the inputs X0 and X23 and the associated commons.



WARNING: For some applications, field device power may still be present on the terminal block even though the Micro PLC is turned off. To minimize the risk of electrical shock, check all field device power before you expose or remove either connector.

Terminal Block Removal

The DL06 terminals are divided into two groups. Each group has its own terminal block. The outputs and power wiring are on one block, and the input wiring is on the other. In some instances, it may be desirable to remove the terminal block for easy wiring. The terminal block is designed for easy removal with just a small screwdriver. The drawing below shows the procedure for removing one of the terminal blocks.



1. Loosen the retention screws on each end of the connector block.

2. From the center of the connector block, pry upward with the screwdriver until the connector is loose.

The terminal blocks on DL06 PLCs have regular (m3 size) screw terminals, which will accept either standard blade-type or #1 Philips screwdriver tips. Use No. 16 to 22 AWG solid/stranded wire. Be careful not to over-tighten; maximum torque is 0.882 to 1.020 N·m (7.806 to 9.028 in·lbs).

Spare terminal blocks are available in an accessory kit. Please refer to part number D0-ACC-2. You can find this and other accessories on our web site.

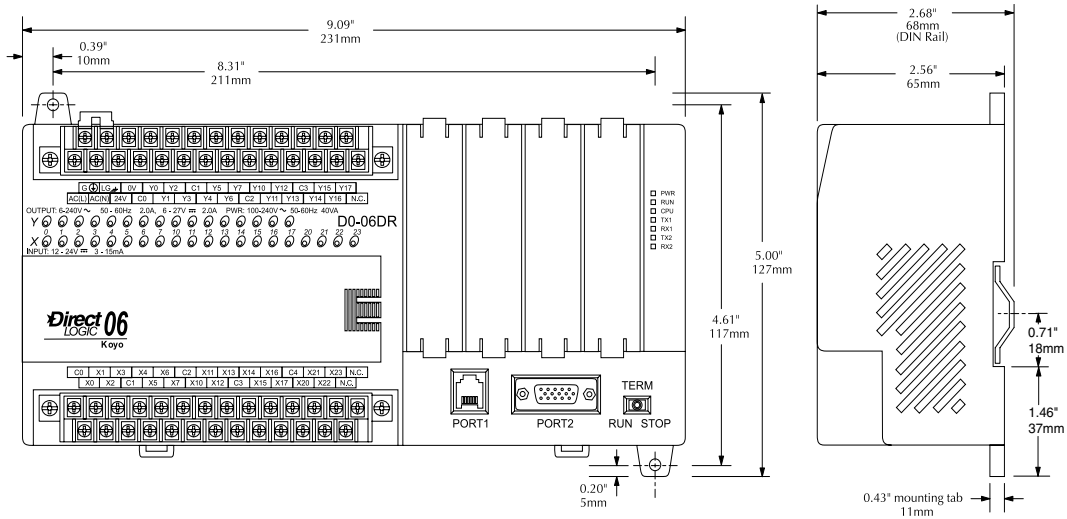
Mounting Guidelines

In addition to the panel layout guidelines, other specifications can affect the installation of a PLC system. Always consider the following:

- Environmental Specifications
- Power Requirements
- Agency Approvals
- Enclosure Selection and Component Dimensions

Unit Dimensions

The following diagram shows the outside dimensions and mounting hole locations for all versions of the DL06. Make sure you follow the installation guidelines to allow proper spacing from other components.



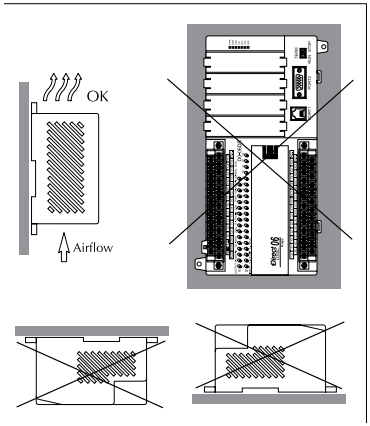
Enclosures

Your selection of a proper enclosure is important to ensure safe and proper operation of your DL06 system. Applications of DL06 systems vary and may require additional features. The minimum considerations for enclosures include:

- Conformance to electrical standards
- Protection from the elements in an industrial environment
- Common ground reference
- Maintenance of specified ambient temperature
- Access to equipment
- Security or restricted access
- Sufficient space for proper installation and maintenance of equipment

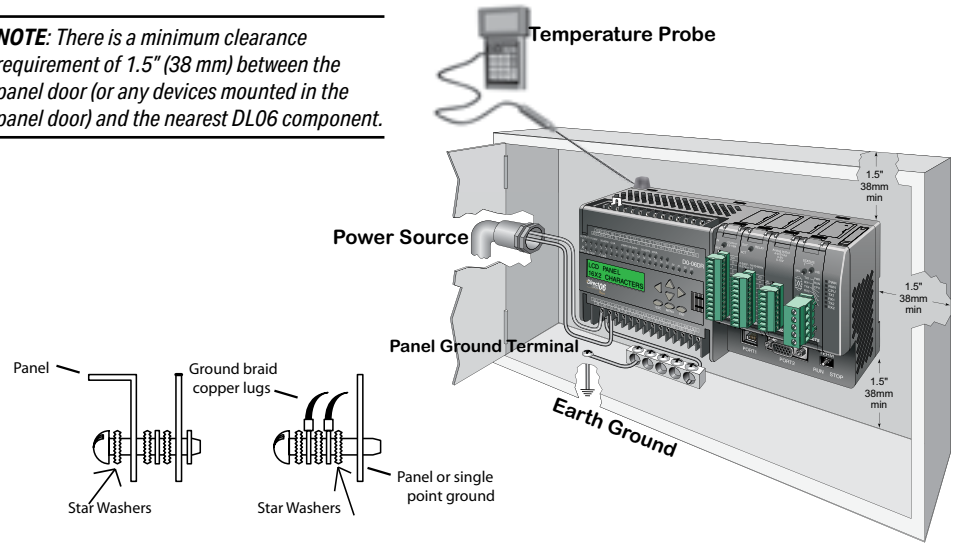
Panel Layout & Clearances

There are many things to consider when designing the panel layout. The following items correspond to the diagram shown. **Note:** there may be additional requirements, depending on your application and use of other components in the cabinet.



1. Mount the PLCs horizontally as shown below to provide proper ventilation. You cannot mount the DL06 units vertically, upside down, or on a flat horizontal surface. If you place more than one unit in a cabinet, there must be a minimum of 7.2" (183mm) between the units.
2. Provide a minimum clearance of 1.5" (38mm) between the unit and all sides of the cabinet. Remember to allow for any operator panels or other items mounted in the door.
3. There should also be at least 3" (78mm) of clearance between the unit and any wiring ducts that run parallel to the terminals.
4. The ground terminal on the DL06 base must be connected to a single point ground. Use copper stranded wire to achieve a low impedance. Copper eye lugs should be crimped and soldered to the ends of the stranded wire to ensure good surface contact.

***NOTE**: There is a minimum clearance requirement of 1.5" (38 mm) between the panel door (or any devices mounted in the panel door) and the nearest DL06 component.*



5. There must be a single point ground (i.e., copper bus bar) for all devices in the panel requiring an earth ground return. The single point of ground must be connected to the panel ground termination. The panel ground termination must be connected to earth ground. Minimum wire sizes, color coding, and general safety practices should comply with appropriate electrical codes and standards for your area.

6. A good common ground reference (Earth ground) is essential for proper operation of the DL06. One side of all control and power circuits and the ground lead on flexible shielded cable must be properly connected to Earth ground. There are several methods of providing an adequate common ground reference, including:

   a) Installing a ground rod as close to the panel as possible
   b) Connection to incoming power system ground

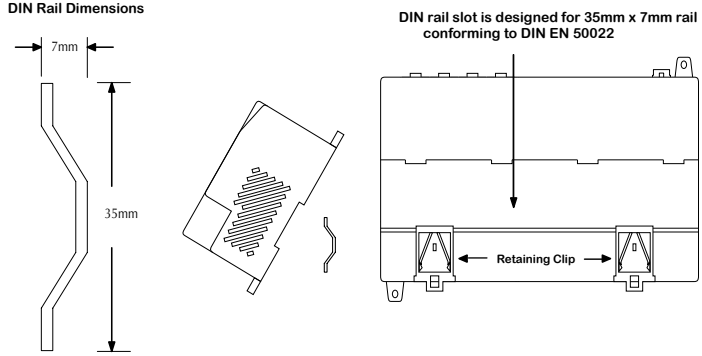
7. Evaluate any installations where the ambient temperature may approach the lower or upper limits of the specifications. If you suspect the ambient temperature will not be within the operating specification for the DL06 system, measures such as installing a cooling/heating source must be taken to get the ambient temperature within the range of specifications.

8. The DL06 systems are designed to be powered by 95–240 VAC or 12–24 VDC normally available throughout an industrial environment. Electrical power in some areas where the PLCs are installed is not always stable and storms can cause power surges. Due to this, power line filters are recommended for protecting the DL06 PLCs from power surges and EMI/RFI noise. The Automationdirect power line filter, for use with 120VAC and 240VAC, 1–5 Amps, is an excellent choice (locate at automationdirect.com); however, you can use a filter of your choice. These units install easily between the power source and the PLC.

NOTE: *If you are using other components in your system, make sure you refer to the appropriate manual to determine how those units can affect mounting dimensions.*

Using Mounting Rails

DL06 Micro PLCs can be secured to a panel by using mounting rails. We recommend rails that conform to DIN EN standard 50022. They are approximately 35 mm high, with a depth of 7 mm. If you mount the Micro PLC on a rail, do consider using end brackets on each side of the PLC. The end bracket helps keep the PLC from sliding horizontally along the rail, reducing the possibility of accidentally pulling the wiring loose. On the bottom of the PLC are two small retaining clips. To secure the PLC to a DIN rail, place it onto the rail and gently push up on the clips to lock it onto the rail. To remove the PLC, pull down on the retaining clips, lift up on the PLC slightly, then pull it away from the rail.



NOTE: *Refer to our catalog or web site for a complete listing of* ***DINnector*** *connection systems.*

Environmental Specifications

The following table lists the environmental specifications that generally apply to DL06 Micro PLCs. The ranges that vary for the Handheld Programmer are noted at the bottom of this chart. Certain output circuit types may have derating curves, depending on the ambient temperature and the number of outputs ON. Please refer to the appropriate section in this chapter pertaining to your particular DL06 PLC.

Environmental Specifications	
Specification	**Rating**
Storage temperature	–4°F to 158°F (–20°C to 70°C)
Ambient operating temperature*	32°F to 131°F (0°C to 55°C)
Ambient humidity**	5% – 95% relative humidity (non–condensing)
Vibration resistance	MIL STD 810C, Method 514.2
Shock resistance	MIL STD 810C, Method 516.2
Noise immunity	NEMA (ICS3–304)
Atmosphere	No corrosive gases
Agency approvals	UL, CE (C1D2), FCC class A

*** Operating temperature for the Handheld Programmer and the DV–1000 is 32° to 122°F (0° to 50°C) Storage temperature for the Handheld Programmer and the DV–1000 is –4° to 158°F (–20° to 70°C).**

****Equipment will operate down to 5% relative humidity; however, static electricity problems occur much more frequently at low humidity levels (below 30%). Make sure you take adequate precautions when you touch the equipment. Consider using ground straps, anti-static floor coverings, etc., if you use the equipment in low-humidity environments.**

Agency Approvals

Some applications require agency approvals for particular components. The DL06 Micro PLC agency approvals are listed below:

- UL (Underwriters' Laboratories, Inc.)
- CUL (Canadian Underwriters' Laboratories, Inc.)
- CE (European Economic Union)

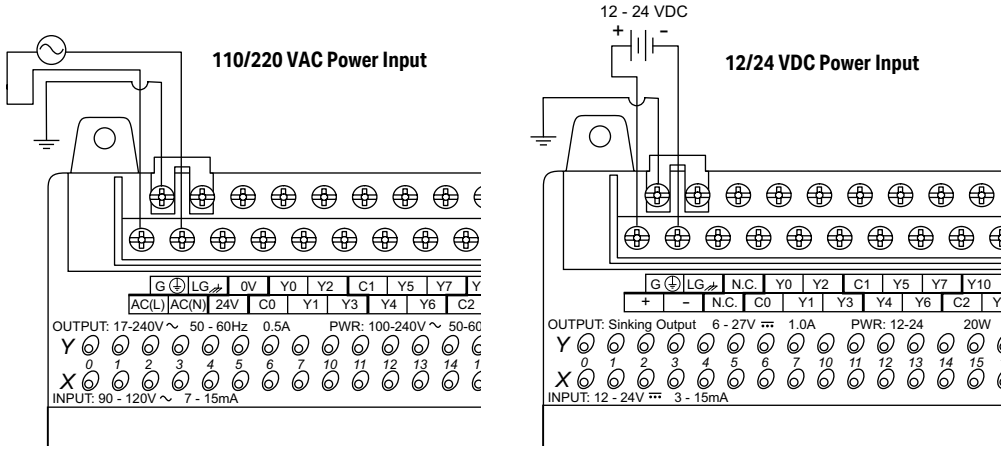
Marine Use

American Bureau of Shipping (ABS) certification requires flame-retarding insulation as per 4-8-3/5.3.6(a). ABS will accept Navy low smoke cables, cable qualified to NEC **Plenum rated** (fire resistant level 4), or other similar flammability resistant rated cables. Use cable specifications for your system that meet a recognized flame retardant standard (i.e., UL, IEEE, etc.), including evidence of cable test certification (i.e., tests certificate, UL file number, etc.).

***NOTE:** Wiring must be **low smoke** per the above paragraph. Teflon coated wire is also recommended.*

Wiring Guidelines

Connect the power input wiring for the DL06. Observe all precautions stated earlier in this manual. When the wiring is complete, close the connector covers. Do not apply power at this time.



WARNING: Once the power wiring is connected, secure the terminal block cover in the closed position. There is a risk of electrical shock if you accidentally touch the connection terminals or power wiring when the cover is open.

External Power Source

The power source must be capable of suppling voltage and current complying with individual Micro PLC specifications, according to the following specifications:

NOTE: *The rating between all internal circuits is BASIC INSULATION ONLY.*

Power Source Specifications		
Item	**DL06 AC Powered Units**	**DL06 DC Powered Units**
Input Voltage Range	110/220 VAC (100–240 VAC/50-60 Hz)	12–24 VDC (10.8–26.4 VDC)
Maximum Inrush Current	13A, 1ms (100–240 VAC) 15A, 1ms (240–264 VAC)	10A
Maximum Power	40VA	20W
Voltage Withstand (dielectric)	1 minute @ 1500VAC between primary, secondary, field ground	
Insulation Resistance	> 10MΩ at 500VDC	

NOTE: *Recommended wire size for field devices is 16–22 AWG solid/stranded. Tighten terminal screws to 7.81 lb·in (0.882 N·m) to 9.03 lb-in (1.02 N·m).*

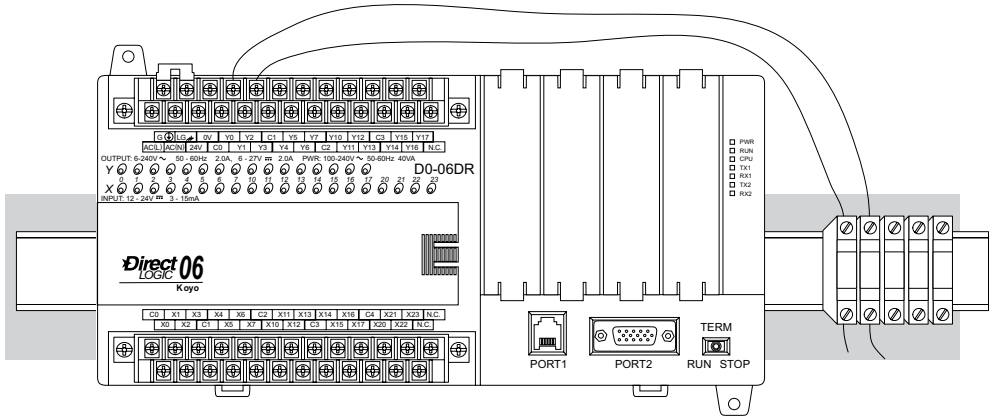
Planning the Wiring Routes

The following guidelines provide general information on how to wire the I/O connections to DL06 Micro PLCs. Refer to the corresponding specification sheet which appears later in this chapter for specific information on wiring a particular PLC .

1. Each terminal connection of the DL06 PLC can accept one 16 AWG wire or two 18 AWG size wires. Do not exceed this recommended capacity.
2. Always use a continuous length of wire. Do not splice wires to attain a needed length.
3. Use the shortest possible wire length.
4. Use wire trays for routing where possible.
5. Avoid running wires near high energy wiring.
6. Avoid running input wiring close to output wiring where possible.
7. To minimize voltage drops when wires must run a long distance, consider using multiple wires for the return line.
8. Avoid running DC wiring in close proximity to AC wiring where possible.
9. Avoid creating sharp bends in the wires.
10. Install the recommended power line filter to reduce power surges and EMI/RFI noise.

Fuse Protection for Input and Output Circuits

Input and Output circuits on DL06 Micro PLCs do not have internal fuses. In order to protect your Micro PLC, we suggest you add external fuses to your I/O wiring. A fast-blow fuse, with a lower current rating than the I/O bank's common current rating, can be wired to each common. Or, a fuse with a rating of slightly less than the maximum current per output point can be added to each output. Refer to the Micro PLC specification sheets further in this chapter to find the maximum current per output point or per output common. Adding the external fuse does not guarantee the prevention of Micro PLC damage, but it will provide added protection.



I/O Point Numbering

All DL06 Micro PLCs have a fixed I/O configuration. It follows the same octal numbering system used on other ***Direct***Logic family PLCs, starting at X0 and Y0. The letter X is always used to indicate inputs and the letter Y is always used for outputs.

The I/O numbering always starts at zero and does not include the digits 8 or 9. The addresses are typically assigned in groups of 8 or 16, depending on the number of points in an I/O group. For the DL06, the twenty inputs use reference numbers X0 – X23. The sixteen output points use references Y0 – Y17.

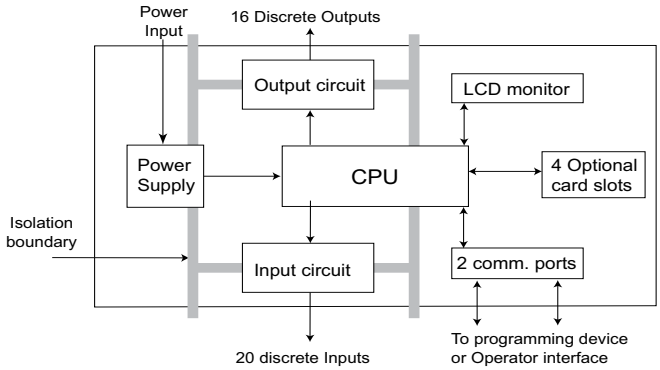
Additional I/O modules can be installed in the four option slots. See the DL05/06 Option Modules User Manual, D0-OPTIONS-M, for a complete selection of modules and how to address them in the DL06. This manual can either be ordered from Automationdirect or downloaded from our website.

System Wiring Strategies

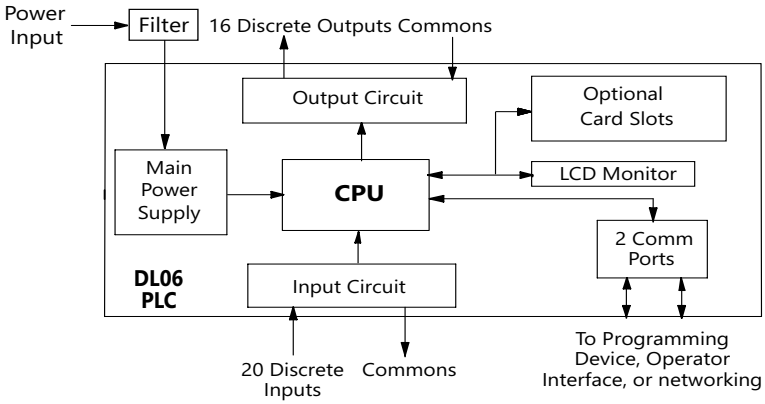
The DL06 Micro PLC is very flexible and will work in many different wiring configurations. By studying this section before actual installation, you can probably find the best wiring strategy for your application. This will help to lower system cost and wiring errors, and avoid safety problems.

PLC Isolation Boundaries

PLC circuitry is divided into three main regions separated by isolation boundaries, shown in the drawing below. Electrical isolation provides safety, so that a fault in one area does not damage another. A power line filter will provide isolation between the power source and the power supply. A transformer in the power supply provides magnetic isolation between the primary and secondary sides. Opto-couplers provide optical isolation in Input and Output circuits. This isolates logic circuitry from the field side, where factory machinery connects. Note that the discrete inputs are isolated from the discrete outputs, because each is isolated from the logic side. Isolation boundaries protect the operator interface (and the operator) from power input faults or field wiring faults. *When wiring a PLC, it is extremely important to avoid making external connections that connect logic side circuits to any other.*

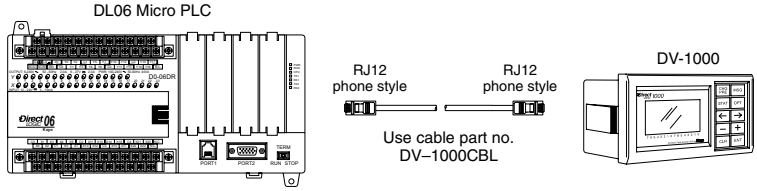


The next figure shows the internal layout of DL06 PLCs, as viewed from the front panel.

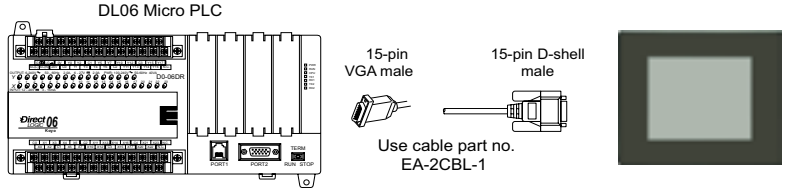


Connecting Operator Interface Devices

Operator interfaces require data and power connections. Some operator interfaces usually require separate AC power. However, other operator interface devices like the popular DV-1000 Data Access Unit may be powered directly from the DL06 Micro PLC. Connect the DV-1000 to communication port 1 on the DL06 Micro PLC using the cable shown below. A single cable contains transmit/receive data wires and +5V power.

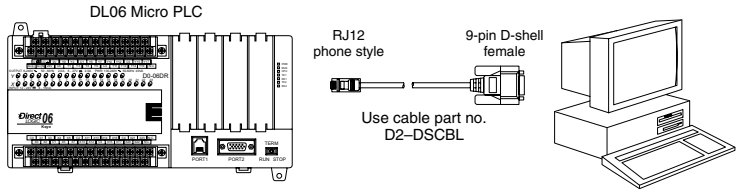


C-more operator interface touch panels use a provided 24 VDC plug-in power supply. Connect the DL06 to the serial connector on the rear of the C-more panel using the cable shown below.

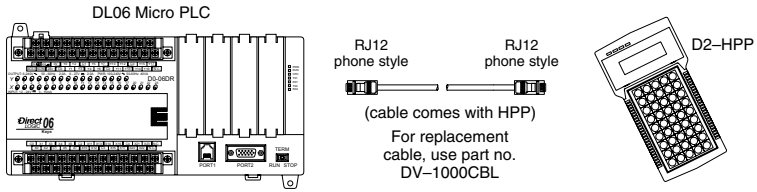


Connecting Programming Devices

DL06 Micro PLCs can be programmed with either a handheld programmer or with ***Direct***SOFT on a PC. Connect the DL06 to a PC using the cable shown below.



The D2-HPP Handheld Programmer comes with a communications cable. For a replacement part, use the cable shown below.



Sinking / Sourcing Concepts

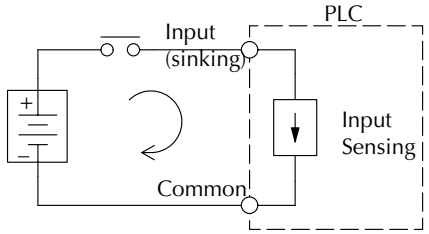
Before going further in our presentation of wiring strategies, we need to introduce the concepts of **sinking** and **sourcing**. These terms apply to typical input or output circuits. It is the goal of this section to make these concepts easy to understand. First, we give the following short definitions, followed by practical applications.

Sinking = Path to supply ground (–)

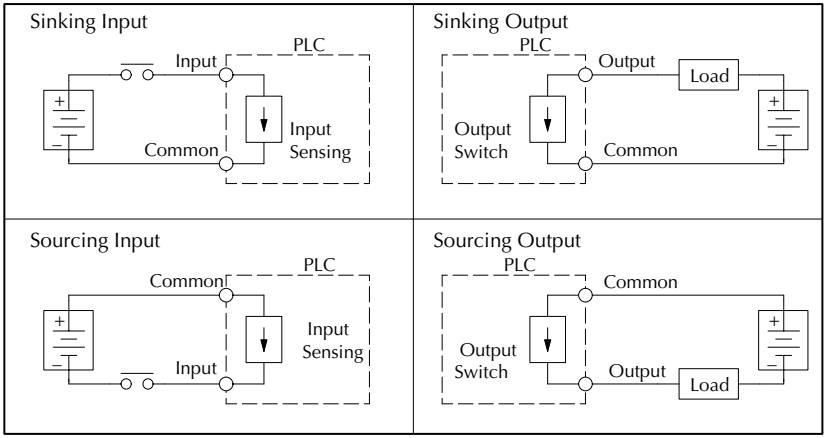
Sourcing = Path to supply source (+)

Notice the reference to (+) and (–) polarities. S*inking and sourcing terminology applies only to DC input and output circuits.* Input and output points that are either sinking or sourcing can conduct current in only one direction. This means it is possible to connect the external supply and field device to the I/O point with current trying to flow in the wrong direction, and the circuit will not operate. However, we can successfully connect the supply and field device every time by understanding **sourcing** and **sinking**.

For example, the figure to the right depicts a **sinking** input. To properly connect the external supply, we just have to connect it so the input *provides a path to ground (–)*. So, we start at the PLC input terminal, follow through the input sensing circuit, exit at the common terminal, and connect the supply (–) to the common terminal. By adding the switch, between the supply (+) and the input, we have completed the circuit. Current flows in the direction of the arrow when the switch is closed.

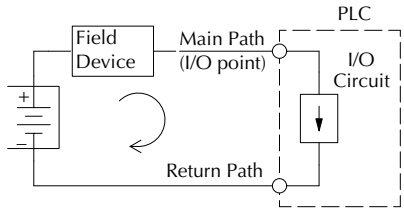


By applying the circuit principle above to the four possible combinations of input/output **sinking/sourcing** types, we have the four circuits as shown below. The DC-powered DL06 Micro PLCs have selectable **sinking** or **sourcing** inputs and either **sinking** or **sourcing** outputs. Any pair of input/output circuits shown below is possible with one of the DL06 models.

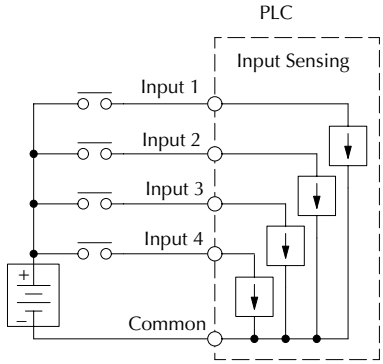


I/O Common Terminal Concepts

In order for a PLC I/O circuit to operate, current must enter at one terminal and exit at another. This means at least two terminals are associated with every I/O point. In the figure to the right, the input or output terminal is the *main path* for the current. One additional terminal must provide the *return path* to the power supply.

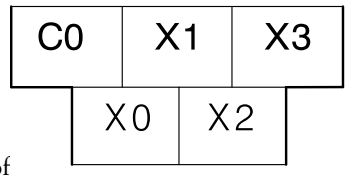


Most input or output point groups on PLCs share the return path among two or more I/O points. The figure to the right shows a group (*or bank*) of 4 input points which share a **common** return path. In this way, the four inputs require only five terminals instead of eight.



***NOTE**: In the circuit to the right, the current in the common path is 4 times any channel's input current when all inputs are energized. This is especially important in output circuits, where heavier gauge wire is sometimes necessary on commons.*

Most DL06 input and output circuits are grouped into banks that share a common return path. The best indication of I/O common grouping is on the wiring label. The I/O common groups are separated by a bold line. A thinner line separates the inputs associated with that common. To the right, notice that X0, X1, X2, and X3 share the common terminal C0, located to the left of X1.



The following complete set of labels shows five banks of four inputs and four banks of four outputs. One common is provided for each bank.

G ⏚	LG	0V	Y0	Y2	C1	Y5	Y7	Y10	Y12	C3	Y15	Y17
AC(L)	AC(N)	24V	C0	Y1	Y3	Y4	Y6	C2	Y11	Y13	Y14	Y16	N.C.

C0	X1	X3	X4	X6	C2	X11	X13	X14	X16	C4	X21	X23	N.C.
X0	X2	C1	X5	X7	X10	X12	C3	X15	X17	X20	X22	N.C.

This set of labels is for DC (sinking) output versions such as the D0-06DD1 and D0-06DD1-D. One common is provided for each group of four outputs, and one designated terminal on the output side accepts power for the output stage.

G ⏚	LG	0V	Y0	Y2	C1	Y5	Y7	Y10	Y12	C3	Y15	Y17
AC(L)	AC(N)	24V	C0	Y1	Y3	Y4	Y6	C2	Y11	Y13	Y14	Y16	+V

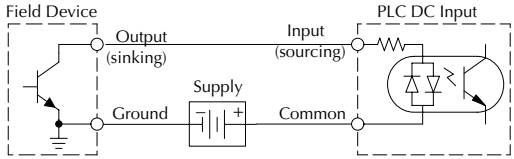
C0	X1	X3	X4	X6	C2	X11	X13	X14	X16	C4	X21	X23	N.C.
X0	X2	C1	X5	X7	X10	X12	C3	X15	X17	X20	X22	N.C.

Connecting DC I/O to Solid State Field Devices

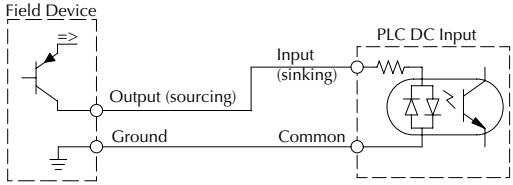
In the previous section on sinking and sourcing concepts, we discussed DC I/O circuits that only allow current to flow one way. This is also true for many of the field devices which have solid-state (transistor) interfaces. In other words, field devices can also be sourcing or sinking. *When connecting two devices in a series DC circuit (as is the case when wiring a field device to a PLC DC input or output), one must be wired as sourcing and the other as sinking.*

Solid State Input Sensors

The DL06's DC inputs are flexible in that they detect current flow in either direction, so they can be wired as either sourcing or sinking. In the following circuit, a field device has an open-collector NPN transistor output. It sinks current from the PLC input point, which sources current. The power supply can be the included auxiliary 24VDC power supply or another supply (+12VDC or +24VDC), as long as the input specifications are met.



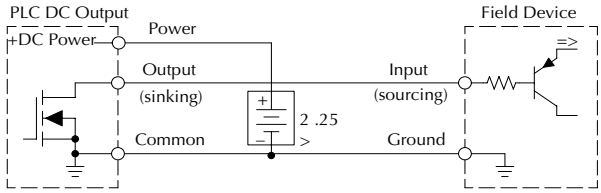
In the next circuit, a field device has an open-emitter PNP transistor output. It sources current to the PLC input point, which sinks the current back to ground. Since the field device is sourcing current, no additional power supply is required between the device and the PLC DC Input.



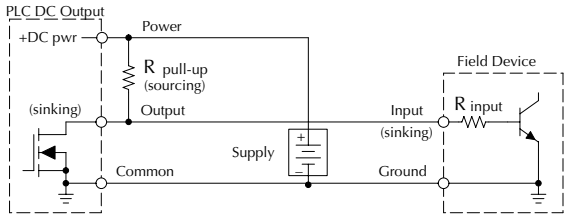
Solid State Output Loads

Sometimes an application requires connecting a PLC output point to a solid state input on a device. This type of connection is usually made to carry a low-level signal, not to send DC power to an actuator.

The DL06 PLC family offers DC outputs that are sinking only or DC outputs that are sourcing. All sixteen outputs have the same electrical common, even though there are four common terminal screws. In the following circuit, the PLC output point sinks current to the output common when energized. It is connected to a sourcing input of a field device input.



In the next example we connect a PLC DC output point to the sinking input of a field device. This is a bit tricky, because both the PLC output and field device input are sinking type. Since the circuit must have one sourcing and one sinking device, we add sourcing capability to the PLC output by using a pull-up resistor. In the circuit below, we connect $R_{pull\text{-}up}$ from the output to the DC output circuit power input.



NOTE: *DO NOT attempt to drive a heavy load (>25mA) with this pull-up method.*

NOTE: *Using the pull-up resistor to implement a sourcing output has the effect of inverting the output point logic. In other words, the field device input is energized when the PLC output is OFF, from a ladder logic point-of-view. Your ladder program must comprehend this and generate an inverted output. Or, you may choose to cancel the effect of the inversion elsewhere, such as in the field device.*

It is important to choose the correct value of $R_{pull\text{-}up}$. In order to do so, we need to know the nominal input current to the field device (I_{input}) when the input is energized. If this value is not known, it can be calculated as shown (a typical value is 15mA). Then use I_{input} and the voltage of the external supply to compute $R_{pull\text{-}up}$. Then calculate the power $P_{pull\text{-}up}$ (in watts), in order to size $R_{pull\text{-}up}$ properly.

$$I_{input} = \frac{V_{input\ (turn\text{–}on)}}{R_{input}}$$

$$R_{pull\text{-}up} = \frac{V_{supply} - 0.7}{I_{input}} - R_{input} \qquad P_{pull\text{-}up} = \frac{V_{supply}^2}{R_{pullup}}$$

Relay Output Wiring Methods

The D0–06AR and the D0–06DR models feature relay outputs. Relays are best for the following applications:

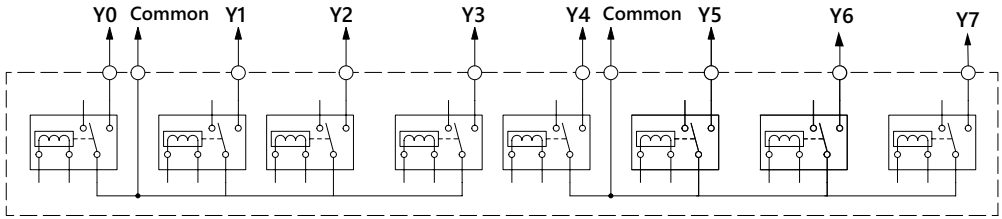
- Loads that require higher currents than the solid-state DL06 outputs can deliver
- Cost-sensitive applications
- Some output channels need isolation from other outputs (such as when some loads require AC while others require DC)

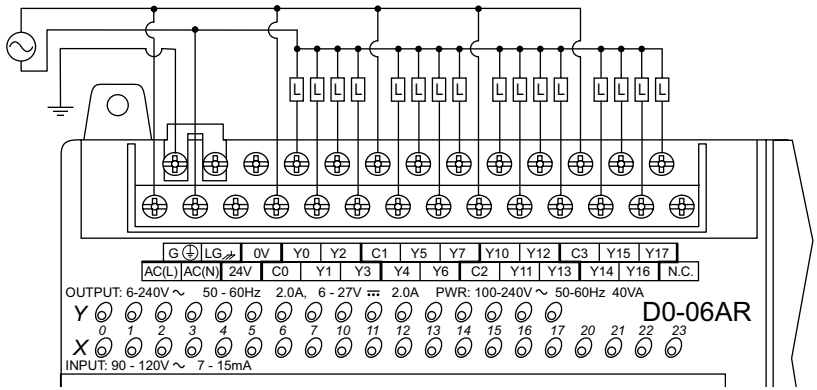
Some applications in which NOT to use relays:

- Loads that require currents under 10mA
- Loads which must be switched at high speed and duty cycle

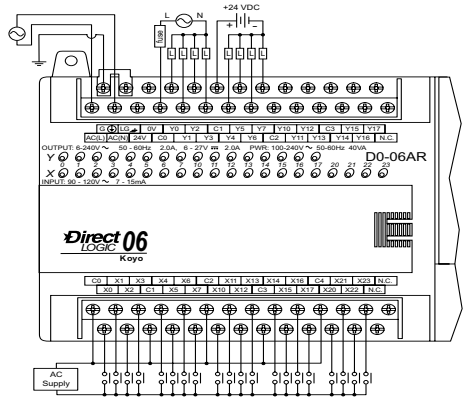
This section presents various ways to wire relay outputs to the loads. The relay output DL06s have sixteen normally-open SPST relays available. They are organized with four relays per common. The figure below shows the relays and the internal wiring of the PLC. Note that each group is isolated from the other group of outputs.

In the circuit below, all loads use the same AC power supply which powers the DL06 PLC. In this example, all commons are connected together.





In the circuit on the following page, loads for Y0 – Y3 use the same AC power supply which powers the DL06 PLC. Loads for Y4 – Y7 use a separate DC supply. In this example, the commons are separated according to which supply powers the associated load.



Relay Outputs – Transient Suppression for Inductive Loads in a Control System

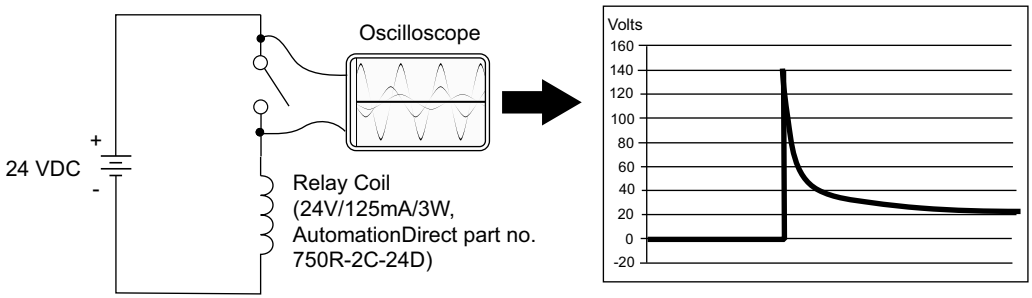
The following pages are intended to give a quick overview of the negative effects of transient voltages on a control system and provide some simple advice on how to effectively minimize them. The need for transient suppression is often not apparent to the newcomers in the automation world. Many mysterious errors that can afflict an installation can be traced back to a lack of transient suppression.

What is a Transient Voltage and Why is it Bad?

Inductive loads (devices with a coil) generate transient voltages as they transition from being energized to being de-energized. If not suppressed, the transient can be many times greater than the voltage applied to the coil. These transient voltages can damage PLC outputs or other electronic devices connected to the circuit, and cause unreliable operation of other electronics in the general area. Transients must be managed with suppressors for long component life and reliable operation of the control system.

This example shows a simple circuit with a small 24V/125mA /3W relay. As you can see, when the switch is opened, thereby de-energizing the coil, the transient voltage generated across the switch contacts peaks at 140V.

Example: Circuit with no Suppression



In the same circuit on the previous page, replacing the relay with a larger 24V/290mA/7W relay will generate a transient voltage exceeding 800V (not shown). Transient voltages like this can cause many problems, including:

- Relay contacts driving the coil may experience arcing, which can pit the contacts and reduce the relay's lifespan.
- Solid state (transistor) outputs driving the coil can be damaged if the transient voltage exceeds the transistor's ratings. In extreme cases, complete failure of the output can occur the very first time a coil is de-energized.
- Input circuits, which might be connected to monitor the coil or the output driver, can also be damaged by the transient voltage.

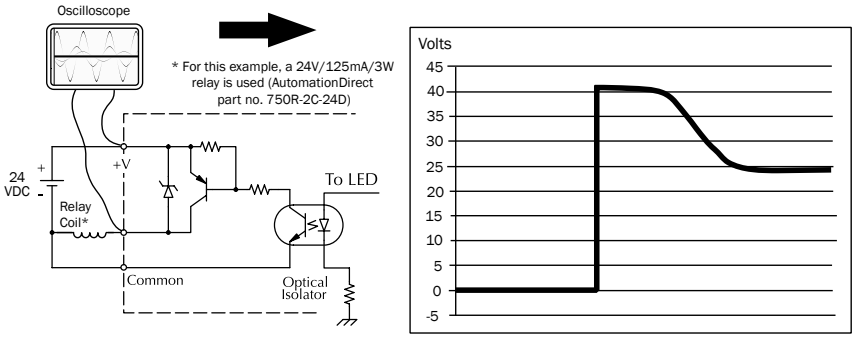
A very destructive side-effect of the arcing across relay contacts is the electromagnetic interference (EMI) it can cause. This occurs because the arcing causes a current surge, which releases RF energy. The entire length of wire between the relay contacts, the coil, and the power source carries the current surge and becomes an antenna that radiates the RF energy. It will readily couple into parallel wiring and may disrupt the PLC and other electronics in the area. This EMI can make an otherwise stable control system behave unpredictably at times.

PLC's Integrated Transient Suppressors

Although the PLC's outputs typically have integrated suppressors to protect against transients, they are not capable of handling them all. It is usually necessary to have some additional transient suppression for an inductive load.

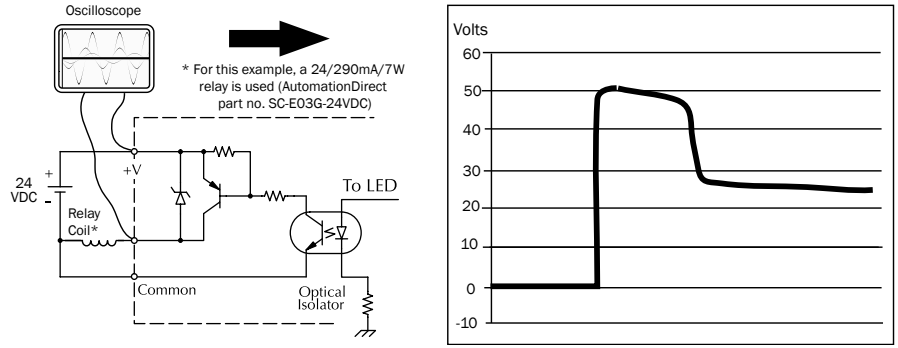
Here is another example using the same 24V / 125mA / 3W relay used earlier. This example measures the PNP transistor output of a D0-06DD2 PLC, which incorporates an integrated Zener diode for transient suppression. Instead of the 140V peak in the first example, the transient voltage here is limited to about 40V by the Zener diode. While the PLC will probably tolerate repeated transients in this range for some time, the 40V is still beyond the module's peak output voltage rating of 30V.

Example: Small Inductive Load with Only Integrated Suppression



The next example uses the same circuit as above, but with a larger 24V / 290mA / 7W relay, thereby creating a larger inductive load. As you can see, the transient voltage generated is much worse, peaking at over 50V. Driving an inductive load of this size without additional transient suppression is very likely to permanently damage the PLC output.

Example: Larger Inductive Load with Only Integrated Suppression

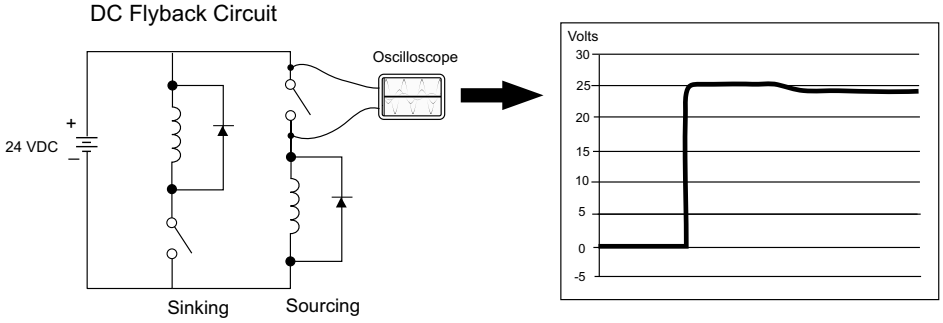


Additional transient suppression should be used in both these examples. If you are unable to measure the transients generated by the connected loads of your control system, using additional transient suppression on all inductive loads would be the safest practice.

Types of Additional Transient Protection

DC Coils:

The most effective protection against transients from a DC coil is a flyback diode. A flyback diode can reduce the transient to roughly 1V over the supply voltage, as shown in this example.



Many AutomationDirect socketed relays and motor starters have add-on flyback diodes that plug or screw into the base, such as the AD-ASMD-250 protection diode module and 784-4C-SKT-1 socket module shown below. If an add-on flyback diode is not available for your inductive load, an easy way to add one is to use AutomationDirect's DN-D10DR-A diode terminal block, a 600VDC power diode mounted in a slim DIN rail housing.

AD-ASMD-250
Protection Diode Module

784-4C-SKT-1
Relay Socket

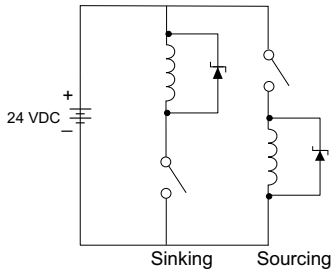
DN-D10DR-A
Diode Terminal Block

Two more common options for DC coils are Metal Oxide Varistors (MOV) or TVS diodes. These devices should be connected across the driver (PLC output) for best protection as shown below. The optimum voltage rating for the suppressor is the lowest rated voltage available that will NOT conduct at the supply voltage, while allowing a safe margin.

AutomationDirect's ZL-TSD8-24 transorb module is a good choice for 24VDC circuits. It is a bank of 8 uni-directional 30V TVS diodes. Since they are uni-directional, be sure to observe the polarity during installation. MOVs or bi-directional TVS diodes would install at the same location, but have no polarity concerns.

ZL-TSD8-24
Transorb Module

DC MOV or TVS Diode Circuit



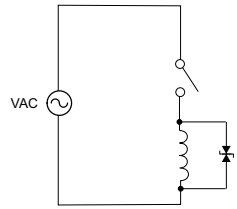
AC Coils:

Two options for AC coils are MOVs or bi-directional TVS diodes. These devices are most effective at protecting the driver from a transient voltage when connected across the driver (PLC output) but are also commonly connected across the coil. The optimum voltage rating for the suppressor is the lowest rated voltage available that will NOT conduct at the supply voltage, while allowing a safe margin.

AutomationDirect's ZL-TSD8-120 transorb module is a good choice for 120VAC circuits. It is a bank of eight bi-directional 180V TVS diodes.

ZL-TSD8-120
Transorb Module

AC MOV or Bi-Directional Diode Circuit



NOTE: *Manufacturers of devices with coils frequently offer MOV or TVS diode suppressors as an add-on option which mount conveniently across the coil. Before using them, carefully check the suppressor ratings. Just because the suppressor is made specifically for that part does not mean it will reduce the transient voltages to an acceptable level.*

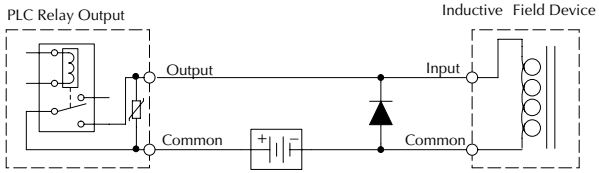
For example, a MOV or TVS diode rated for use on 24–48 VDC coils would need to have a high enough voltage rating to NOT conduct at 48V. That suppressor might typically start conducting at roughly 60VDC. If it were mounted across a 24V coil, transients of roughly 84V (if sinking output) or -60V (if sourcing output) could reach the PLC output. Many semiconductor PLC outputs cannot tolerate such levels.

Prolonging Relay Contact Life

Relay contacts wear according to the amount of relay switching, amount of spark created at the time of open or closure, and presence of airborne contaminants. There are some steps you can take to help prolong the life of relay contacts, such as switching the relay on or off only when it is necessary, and if possible, switching the load on or off at a time when it will draw the least current. Also, take measures to suppress inductive voltage spikes from inductive DC loads such as contactors and solenoids.

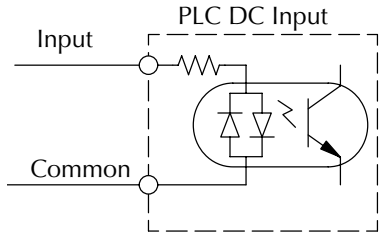
For inductive loads in DC circuits we recommend using a suppression diode as shown in the following diagram (DO NOT use this circuit with an AC power supply). When the load is energized the diode is reverse-biased (high impedance). When the load is turned off, energy stored in its coil is released in the form of a negative-going voltage spike. At this moment the diode is forward-biased (low impedance) and shunts the energy to ground. This protects the relay contacts from the high voltage arc that would occur just as the contacts are opening.

Place the diode as close to the inductive field device as possible. Use a diode with a peak inverse voltage rating (PIV) at least 100 PIV, 3A forward current or larger. Use a fast-recovery type (such as Schottky type). DO NOT use a small-signal diode such as 1N914, 1N941, etc. Be sure the diode is in the circuit correctly before operation. If installed backwards, it short-circuits the supply when the relay energizes.

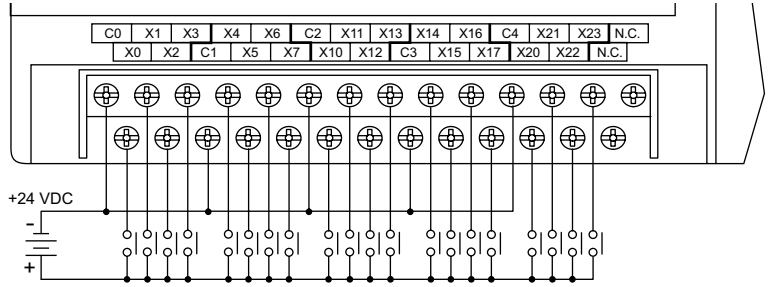


DC Input Wiring Methods

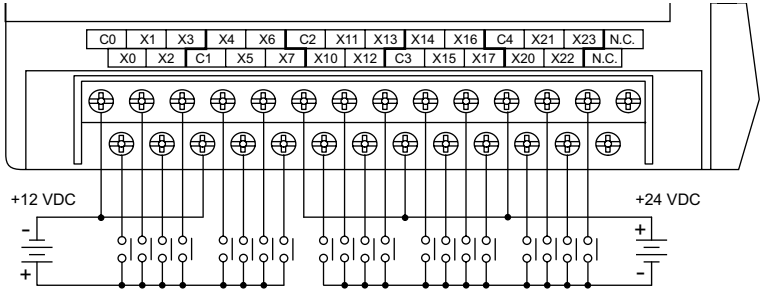
DL06 Micro PLCs with DC inputs are particularly flexible because they can be wired as either sinking or sourcing. The dual diodes (shown to the right) allow 10.8–26.4 VDC. The target applications are +12VDC and +24VDC. You can actually wire each group of inputs associated common group of inputs as DC sinking and the other half as DC sourcing. Inputs grouped by a common must be all sinking or all sourcing.



In the first and simplest example below, all commons are connected together and all inputs are sinking.



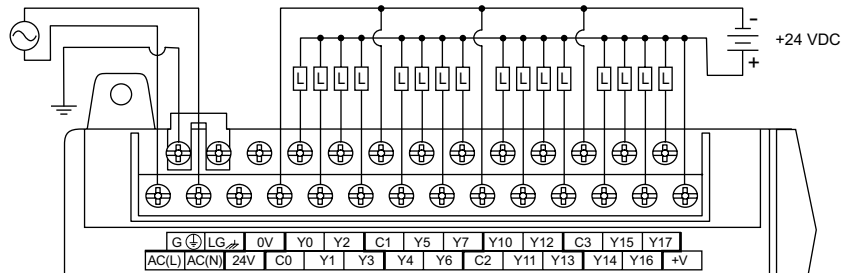
In the next example, the first eight inputs are sinking, and the last twelve are sourcing.



DC Output Wiring Methods

DL06 DC output circuits are high-performance transistor switches with low on-resistance and fast switching times. Please note the following characteristics which are unique to the DC output type:

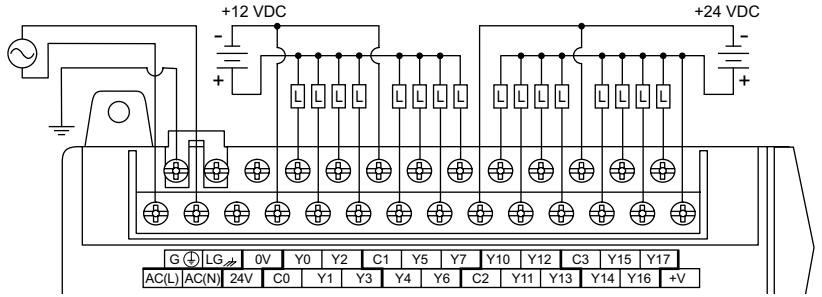
- There is only one electrical common for all sixteen outputs. All sixteen outputs belong to one bank.
- The output switches are current-sinking only or current sourcing only. Refer to the detailed specifications in this manual to determine which type output is present on a particular model.
- The output circuit inside the PLC requires external power. The supply (–) must be connected to a common terminal, and the supply (+) connects the right-most terminal on the upper connector (+V).



In the example below, all sixteen outputs share a common supply.

In the next example below, the outputs have **split** supplies. The first eight outputs are using a +12 VDC supply, and the last eight are using a +24VDC supply. However, you can split the outputs among any number of supplies, as long as:

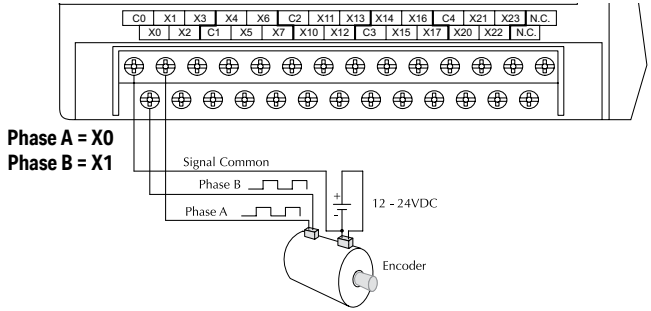
- All supply voltages are within the specified range
- All output points are wired as sinking
- All source (–) terminals are connected together



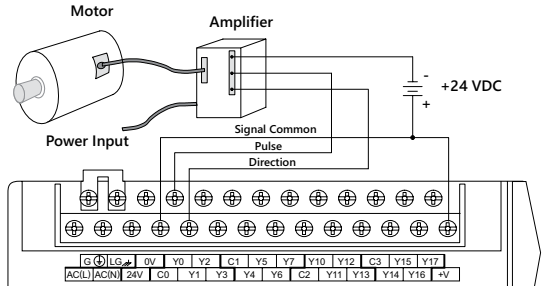
Warning: The maximum output current from the Auxiliary 24VDC power depends on the I/O configuration. Refer to Chapter 4, page 4-6, to determine how much current can be drawn from the Auxiliary 24VDC power for your particular I/O configuration.

High-Speed I/O Wiring Methods

DL06 versions with DC type input or output points contain a dedicated High-Speed I/O circuit (HSIO). The circuit configuration is programmable, and it processes specific I/O points independently from the CPU scan. Appendix E discusses the programming options for HSIO. While the HSIO circuit has six modes, we show wiring diagrams for two of the most popular modes in this chapter. The high-speed input interfaces to points X0–X3. Properly configured, the DL06 can count quadrature pulses at up to 7kHz from an incremental encoder as shown below.



NOTE: *Do not use this drawing to wire your device. This is a general example and is not specific to any PLC model, stepper or encoder. Always refer to the device documentation for proper wiring connections.*



DL06 versions with DC type output points can use the High Speed I/O Pulse Output feature. It can generate high-speed pulses at up to 10 kHz for specialized control such as stepper motor / intelligent drive systems. Output Y0 and Y1 can generate pulse and direction signals, or it can generate CCW and CW pulse signals respectively. See Appendix E on high-speed input and pulse output options.

NOTE: *Do not use this drawing to wire your device. This is a general example and is not specific to any PLC model, stepper or encoder. Always refer to the device documentation for proper wiring connections.*

Wiring Diagrams and Specifications

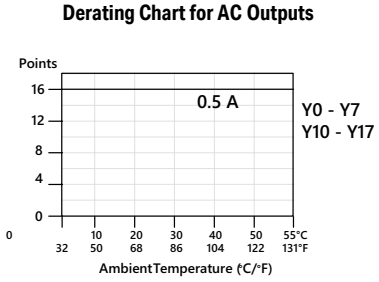
The remainder of this chapter provides detailed technical information for the DL06 PLCs. A basic wiring diagram, equivalent I/O circuits, and specification tables are laid out for each PLC.

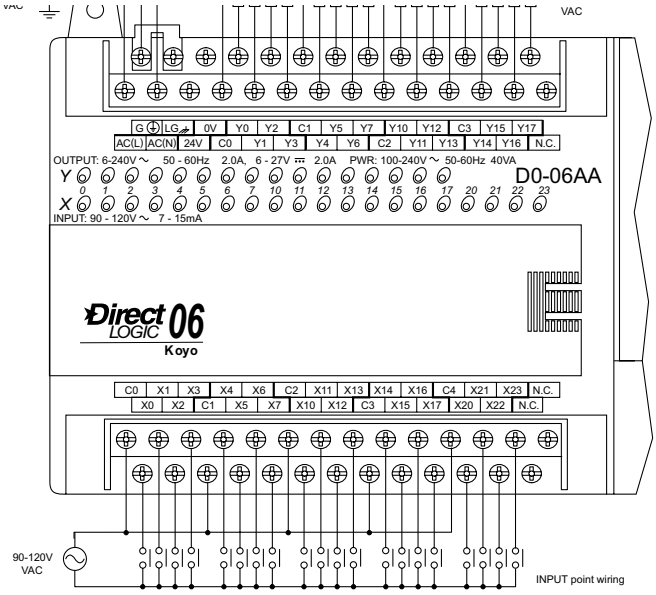
D0–06AA I/O Wiring Diagram

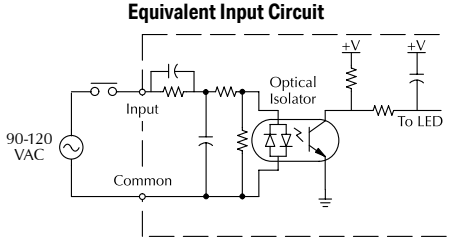
The D0–06AA PLC has twenty AC inputs and sixteen AC outputs. The following diagram shows a typical field wiring example. The AC external power connection uses four terminals as shown.

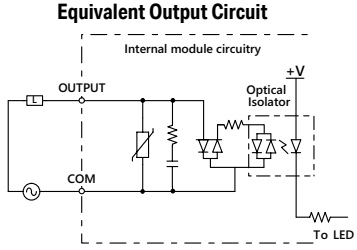
Inputs are organized into five banks of four. Each bank has an isolated common terminal. The wiring example below shows all commons connected together, but separate supplies and common circuits may be used. The equivalent input circuit shows one channel of a typical bank.

Outputs are organized into four banks of four triac switches. Each bank has a common terminal. The wiring example below shows all commons connected together, but separate supplies and common circuits may be used. The equivalent output circuit shows one channel of a typical bank.









D0-06AA General Specifications	
External Power Requirements	100–240 VAC/50–60 Hz, 40VA maximum
Communication Port 1 9600 baud (Fixed), 8 data bits, 1 stop bit odd parity	K–Sequence (Slave), DirectNET (Slave), MODBUS (Slave)
Communication Port 2 9600 baud (default) 8 data bits, 1 stop bit odd parity	K–Sequence (Slave),DirectNET (Master/Slave), MODBUS (Master/Slave), Non-sequence / print, ASCII in/out
Programming cable type	D2–DSCBL
Operating Temperature	32 to 131°F (0 to 55°C)
Storage Temperature	–4 to 158°F (–20 to 70°C)
Relative Humidity	5 to 95% (non-condensing)
Environmental air	No corrosive gases permitted
Vibration	MIL STD 810C 514.2
Shock	MIL STD 810C 516.2
Noise Immunity	NEMA ICS3–304
Terminal Type	Removable
Wire Gauge	One 16 AWG or two 18 AWG, 24 AWG minimum

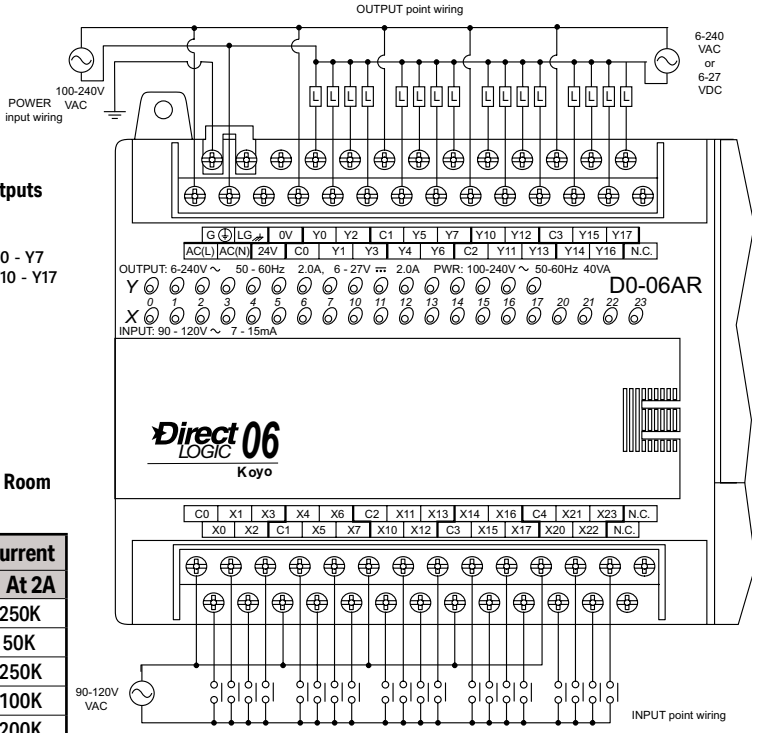
AC Input Specifications	
Input Voltage Range (Min. - Max.)	80–132 VAC, 47–63 Hz
Operating Voltage Range	90–120 VAC, 47–63 Hz
Input Current	8mA @100VAC at 50Hz 10mA @100VAC at 60Hz
Max. Input Current	12mA @132VAC at 50 Hz 15mA @132VAC at 60 Hz
Input Impedance	14KΩ @50 Hz, 12KΩ @60Hz
ON Current/Voltage	> 6mA @ 75VAC
OFF Current/Voltage	< 2mA @ 20VAC
OFF to ON Response	< 40ms
ON to OFF Response	< 40ms
Status Indicators	Logic Side
Commons	4 channels / common x 5 banks (isolated)

AC Output Specifications	
Output Voltage Range (Min. - Max.)	15–264 VAC, 47–63 Hz
Operating Voltage	17–240 VAC, 47–63 Hz
On Voltage Drop	1.5 VAC (>50mA) 4.0 VAC (<50mA)
Max Current	0.5 A / point, 1.5 A / common
Max leakage current	<4mA @ 264VAC
Max inrush current	10A for 10ms
Minimum Load	10mA
OFF to ON Response	1ms
ON to OFF Response	1 ms +1/2 cycle
Status Indicators	Logic Side
Commons	4 channels / common x 4 banks (isolated)
Fuses	None (external recommended)

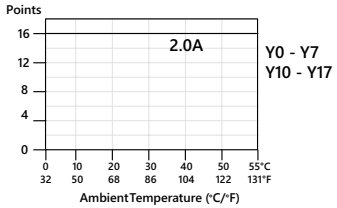
D0–06AR I/O Wiring Diagram

The D0–06AR PLC has twenty AC inputs and sixteen relay contact outputs. The following diagram shows a typical field wiring example. The AC external power connection uses four terminals at the left as shown.

The twenty AC input channels use terminals on the bottom of the connector. Inputs are organized into five banks of four. Each bank has a common terminal. The wiring example below shows all commons connected together, but separate supplies and common circuits may be used. The equivalent input circuit shows one channel of a typical bank.



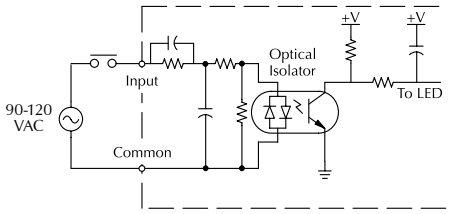
Derating Chart for Relay Outputs



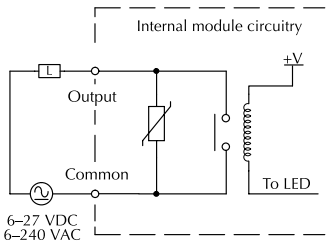
Typical Relay Life (Operations) at Room Temperature

Voltage & Load Type	Load Current	
	At 1A	At 2A
24VDC Resistive	500K	250K
24VDC Inductive	100K	50K
110VAC Resistive	500K	250K
110VAC Inductive	200K	100K
220VAC Resistive	350K	200K
220VAC Inductive	100K	50K

Equivalent Input Circuit



Equivalent Output Circuit



The sixteen relay output channels use terminals on the right side top connector. Outputs are organized into four banks of four normally-open relay contacts. Each bank has a common terminal. The wiring example on the last page shows all commons connected together, but separate supplies and common circuits may be used. The equivalent output circuit shows one channel of a typical bank. The relay contacts can switch AC or DC voltages.

D0-06AR General Specifications	
External Power Requirements	100– 240 VAC/ 50–60 Hz, 40VA maximum
Communication Port 1 9600 baud (Fixed), 8 data bits, 1 stop bit, odd parity	K–Sequence (Slave), DirectNET (Slave), MODBUS (Slave)
Communication Port 2 9600 baud (default), 8 data bits, 1 stop bit, odd parity	K–Sequence (Slave), DirectNET (Master/Slave), MODBUS (Master/Slave), Non-sequence / print, ASCII in/out
Programming cable type	D2–DSCBL
Operating Temperature	32 to 131°F (0 to 55°C)
Storage Temperature	–4 to 158°F (–20 to 70°C)
Relative Humidity	5 to 95% (non-condensing)
Environmental air	No corrosive gases permitted
Vibration	MIL STD 810C 514.2
Shock	MIL STD 810C 516.2
Noise Immunity	NEMA ICS3–304
Terminal Type	Removable
Wire Gauge	One 16 AWG or two 18 AWG, 24 AWG minimum

AC Input Specifications X0-X23	
Input Voltage Range (Min. - Max.)	80–132 VAC, 47–63 Hz
Operating Voltage Range	90–120 VAC, 47–63 Hz
Input Current	8mA @ 100VAC at 50Hz 10mA @ 100VAC at 60Hz
Max. Input Current	12mA @ 132VAC at 50Hz 15mA @ 132VAC at 60Hz
Input Impedance	14KΩ @50Hz, 12KΩ @60Hz
ON Current/Voltage	>6mA @ 75VAC
OFF Current/Voltage	<2mA @ 20VAC
OFF to ON Response	< 40ms
ON to OFF Response	< 40ms
Status Indicators	Logic Side
Commons	4 channels / common x 5 banks (isolated)

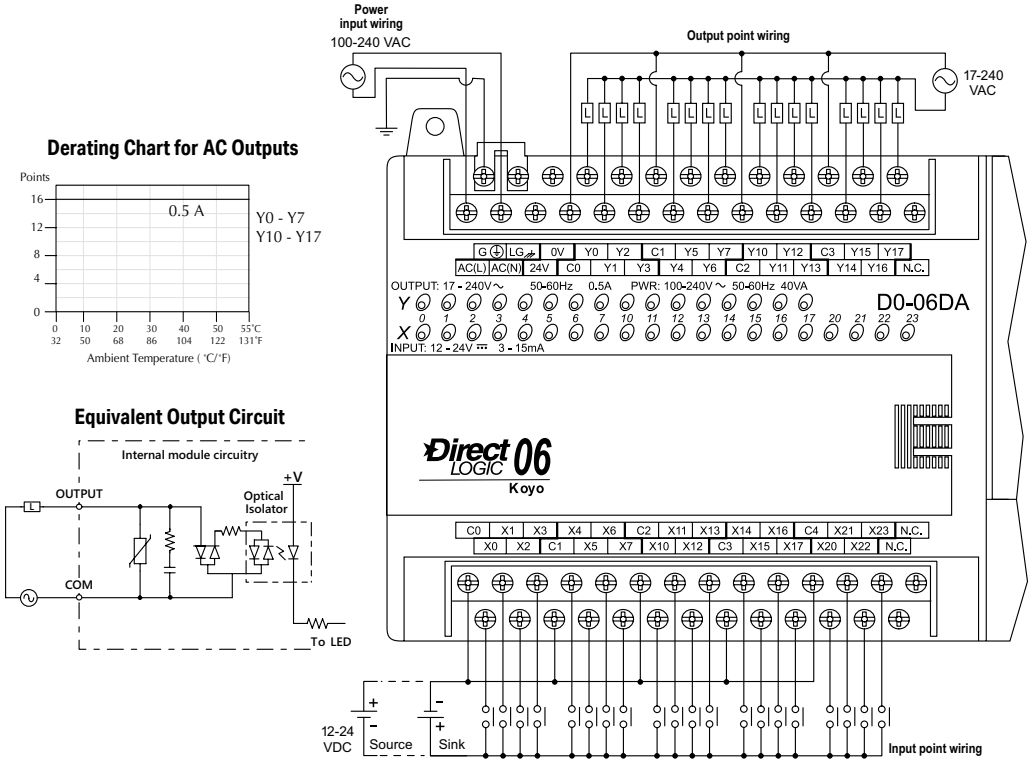
Relay Output Specifications Y0-Y17	
Output Voltage Range	(Min. – Max.) 5–264 VAC (47–63 Hz), 5–30 VDC
Operating Voltage Range	6–240 VAC (47–63 Hz), 6–27 VDC
Output Current	2A / point, 6A / common
Max. leakage current	0.1 mA @ 264VAC
Smallest Recommended Load	5mA @ 5VDC
OFF to ON Response	< 15ms
ON to OFF Response	< 10ms
Status Indicators	Logic Side
Commons	4 channels / common x 4 banks (isolated)
Fuses	None (external recommended)

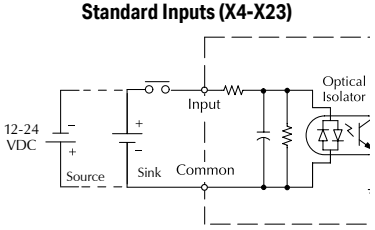
D0–06DA I/O Wiring Diagram

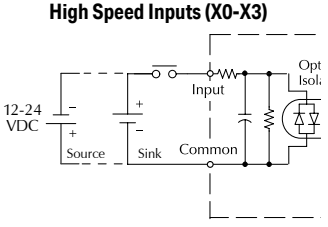
The D0–06DA PLC has twenty DC inputs and sixteen AC outputs. The following diagram shows a typical field wiring example. The AC external power connection uses four terminals as shown.

Inputs are organized into five banks of four. Each bank has an isolated common terminal, and may be wired as sinking or sourcing. The wiring example below shows all commons connected together, but separate supplies and common circuits may be used. The equivalent circuit for standard inputs is shown below, and the high-speed input circuit is shown to the left.

Outputs are organized into four banks of four triac switches. Each bank has a common terminal. The wiring example below shows all commons connected together, but separate supplies and common circuits may be used. The equivalent output circuit shows one channel of a typical bank.







D0-06DA General Specifications	
External Power Requirements	100–240 VAC/ 50–60 Hz, 40VA maximum
Communication Port 1 9600 baud (Fixed), 8 data bits, 1 stop bit, odd parity	K–Sequence (Slave), DirectNET (Slave), MODBUS (Slave)
Communication Port 2 9600 baud (default), 8 data bits, 1 stop bit, odd parity	K–Sequence (Slave), DirectNET (Master/Slave), MODBUS (Master/Slave), Non-sequence/print, ASCII in/out
Programming cable type	D2–DSCBL
Operating Temperature	32 to 131°F (0 to 55°C)
Storage Temperature	–4 to 158°F (–20 to 70°C)
Relative Humidity	5 to 95% (non-condensing)
Environmental air	No corrosive gases permitted
Vibration	MIL STD 810C 514.2
Shock	MIL STD 810C 516.2
Noise Immunity	NEMA ICS3–304
Terminal Type	Removable
Wire Gauge	One 16 AWG or two 18 AWG, 24 AWG minimum

DC Input Specifications		
Parameter	High–Speed Inputs, X0 – X3	Standard DC Inputs X4 – X23
Input Voltage Range	10.8–26.4 VDC	10.8–26.4 VDC
Operating Voltage Range	12–24 VDC	12–24 VDC
Maximum Voltage	30VDC (7kHz maximum frequency)	30VDC
Minimum Pulse Width	70µs	N/A
ON Voltage Level	> 10VDC	> 10VDC
OFF Voltage Level	< 2.0 VDC	< 2.0 VDC
Input Impedance	1.8 kΩ @ 12–24 VDC	2.8 kΩ @ 12–24 VDC
Minimum ON Current	>5mA	>4mA
Maximum OFF Current	< 0.5 mA	<0.5 mA
OFF to ON Response	<70µs	2–8 ms, 4ms typical
ON to OFF Response	<70µs	2–8 ms, 4ms typical
Status Indicators	Logic side	Logic side
Commons	4 channels / common x 5 bank (isolated)	

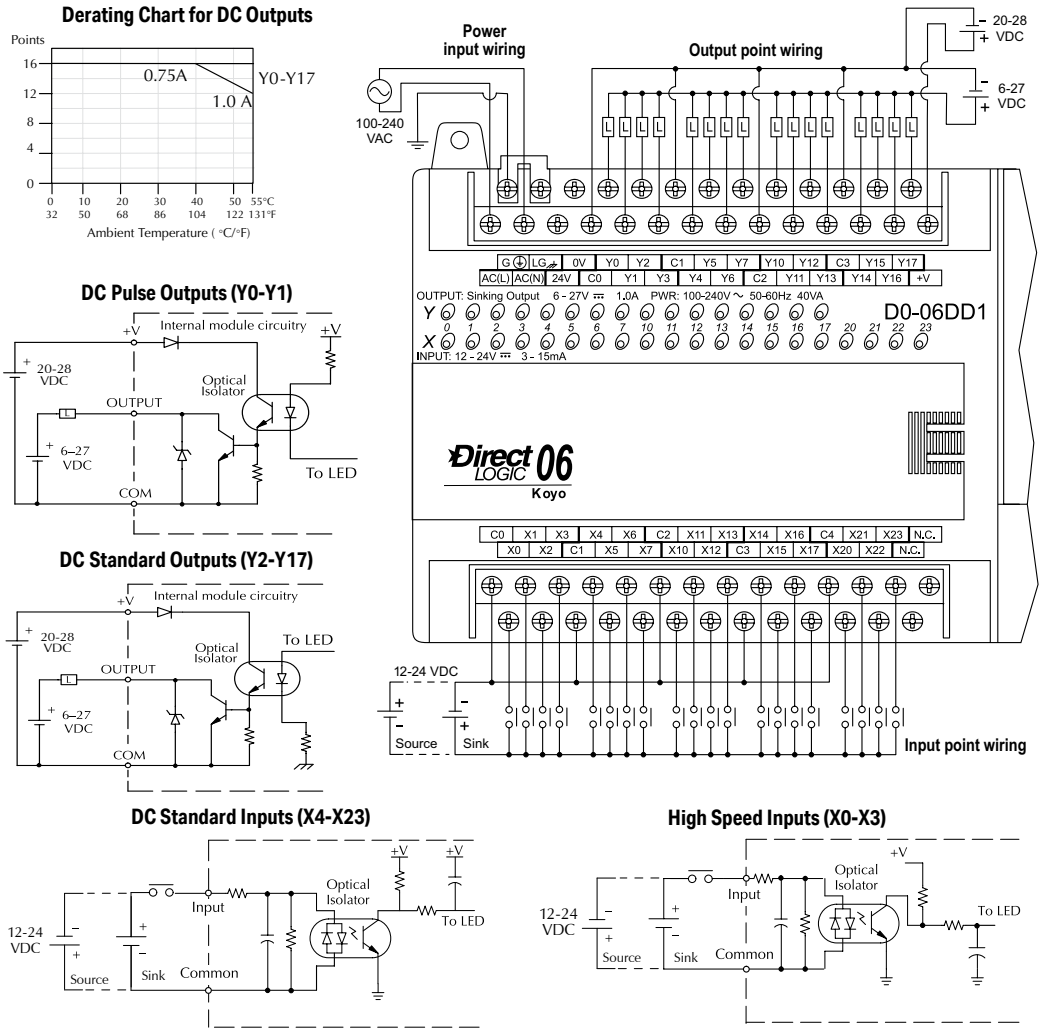
AC Output Specifications	
Output Voltage Range (Min. - Max.)	15–264 VAC, 47–63 Hz
Operating Voltage	17–240 VAC, 47–63 Hz
On Voltage Drop	1.5 VAC @> 50mA, 4VAC @< 50mA
Max Current	0.5 A / point, 1.5 A / common
Max leakage current	< 4mA @ 264VAC, 60Hz
Max inrush current	10A for 10ms
Minimum Load	10mA
OFF to ON Response	1ms
ON to OFF Response	1 ms +1/2 cycle
Status Indicators	Logic Side
Commons	4 channels / common x 4 banks (isolated)
Fuses	None (external recommended)

D0–06DD1 I/O Wiring Diagram

The D0-06DD1 PLC has twenty sinking/sourcing DC inputs and sixteen sinking DC outputs. The following diagram shows a typical field wiring example. The AC external power connection uses four terminals as shown.

Inputs are organized into five banks of four. Each bank has an isolated common terminal, and may be wired as either sinking or sourcing inputs. The wiring example below shows all commons connected together, but separate supplies and common circuits may be used.

Outputs all share the same common. Note the requirement for external power.



D0-06DD1 General Specifications	
External Power Requirements	100–240 VAC/ 50–60 Hz, 40VA maximum
Communication Port 1 9600 baud (Fixed), 8 data bits, 1 stop bit, odd parity	K–Sequence (Slave), DirectNET (Slave), MODBUS (Slave)
Communication Port 2 9600 baud (default), 8 data bits, 1 stop bit, odd parity	K–Sequence (Slave), DirectNET (Master/Slave), MODBUS (Master/Slave), Non-sequence / print, ASCII in/out
Programming cable type	D2–DSCBL
Operating Temperature	32 to 131°F (0 to 55°C)
Storage Temperature	–4 to 158°F (–20 to 70°C)
Relative Humidity	5 to 95% (non-condensing)
Environmental air	No corrosive gases permitted
Vibration	MIL STD 810C 514.2
Shock	MIL STD 810C 516.2
Noise Immunity	NEMA ICS3–304
Terminal Type	Removable
Wire Gauge	One 16 AWG or two 18 AWG, 24 AWG minimum

DC Input Specifications		
Parameter	High–Speed Inputs, X0 – X3	Standard DC Inputs X4 – X23
Min. - Max. Voltage Range	10.8–26.4 VDC	10.8–26.4 VDC
Operating Voltage Range	12–24 VDC	12–24 VDC
Peak Voltage	30VDC (7kHz maximum frequency)	30VDC
Minimum Pulse Width	100µs	N/A
ON Voltage Level	> 10.0 VDC	> 10.0 VDC
OFF Voltage Level	< 2.0 VDC	< 2.0 VDC
Max. Input Current	6mA @12VDC, 13mA @24VDC	4mA @12VDC, 8.5mA @24VDC
Input Impedance	1.8 Ωk @ 12–24 VDC	2.8 Ωk @ 12–24 VDC
Minimum ON Current	>5mA	>4mA
Maximum OFF Current	< 0.5 mA	<0.5 mA
OFF to ON Response	<70µs	2–8 ms, 4ms typical
ON to OFF Response	<70µs	2–8 ms, 4ms typical
Status Indicators	Logic side	Logic side
Commons	4 channels / common x 5 banks isolated	

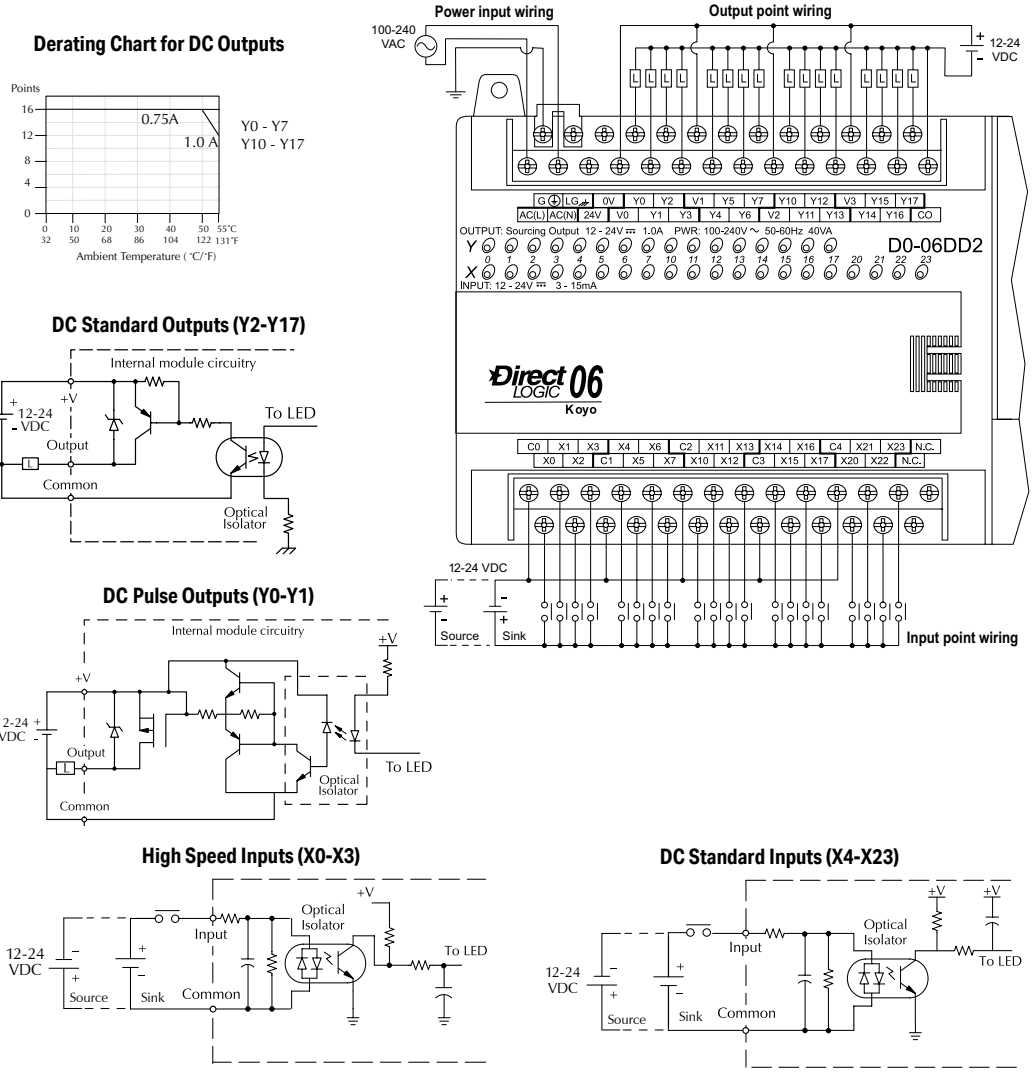
DC Output Specifications		
Parameter	Pulse Outputs Y0 – Y1	Standard Outputs Y2 – Y17
Min. - Max. Voltage Range	5–30 VDC	5–30 VDC
Operating Voltage	6–27 VDC	6–27 VDC
Peak Voltage	< 50VDC (10kHz max. frequency)	< 50VDC
On Voltage Drop	0.3 VDC @ 1 A	0.3 VDC @ 1A
Max Current (resistive)	0.5 A / pt., 1A / pt. as standard pt.	1.0 A / point
Max leakage current	15µA @ 30VDC	15µA @ 30VDC
Max inrush current	2A for 100ms	2A for 100ms
External DC power required	20–28 VDC Max 150mA	20–28 VDC Max 280mA (Aux. 24VDC powers V+ terminal (sinking outputs)
OFF to ON Response	< 10µs	< 10µs
ON to OFF Response	< 20µs	< 60µs
Status Indicators	Logic Side	Logic Side
Commons	4 channels / common x 4 banks non-isolated	
Fuses	None (external recommended)	

D0–06DD2 I/O Wiring Diagram

The D0–06DD2 PLC has twenty sinking/sourcing DC inputs and sixteen sourcing DC outputs. The following diagram shows a typical field wiring example. The AC external power connection uses four terminals as shown.

Inputs are organized into four banks of four. Each bank has an isolated common terminal, and may be wired as either sinking or sourcing inputs. The wiring example below shows all commons connected together, but separate supplies and common circuits may be used.

All outputs share the same common. Note the requirement for external power.



D0-06DD2 General Specifications	
External Power Requirements	100–240 VAC/50–60 Hz, 40VA maximum
Communication Port 1 9600 baud (Fixed), 8 data bits, 1 stop bit, odd parity	K–Sequence (Slave), DirectNET (Slave), MODBUS (Slave)
Communication Port 2 9600 baud (default), 8 data bits, 1 stop bit, odd parity	K–Sequence (Slave), DirectNET (Master/Slave), MODBUS (Master/Slave), Non-sequence / print, ASCII in/out
Programming cable type	D2–DSCBL
Operating Temperature	32 to 131°F (0 to 55°C)
Storage Temperature	–4 to 158°F (–20 to 70°C)
Relative Humidity	5 to 95% (non-condensing)
Environmental air	No corrosive gases permitted
Vibration	MIL STD 810C 514.2
Shock	MIL STD 810C 516.2
Noise Immunity	NEMA ICS3–304
Terminal Type	Removable
Wire Gauge	One 16 AWG or two 18 AWG, 24 AWG minimum

DC Input Specifications		
Parameter	High–Speed Inputs, X0 – X3	Standard DC Inputs X4 – X23
Min. - Max. Voltage Range	10.8–26.4 VDC	10.8–26.4 VDC
Operating Voltage Range	12–24 VDC	12–24 VDC
Peak Voltage	30 VDC (7 kHz maximum frequency)	30VDC
Minimum Pulse Width	70 µs	N/A
ON Voltage Level	> 10.0 VDC	> 10.0 VDC
OFF Voltage Level	< 2.0 VDC	< 2.0 VDC
Max. Input Current	6mA @ 12VDC, 13mA @ 24VDC	4mA @ 12VDC, 8.5 mA @ 24VDC
Input Impedance	1.8 Ωk @ 12–24 VDC	2.8 Ωk @ 12–24 VDC
Minimum ON Current	>5mA	>4mA
Maximum OFF Current	< 0.5 mA	<0.5 mA
OFF to ON Response	<70µs	2–8 ms, 4ms typical
ON to OFF Response	<70µs	2–8 ms, 4ms typical
Status Indicators	Logic side	Logic side
Commons	4 channels/common x 5 banks (isolated)	

DC Output Specifications		
Parameter	Pulse Outputs Y0 – Y1	Standard Outputs Y2 – Y17
Min. - Max. Voltage Range	10.8–26.4 VDC	10.8–26.4 VDC
Operating Voltage	12–24 VDC	12–24 VDC
Peak Voltage	< 50VDC (10kHz max. frequency)	< 50VDC
On Voltage Drop	0.5 VDC @ 1A	1.2 VDC @ 1A
Max Current (resistive)	0.5 A / pt., 1A / pt. as standard pt.	1.0 A / point
Max leakage current	15µA @ 30VDC	15µA @ 30VDC
Max inrush current	2A for 100ms	2A for 100ms
External DC power required	12–24 VDC	12 -24 VDC
OFF to ON Response	< 10µs	< 10µs
ON to OFF Response	< 20µs	< 0.5 µs
Status Indicators	Logic Side	Logic Side
Commons	4 channels / common x 4 banks (non-isolated)	
Fuses	None (external recommended)	

D0–06DR I/O Wiring Diagram

The D0–06DR PLCs feature twenty DC inputs and sixteen relay contact outputs. The following diagram shows a typical field wiring example. The AC external power connection uses four terminals as shown.

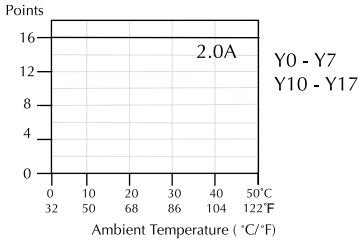
Inputs are organized into five banks of four. Each bank has an isolated common terminal, and may be wired as either sinking or sourcing inputs. The wiring example below shows all commons connected together, but separate supplies and common circuits may be used. The equivalent circuit for standard inputs is shown below, and the high-speed input circuit is shown to the left.

Outputs are organized into four banks of four normally-open relay contacts. Each bank has a common terminal. The wiring example below shows all commons connected together, but separate supplies and common circuits may be used. The equivalent output circuit shows one channel of a typical bank. The relay contacts can switch AC or DC voltages.

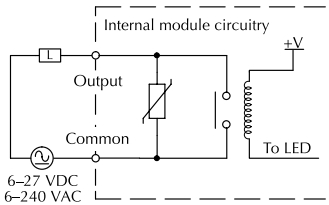
Typical Relay Life (Operations) at Room Temperature

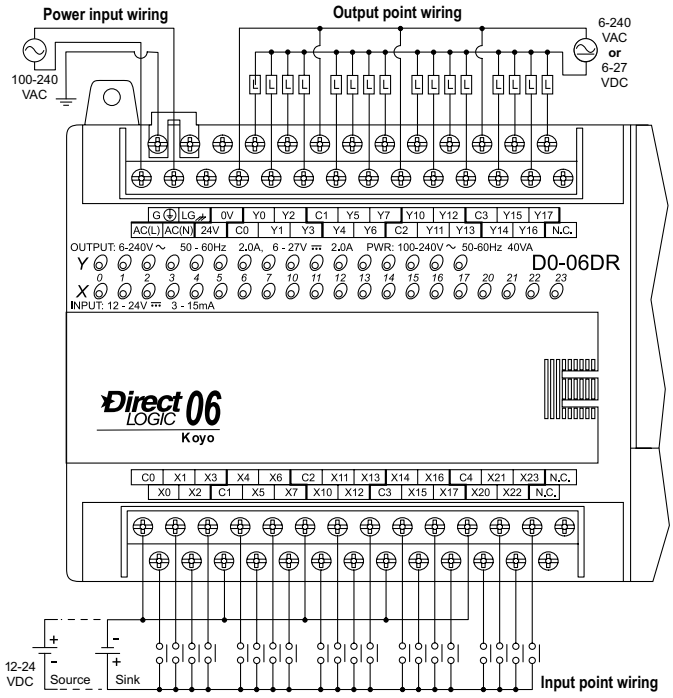
Voltage & Load Type	Load Current	
	At 1A	At 2A
24VDC Resistive	500K	250K
24VDC Inductive	100K	50K
110VAC Resistive	500K	250K
110VAC Inductive	200K	100K
220VAC Resistive	350K	200K
220VAC Inductive	100K	50K

Derating Chart for Relay Outputs

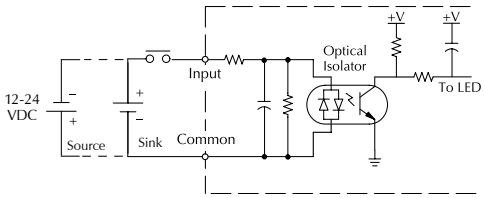


Equivalent Output Circuit

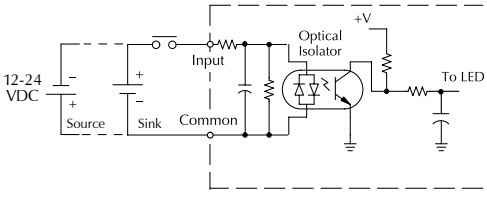




Equivalent Circuit, Standard Inputs (X4-X23)



Equivalent Circuit, High-speed Inputs (X0-X3)



D0-06DR General Specifications	
External Power Requirements	100–240 VAC/ 50–60 Hz, 40VA maximum
Communication Port 1 9600 baud (Fixed), 8 data bits, 1 stop bit, odd parity	K–Sequence (Slave), DirectNET (Slave), MODBUS (Slave)
Communication Port 2 9600 baud (default), 8 data bits, 1 stop bit, odd parity	K–Sequence (Slave), DirectNET (Master/Slave), MODBUS (Master/Slave), Non-sequence /print, ASCII in/out
Programming cable type	D2–DSCBL
Operating Temperature	32 to 131°F (0 to 55°C)
Storage Temperature	–4 to 158°F (–20 to 70°C)
Relative Humidity	5 to 95% (non-condensing)
Environmental air	No corrosive gases permitted
Vibration	MIL STD 810C 514.2
Shock	MIL STD 810C 516.2
Noise Immunity	NEMA ICS3–304
Terminal Type	Removable
Wire Gauge	One 16 AWG or two 18 AWG, 24 AWG minimum

DC Input Specifications		
Parameter	High–Speed Inputs, X0 – X3	Standard DC Inputs X4 – X23
Min. - Max. Voltage Range	10.8–26.4 VDC	10.8–26.4 VDC
Operating Voltage Range	12–24 VDC	12–24 VDC
Peak Voltage	30VDC (7kHz maximum frequency)	30VDC
Minimum Pulse Width	70µs	N/A
ON Voltage Level	> 10VDC	> 10VDC
OFF Voltage Level	< 2.0 VDC	< 2.0 VDC
Input Impedance	1.8 kΩ @ 12–24 VDC	2.8 kΩ @ 12–24 VDC
Max. Input Current	6mA @12VDC 13mA @ 24VDC	4mA @ 12VDC 8.5 mA @ 24VDC
Minimum ON Current	>5mA	>4mA
Maximum OFF Current	< 0.5 mA	<0.5 mA
OFF to ON Response	<70µs	2–8 ms, 4ms typical
ON to OFF Response	<70µs	2–8 ms, 4ms typical
Status Indicators	Logic side	Logic side
Commons	4 channels / common x 5 banks (isolated)	

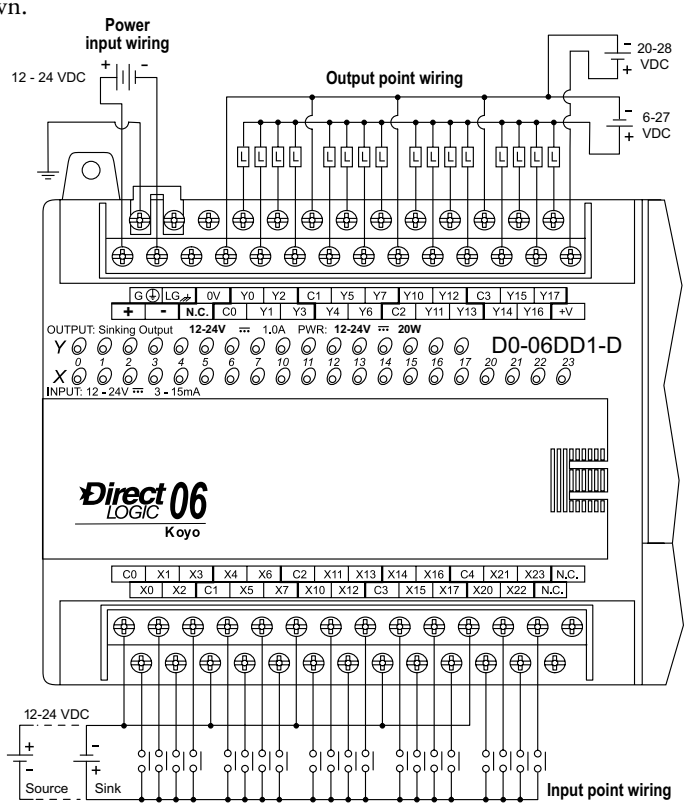
Relay Output Specifications	
Output Voltage Range (Min. - Max.)	5–264 VAC (47–63 Hz), 5–30 VDC
Operating Voltage	6–240 VAC (47–63 Hz), 6–27 VDC
Output Current	2A / point 6A / common
Maximum Voltage	264VAC, 30VDC
Max leakage current	0.1 mA @ 264VAC
Smallest Recommended Load	5mA
OFF to ON Response	< 15ms
ON to OFF Response	< 10ms
Status Indicators	Logic Side
Commons	4 channels / common x 4 banks (isolated)
Fuses	None (external recommended)

D0–06DD1–D I/O Wiring Diagram

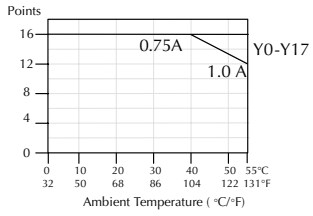
These micro PLCs feature twenty DC inputs and sixteen sinking DC outputs. The following diagram shows a typical field wiring example. The DC external power connection uses four terminals at the left as shown.

Inputs are organized into five banks of four. Each bank has an isolated common terminal, and may be wired as either sinking or sourcing inputs. The wiring example below shows all commons connected together, but separate supplies and common circuits may be used.

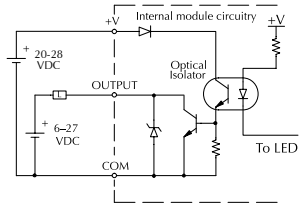
All outputs actually share the same common. Note the requirement for external power.



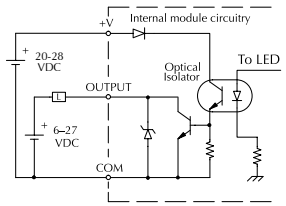
Derating Chart for DC Outputs



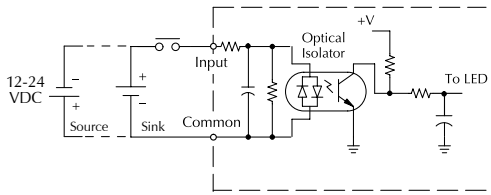
DC Pulse Outputs (Y0-Y1)



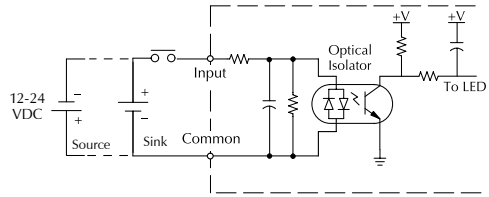
DC Standard Outputs (Y2-Y17)



High Speed Inputs (X0-X3)



Standard Input Circuit (X4-X23)



D0-06DD1-D General Specifications	
External Power Requirements	12–24 VDC, 20W maximum,
Communication Port 1: 9600 baud (Fixed), 8 data bits, 1 stop bit, odd parity	K–Sequence (Slave), DirectNET (Slave), MODBUS (Slave)
Communication Port 2: 9600 baud (default), 8 data bits, 1 stop bit,odd parity	K–Sequence (Slave), DirectNET (Master/Slave), MODBUS (Master/Slave), Non-sequence/print, ASCII in/out
Programming cable type	D2–DSCBL
Operating Temperature	32 to 131°F (0 to 55°C)
Storage Temperature	–4 to 158°F (–20 to 70°C)
Relative Humidity	5 to 95% (non-condensing)
Environmental air	No corrosive gases permitted
Vibration	MIL STD 810C 514.2
Shock	MIL STD 810C 516.2
Noise Immunity	NEMA ICS3–304
Terminal Type	Removable
Wire Gauge	One 16 AWG or two 18 AWG, 24 AWG minimum

DC Input Specifications		
Parameter	High–Speed Inputs, X0 – X3	Standard DC Inputs X4 – X23
Min. - Max. Voltage Range	10.8–26.4 VDC	10.8–26.4 VDC
Operating Voltage Range	12–24 VDC	12–24 VDC
Peak Voltage	30VDC (7kHz maximum frequency)	30VDC
Minimum Pulse Width	70µs	N/A
ON Voltage Level	>10.0 VDC	> 10.0 VDC
OFF Voltage Level	< 2.0 VDC	< 2.0 VDC
Max. Input Current	6mA @12VDC, 13mA @ 24VDC	4mA @12VDC, 8.5 mA @ 24VDC
Input Impedance	1.8 kΩ @ 12–24VDC	2.8 kΩ @ 12–24 VDC
Minimum ON Current	>5mA	>4mA
Maximum OFF Current	< 0.5 mA	<0.5 mA
OFF to ON Response	<70µs	2–8 ms, 4ms typical
ON to OFF Response	<70µs	2–8 ms, 4 ms typical
Status Indicators	Logic side	Logic side
Commons	4 channels / common x 5 banks (isolated)	

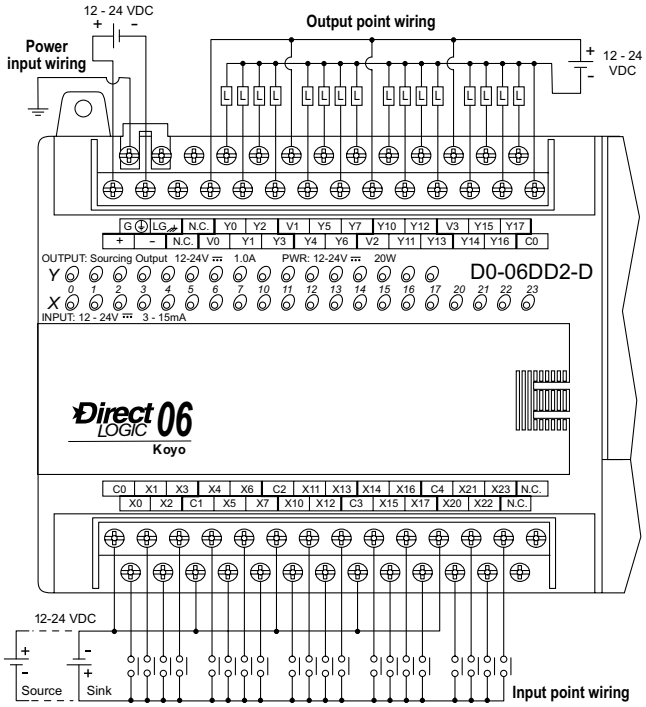
DC Output Specifications		
Parameter	Pulse Outputs, Y0 – Y1	Standard Outputs, Y2 – Y17
Min. - Max. Voltage Range	5–30 VDC	5–30 VDC
Operating Voltage	6–27 VDC	6–27 VDC
Peak Voltage	< 50VDC (10kHz max. frequency)	< 50VDC
On Voltage Drop	0.3 VDC @ 1 A	0.3 VDC @ 1A
Max Current (resistive)	0.5 A / pt., 1A / pt. as standard pt.	1.0 A / point
Max leakage current	15µA @ 30VDC	15µA @ 30VDC
Max inrush current	2 A for 100 ms	2A for 100ms
External DC power required	20–28 VDC Max 150mA	20–28 VDC Max 150mA
OFF to ON Response	< 10µs	< 10µs
ON to OFF Response	< 20µs	< 60µs
Status Indicators	Logic Side	Logic Side
Commons	4 channels / common x 4 banks (non-isolated)	
Fuses	None (external recommended)	

D0–06DD2–D I/O Wiring Diagram

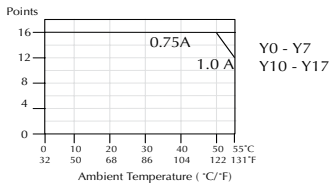
These micro PLCs feature twenty DC inputs and sixteen sourcing DC outputs. The following diagram shows a typical field wiring example. The DC external power connection uses four terminals at the left as shown.

Inputs are organized into five banks of four. Each bank has an isolated common terminal, and may be wired as either sinking or sourcing inputs. The wiring example below shows all commons connected together, but separate supplies and common circuits may be used.

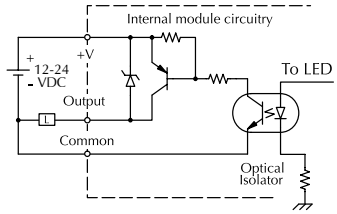
All outputs actually share the same common. Note the requirement for external power.



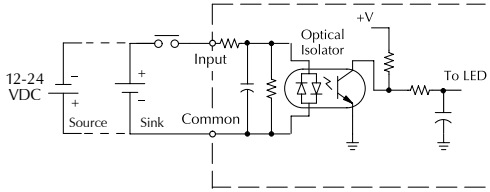
Derating Chart for DC Outputs



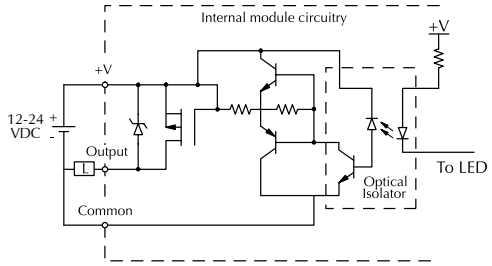
DC Standard Outputs (Y2-Y17)



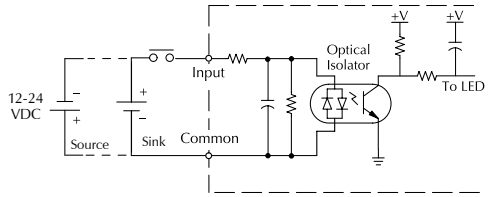
High Speed Inputs (X0-X3)



DC Pulse Outputs (Y0-Y1)



Standard Input Circuit (X4-X23)



D0-06DD2-D General Specifications	
External Power Requirements	12–24 VDC, 20W maximum,
Communication Port 1: 9600 baud (Fixed), 8 data bits, 1 stop bit, odd parity	K–Sequence (Slave), DirectNET (Slave), MODBUS (Slave)
Communication Port 2: 9600 baud (default), 8 data bits, 1 stop bit, odd parity	K–Sequence (Slave), DirectNET (Master/Slave), MODBUS (Master/Slave), Non-sequence/print, ASCII in/out
Programming cable type	D2–DSCBL
Operating Temperature	32 to 131°F (0 to 55°C)
Storage Temperature	–4 to 158°F (–20 to 70°C)
Relative Humidity	5 to 95% (non-condensing)
Environmental air	No corrosive gases permitted
Vibration	MIL STD 810C 514.2
Shock	MIL STD 810C 516.2
Noise Immunity	NEMA ICS3–304
Terminal Type	Removable
Wire Gauge	One 16 AWG or two 18 AWG, 24 AWG minimum

DC Input Specifications		
Parameter	High–Speed Inputs, X0 – X3	Standard DC Inputs X4 – X23
Min. - Max. Voltage Range	10.8–26.4 VDC	10.8–26.4 VDC
Operating Voltage Range	12–24 VDC	12–24 VDC
Peak Voltage	30VDC (7kHz maximum frequency)	30VDC
Minimum Pulse Width	70µs	N/A
ON Voltage Level	>10.0 VDC	> 10.0 VDC
OFF Voltage Level	< 2.0 VDC	< 2.0 VDC
Max. Input Current	15mA @26.4VDC	11mA @ 26.4 VDC
Input Impedance	1.8 kq @ 12–24 VDC	2.8 kq @ 12–24 VDC
Minimum ON Current	5mA	3mA
Maximum OFF Current	0.5 mA	0.5 mA
OFF to ON Response	<70µs	2–8 ms, 4ms typical
ON to OFF Response	<70µs	2–8 ms, 4ms typical
Status Indicators	Logic side	Logic side
Commons	4 channels / common x 5 banks (isolated)	

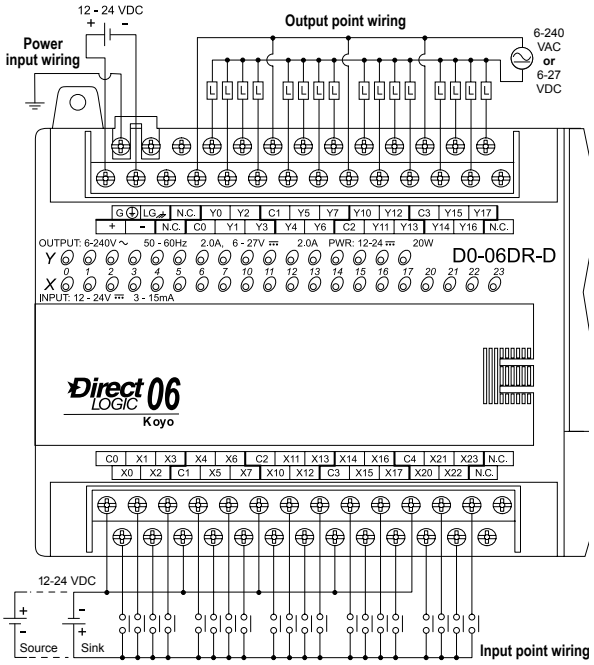
DC Output Specifications		
Parameter	Pulse Outputs, Y0 – Y1	Standard Outputs, Y2 – Y17
Min. - Max. Voltage Range	10.8–26.4 VDC	10.8–26.4 VDC
Operating Voltage	12–24 VDC	12–24 VDC
Peak Voltage	30VDC (10kHz max. frequency)	30VDC
On Voltage Drop	0.5 VDC @ 1A	1.2 VDC @ 1 A
Max Current (resistive)	0.5 A / pt., 1A / pt. as standard pt.	1.0 A / point
Max leakage current	15µA @ 30VDC	15µA @ 30VDC
Max inrush current	2A for 100ms	2A for 100ms
External DC power required	N/A	N/A
OFF to ON Response	< 10µs	< 10µs
ON to OFF Response	< 20µs	< 0.5 ms
Status Indicators	Logic Side	Logic Side
Commons	4 channels / common x 4 banks (non-isolated)	
Fuses	None (external recommended)	

D0–06DR–D I/O Wiring Diagram

The D0–06DR–D PLC has twenty DC inputs and sixteen relay contact outputs. The following diagram shows a typical field wiring example. The DC external power connection uses three terminals as shown.

Inputs are organized into five banks of four. Each bank has an isolated common terminal, and may be wired as either sinking or sourcing inputs. The wiring example above shows all commons connected together, but separate supplies and common circuits may be used.

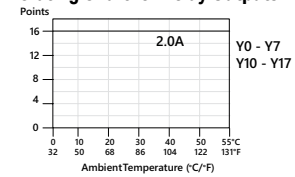
Outputs are organized into four banks of four normally-open relay contacts. Each bank has a common terminal. The wiring example above shows all commons connected together, but separate supplies and common circuits may be used. The equivalent output circuit shows one channel of a typical bank. The relay contacts can switch AC or DC voltages.



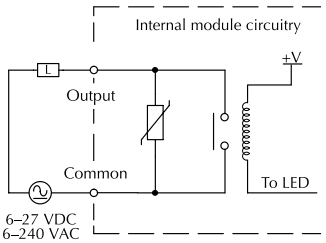
Typical Relay Life (Operations) at Room Temperature

Voltage & Load Type	Load Current	
	At 1A	At 2A
24VDC Resistive	500K	250K
24VDC Inductive	100K	50K
110VAC Resistive	500K	250K
110VAC Inductive	200K	100K
220VAC Resistive	350K	200K
220VAC Inductive	100K	50K

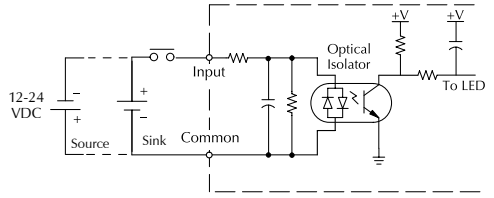
Derating Chart for Relay Outputs



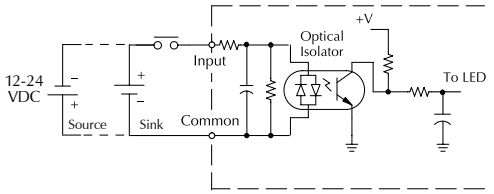
Standard Output Circuit



Standard Input Circuit (X4-X23)



High-speed Input Circuit (X0-X3)



D0-06DR-D General Specifications	
External Power Requirements	12–24 VDC, 20W maximum,
Communication Port 1 9600 baud (Fixed), 8 data bits, 1 stop bit, odd parity	K–Sequence (Slave), DirectNET (Slave), MODBUS (Slave)
Communication Port 2 9600 baud (default), 8 data bits, 1 stop bit, odd parity	K–Sequence (Slave), DirectNET (Master/Slave), MODBUS (Master/Slave),Non-sequence/print, ASCII in/out
Programming cable type	D2–DSCBL
Operating Temperature	32 to 131°F (0 to 55°C)
Storage Temperature	–4 to 158°F (–20 to 70°C)
Relative Humidity	5 to 95% (non-condensing)
Environmental air	No corrosive gases permitted
Vibration	MIL STD 810C 514.2
Shock	MIL STD 810C 516.2
Noise Immunity	NEMA ICS3–304
Terminal Type	Removable
Wire Gauge	One 16 AWG or two 18AWG, 24AWG minimum

DC Input Specifications		
Parameter	High–Speed Inputs, X0 – X3	Standard DC Inputs X4 – X23
Min. - Max. Voltage Range	10.8–26.4 VDC	10.8–26.4 VDC
Operating Voltage Range	12–24 VDC	12–24 VDC
Peak Voltage	30VDC (7kHz maximum frequency)	30VDC
Minimum Pulse Width	70µs	N/A
ON Voltage Level	> 10VDC	> 10VDC
OFF Voltage Level	< 2.0 VDC	< 2.0 VDC
Input Impedance	1.8 kq @ 12–24 VDC	2.8 kq @ 12–24 VDC
Max. Input Current	6mA @ 12VDC 13mA @ 24VDC	4mA @12VDC 8.5mA @ 24VDC
Minimum ON Current	>5mA	>4 mA
Maximum OFF Current	< 0.5 mA	<0.5 mA
OFF to ON Response	<70µs	2–8 ms, 4ms typical
ON to OFF Response	< 70µs	2–8 ms, 4ms typical
Status Indicators	Logic side	Logic side
Commons	4 channels / common x 5 banks (isolated)	

Relay Output Specifications	
Output Voltage Range (Min. - Max.)	5–264 VAC (47–63 Hz), 5–30 VDC
Operating Voltage	6–240 VAC (47–63 Hz), 6–27 VDC
Output Current	2A / point 6A / common
Maximum Voltage	264VAC, 30VDC
Max leakage current	0.1 mA @ 264VAC
Smallest Recommended Load	5mA
OFF to ON Response	< 15ms
ON to OFF Response	< 10ms
Status Indicators	Logic Side
Commons	4 channels / common x 4 banks isolated commons
Fuses	None (external recommended)

Glossary of Specification Terms

Discrete Input
One of twenty input connections to the PLC which converts an electrical signal from a field device to a binary status (off or on), which is read by the internal CPU each PLC scan.

Discrete Output
One of sixteen output connections from the PLC which converts an internal ladder program result (0 or 1) to turn On or Off an output switching device. This enables the program to turn on and off large field loads.

I/O Common
A connection in the input or output terminals which is shared by multiple I/O circuits. It usually is in the return path to the power supply of the I/O circuit.

Input Voltage Range
The operating voltage range of the input circuit.

Maximum Voltage
Maximum voltage allowed for the input circuit.

ON Voltage Level
The minimum voltage level at which the input point will turn ON.

OFF Voltage Level
The maximum voltage level at which the input point will turn OFF

Input Impedance
Input impedance can be used to calculate input current for a particular operating voltage.

Input Current
Typical operating current for an active (ON) input.

Minimum ON Current
The minimum current for the input circuit to operate reliably in the ON state.

Maximum OFF Current
The maximum current for the input circuit to operate reliably in the OFF state.

OFF to ON Response
The time the module requires to process an OFF to ON state transition.

ON to OFF Response
The time the module requires to process an ON to OFF state transition.

Status Indicators
The LEDs that indicate the ON/OFF status of an input or output point. All LEDs on DL06 Micro PLCs are electrically located on the logic side of the input or output circuit.

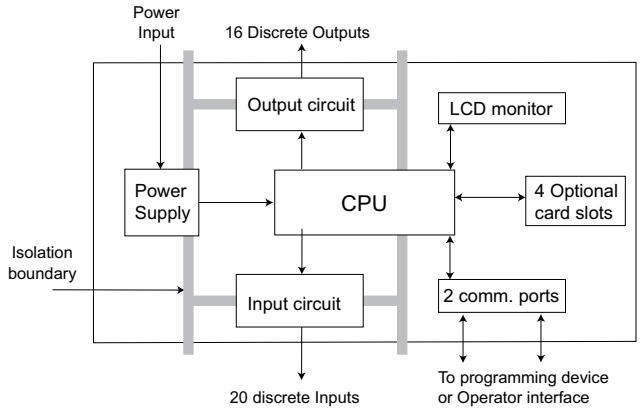
CPU Specifications and Operation

In This Chapter...

Overview

The Central Processing Unit (CPU) is the heart of the Micro PLC. Almost all PLC operations are controlled by the CPU, so it is important that it is set up correctly. This chapter provides the information needed to understand:

- Steps required to set up the CPU
- Operation of ladder programs
- Organization of Variable Memory



NOTE: *The High-Speed I/O function (HSIO) consists of dedicated but configurable hardware in the DL06. It is not considered part of the CPU because it does not execute the ladder program. For more on HSIO operation, see Appendix E.*

DL06 CPU Features

The DL06 Micro PLC has 14.8K words of memory comprised of 7.6K of ladder memory and 7.6K words of V-memory (data registers). Program storage is in the FLASH memory which is a part of the CPU board in the PLC. In addition, there is RAM with the CPU which will store system parameters, V-memory, and other data not in the application program. The RAM is backed up by a super-capacitor, storing the data for several hours in the event of a power outage. The capacitor automatically charges during powered operation of the PLC.

The DL06 supports fixed I/O which includes twenty discrete input points and sixteen output points.

Over 220 different instructions are available for program development as well as extensive internal diagnostics that can be monitored from the application program or from an operator interface. Chapters 5, 6, and 7 provide detailed descriptions of the instructions.

The DL06 provides two built-in communication ports, so you can easily connect a handheld programmer, operator interface, or a personal computer without needing any additional hardware.

CPU Specifications

Specifications	
Feature	**DL06**
Total Program memory (words)	14.8K
Ladder memory (words)	7680
Total V-memory (words)	7616
User V-memory (words)	7488
Non-volatile V Memory (words)	128
Contact execution (boolean)	<0.6us
Typical scan (1k boolean)	1-2ms
RLL Ladder style Programming	Yes
RLL and RLLPLUS Programming	Yes
Run Time Edits	Yes
Supports Overrides	Yes
Scan	Variable / fixed
Handheld programmer	Yes
*Direct*SOFT programming for Windows	Yes
Built-in communication ports (RS232C)	Yes
FLASH Memory	Standard on CPU
Local Discrete I/O points available	36
Local Analog input / output channels maximum	None
High-Speed I/O (quad, pulse out, interrupt, pulse catch, etc.)	Yes, 2
I/O Point Density	20 inputs, 16 outputs
Number of instructions available (see Chapter 5 for details)	229
Control relays	1024
Special relays (system defined)	512
Stages in RLL^{PLUS}	1024
Timers	256
Counters	128
Immediate I/O	Yes
Interrupt input (external / timed)	Yes
Subroutines	Yes
For/Next Loops	Yes
Math (Integer and floating point)	Yes
Drum Sequencer Instruction	Yes
Time of Day Clock/Calendar	Yes
Internal diagnostics	Yes
Password security	Yes
System error log	Yes
User error log	Yes
Battery backup	Optional D2-BAT-1 available (not included with unit)

CPU Hardware Setup

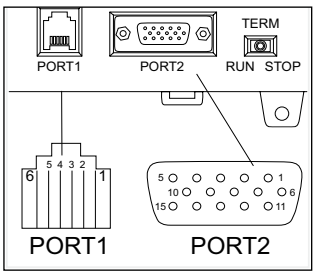
Communication Port Pinout Diagrams

Cables are available that allow you to quickly and easily connect a Handheld Programmer or a personal computer to the DL06 PLCs. However, if you need to build your cable(s), use the pinout descriptions shown below, or use the Tech Support/Cable Wiring Diagrams located on our website. The DL06 PLCs require an RJ-12 phone plug for port 1 (D2-DSCBL) and a 15-pin SVGA DSub for port 2 (D2-DSCBL-1).

The DL06 PLC has two built-in serial communication ports. Port 1 (RS232C only) is generally used for connecting to a D2-HPP, *Direct*SOFT, operator interface, MODBUS slave only, or a *Direct*NET slave only. The baud rate is fixed at 9600 baud for port 1. Port 2 (RS232C/RS422/RS485) can be used to connect to a D2-HPP, *Direct*SOFT, operator interface, MODBUS master/slave, *Direct*NET master/slave or ASCII in/out. Port 2 has a range of speeds from 300 baud to 38.4K baud.

NOTE: *The 5v pins are rated at 220mA maximum, primarily for use with some operator interface units.*

Port 1 Pin Descriptions		
1	0V	Power (-) connection (GND)
2	5V	Power (-) 220 mA max
3	RXD	Receive data (RS-232C)
4	TXD	Transmit data (RS-232C)
5	5V	Power (+) connection
6	0V	Power (-) connection (GND)



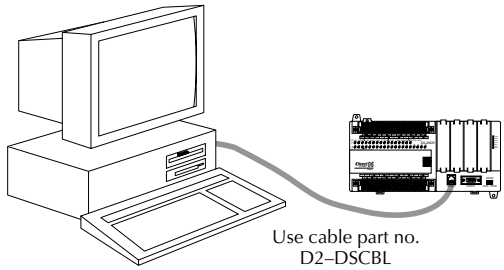
Port 2 Pin Descriptions		
1	5V	Power (+) connection
2	TXD	Transmit data (RS-232C)
3	RXD	Receive data (RS-232C)
4	RTS	Ready to send
5	CTS	Clear to send
6	RXD-	Receive data (-) (RS-422/485)
7	0V	Power (-) connection (GND)
8	0V	Power (-) connection (GND)
9	TXD+	Transmit data (+) (RS-422/485)
10	TXD-	Transmit data (-) (RS-422/485)
11	RTS+	Ready to send (+) (RS-422/485)
12	RTS-	Ready to send (-) (RS-422/485)
13	RXD+	Receive data (+) (RS-422/485)
14	CTS+	Clear to send (+) (RS-422/485)
15	CTS-	Clear to send (-) (RS-422/485)

Communications Port 1	
Com 1	Connects to HPP, *Direct*SOFT, operator interfaces, etc.
	6-pin, RS232C
	Communication speed (baud): 9600 (fixed)
	Parity: odd (fixed)
	Station Address: 1 (fixed)
	8 data bits
	1 start, 1 stop bit
	Asynchronous, half-duplex, DTE
	Protocol (auto-select): K-sequence (slave only), *Direct*NET (slave only), MODBUS (slave only)

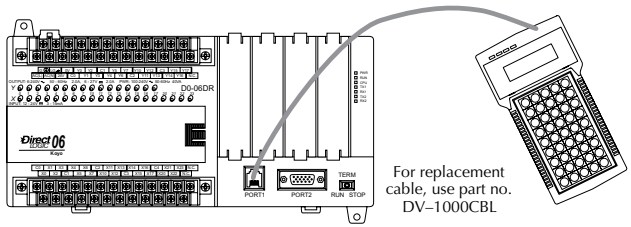
Communications Port 2	
Com 2	Connects to HPP, *Direct*SOFT, operator interfaces, etc.
	15-pin, multifunction port, RS232C, RS422, RS485 (RS485 with 2-wire is only available for MODBUS and Non-sequence.)
	Communication speed (baud): 300, 600, 1200, 2400, 4800, 9600, 19200, 38400
	Parity: odd (default), even, none
	Station Address: 1 (default)
	8 data bits
	1 start, 1 stop bit
	Asynchronous, half-duplex, DTE
	Protocol (auto-select): K-sequence (slave only), *Direct*NET (master/slave), MODBUS (master/slave), non-sequence/print/ASCII in/out

Connecting the Programming Devices

If you're using a Personal Computer with the *Direct*SOFT programming package, you can connect the computer to either of the DL06's serial ports. For an engineering office environment (typical during program development), this is the preferred method of programming.



The handheld programmer D2-HPP is connected to the CPU with a handheld programmer cable. This device is ideal for maintaining existing installations or making small program changes. The handheld programmer is shipped with a cable, which is approximately 6.5 feet (200 cm) long.

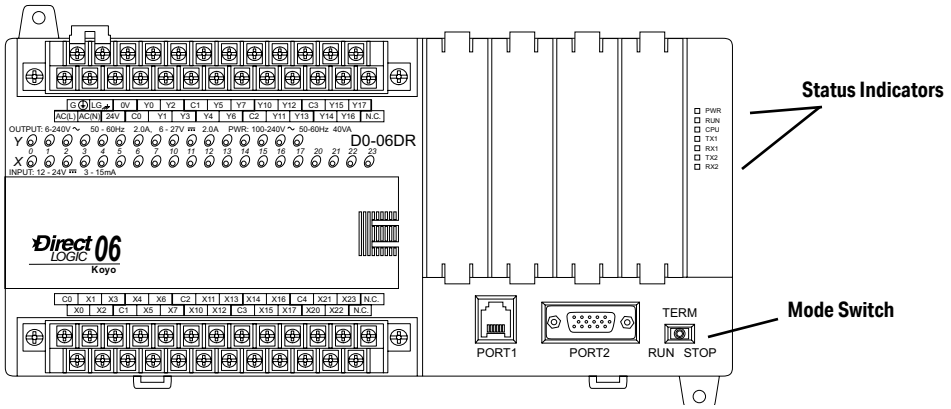


CPU Setup Information

Even if you have years of experience using PLCs, there are a few things you need to do before you can start entering programs. This section includes some basic things, such as changing the CPU mode, but it also includes some things that you may never have to use. Here's a brief list of the items that are discussed:

- Using Auxiliary Functions
- Clearing the program (and other memory areas)
- How to initialize system memory
- Setting retentive memory ranges

The following paragraphs provide the setup information necessary to get the CPU ready for programming. They include setup instructions for either type of programming device you are using. The D2–HPP Handheld Programmer Manual provides the Handheld keystrokes required to perform all of these operations. The *Direct*SOFT Manual provides a description of the menus and keystrokes required to perform the setup procedures via *Direct*SOFT.



Status Indicators

The status indicator LEDs on the CPU front panels have specific functions which can help in programming and troubleshooting.

Mode Switch Functions

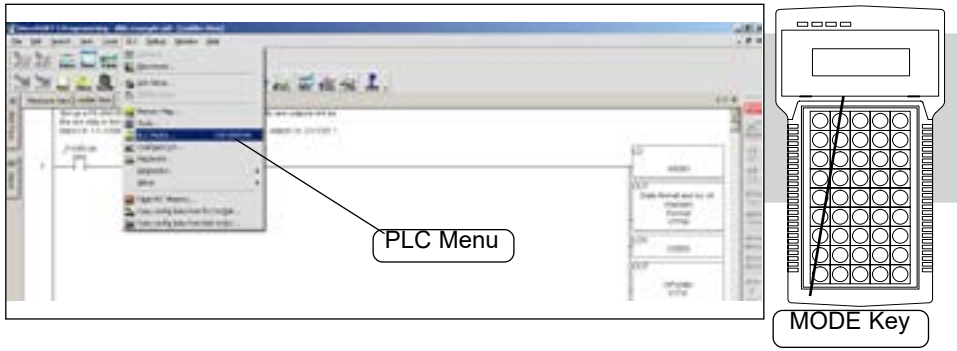
The mode switch on the DL06 PLC provides positions for enabling and disabling program changes in the CPU. Unless the mode switch is in the TERM position, RUN and STOP mode changes will not be allowed by any interface device, (handheld programmer, ***Direct***SOFT programming package or operator interface). Programs may be viewed or monitored but no changes may be made. If the switch is in the TERM position and no program password is in effect, all operating modes as well as program access will be allowed through the connected programming or monitoring device.

Indicator	Status	Meaning
PWR	ON	Power good
	OFF	Power failure
RUN	ON	CPU is in Run Mode
	OFF	CPU is in Stop or Program Mode
	Blinking	CPU is in firmware upgrade mode
CPU	ON	CPU self diagnostics error
	OFF	CPU self diagnostics good
	Blinking	The CPU indicator will blink if the battery is less than 2.5 VDC
TX1	ON	Data is being transmitted by the CPU - Port 1
	OFF	No data is being transmitted by the CPU - Port 1
RX1	ON	Data is being received by the CPU - Port 1
	OFF	No data is being received by the CPU - Port 1
TX2	ON	Data is being transmitted by the CPU - Port 2
	OFF	No data is being transmitted by the CPU - Port 2
RX2	ON	Data is being received by the CPU - Port 2
	OFF	No data is being received by the CPU - Port 2

Changing Modes in the DL06 PLC

Mode Switch Position	CPU Action
RUN (Run Program)	CPU is forced into the RUN mode if no errors are encountered. No changes are allowed by the attached programming/monitoring device.
TERM (Terminal) RUN	PROGRAM and the TEST modes are available. Mode and program changes are allowed by the programming/monitoring device.
STOP	CPU is forced into the STOP mode. No changes are allowed by the programming/monitoring device.

There are two ways to change the CPU mode. You can use the CPU mode switch to select the operating mode, or you can place the mode switch in the TERM position and use a programming device to change operating modes. With the switch in this position, the CPU can be changed between Run and Program modes. You can use either *Direct*SOFT or the Handheld Programmer to change the CPU mode of operation. With *Direct*SOFT use the PLC menu option **PLC > Mode** or use the **Mode** button located on the Online toolbar. With the Handheld Programmer, you use the MODE key.



Mode of Operation at Power-up

The DL06 CPU will normally power-up in the mode that it was in just prior to the power interruption. For example, if the CPU was in Program Mode when the power was disconnected, the CPU will power-up in Program Mode (see warning note below).

WARNING: Once the super capacitor has discharged, the system memory may not retain the previous mode of operation. When this occurs, the PLC can power-up in either Run or Program Mode if the mode switch is in the term position. There is no way to determine which mode will be entered as the startup mode. Failure to adhere to this warning greatly increases the risk of unexpected equipment startup.

The mode which the CPU will power-up in is also determined by the state of B7633.13. If the bit is set and the Mode Switch is in the TERM position, the CPU will power-up in RUN mode. If B7633.13 is not set with the Mode Switch in TERM position, then the CPU will power-up in the state it was in when it was powered-down.

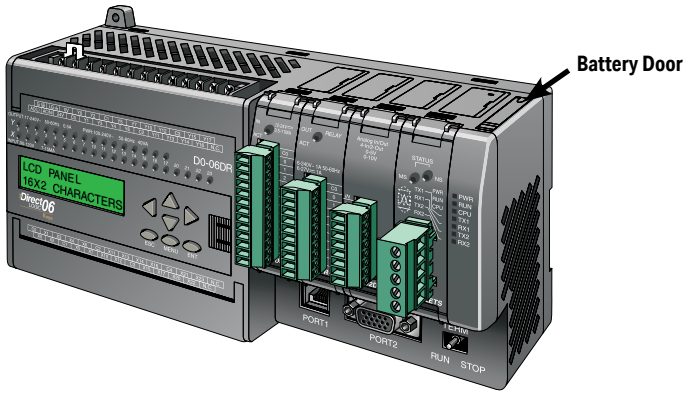
Using Battery Backup

An optional lithium battery is available to maintain the system RAM retentive memory when the DL06 system is without external power. Typical CPU battery life is five years, which includes PLC runtime and normal shutdown periods. However, consider installing a fresh battery if your battery has not been changed recently and the system will be shut down for a period of more than ten days.

***NOTE**: Before installing or replacing your CPU battery, back-up your V-memory and system parameters. You can do this by using **Direct**SOFT to save the program, V-memory, and system parameters to hard/floppy disk on a personal computer.*

To install the D2–BAT–1 CPU battery in the DL06 CPU:

1. Press the retaining clip on the battery door down and swing the battery door open.
2. Place the battery into the coin–type slot with the +, or larger, side out.
3. Close the battery door making sure that it locks securely in place.
4. Make a note of the date the battery was installed



WARNING: Do not attempt to recharge the battery or dispose of an old battery by fire. The battery may explode or release hazardous materials.

Battery Backup

The battery backup is available immediately after the battery has been installed. ***The CPU indicator will blink if the battery is low*** (refer to the table on page 3-6). Special Relay 43 (SP43) will also be set when the battery is low. The low battery indication is enabled by setting bit 12 of V7633 (B7633.12). If the low battery feature is not desired, do not set bit V7633.12. The super capacitor will retain memory IF it is configured as retentive regardless of the state of B7633.12. The battery will do the same, but for a much longer time.

Auxiliary Functions

Many CPU setup tasks involve the use of Auxiliary (AUX) Functions. The AUX Functions perform many different operations, ranging from clearing ladder memory, displaying the scan time, copying programs to EEPROM in the handheld programmer, etc. They are divided into categories that affect different system parameters. Appendix A provides a description of the AUX functions.

You can access the AUX Functions from *Direct*SOFT or from the D2–HPP Handheld Programmer. The manuals for those products provide step-by-step procedures for accessing the AUX Functions. Some of these AUX Functions are designed specifically for the Handheld Programmer setup, so they will not be needed (or available) with the *Direct*SOFT package. The following table shows a list of the Auxiliary functions for the Handheld Programmer.

Auxiliary Functions	
AUX 2* — RLL Operations	
21	Check Program
22	Change Reference
23	Clear Ladder Range
24	Clear All Ladders
AUX 3* — V-Memory Operations	
31	Clear V-memory
AUX 4* — I/O Configuration	
41	Show I/O Configuration
42	I/O Diagnostics
44	Power Up I/O Configuration check
45	Select Configuration
46	Configure I/O
AUX 5* — CPU Configuration	
51	Modify Program Name
52	Display/Change Calendar
53	Display Scan Time
54	Initialize Scratchpad
55	Set Watchdog Timer
56	Set Communication Port 2

Auxiliary Functions (cont'd)	
57	Set Retentive Ranges
58	Test Operations
59	Override Setup
5B	HSIO Configuration
5C	Display Error History
5D	Scan Control Setup
AUX 6* — Handheld Programmer Configuration	
61	Show Revision Numbers
62	Beeper On / Off
65	Run Self Diagnostics
AUX 7* — EEPROM Operations	
71	Copy CPU memory to HPP EEPROM
72	Write HPP EEPROM to CPU
73	Compare CPU to HPP EEPROM
74	Blank Check (HPP EEPROM)
75	Erase HPP EEPROM
76	Show EEPROM Type (CPU and HPP)
AUX 8* — Password Operations	
81	Modify Password
82	Unlock CPU
83	Lock CPU

Clearing an Existing Program

Before you enter a new program, be sure to always clear ladder memory. You can use AUX Function 24 to clear the complete program. Also you can use other AUX functions to clear other memory areas.

- AUX 23 — Clear Ladder Range
- AUX 24 — Clear all Ladders
- AUX 31 — Clear V-memory

Initializing System Memory

The DL06 Micro PLC maintains system parameters in a memory area often referred to as the **scratchpad**. In some cases, you may make changes to the system setup that will be stored in system memory. For example, if you specify a range of Control Relays (CRs) as retentive, these changes are stored in system memory. AUX 54 resets the system memory to the default values.

WARNING: You may never have to use this feature unless you want to clear any setup information that is stored in system memory. Usually, you'll only need to initialize the system memory if you are changing programs and the old program required a special system setup. You can usually load in new programs without ever initializing system memory.

Remember, this AUX function will reset all system memory. If you have set special parameters such as retentive ranges, for example, they will be erased when AUX 54 is used. Make sure that you have considered all ramifications of this operation before you select it. See Appendix F for additional information in reference to PLC memory.

Setting Retentive Memory Ranges

The DL06 PLCs provide certain ranges of retentive memory by default. The default ranges are suitable for many applications, but you can change them if your application requires additional retentive ranges or no retentive ranges at all. The default settings are:

Memory Area	DL06	
	Default Range	Available Range
Control Relays	C1000 – C1777	C0 – C1777
V-Memory	V400 – V37777	V0 – V37777
Timers	None by default	T0 – T377
Counters	CT0 – CT177	CT0 – CT177
Stages	None by default	S0 – S1777

You can use AUX 57 to set the retentive ranges. You can also use *Direct*SOFT menus to select the retentive ranges. Appendix A contains detailed information about auxiliary functions.

WARNING: The DL06 CPUs do not come with a battery. The super capacitor will retain the values in the event of a power loss, but only for a short period of time, depending on conditions (typically 4 to 7 days). If the retentive ranges are important for your application, make sure you obtain the optional battery.

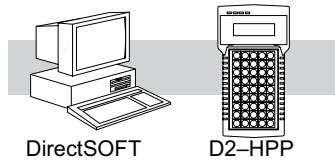
Using a Password

The DL06 PLCs allow you to use a password to help minimize the risk of unauthorized program and/or data changes. Once you enter a password you can lock the PLC against access. Once the CPU is locked you must enter the password before you can use a programming device to change any system parameters.

You can select an 8-digit numeric password. The Micro PLCs are shipped from the factory with a password of 00000000. All zeros removes the password protection. If a password has been entered into the CPU you cannot just enter all zeros to remove it. Once you enter the correct password, you can change the password to all zeros to remove the password protection.

WARNING: Make sure you remember your password. If you forget your password you will not be able to access the CPU. The Micro PLC must be returned to the factory to have the password (along with the ladder project) removed. It is the policy of Automationdirect to require the memory of the PLC to be cleared along with the password.

You can use the D2–HPP Handheld Programmer or ***DirectSOFT*** to enter a password. The following diagram shows how you can enter a password with the Handheld Programmer.



Select AUX 81

CLR CLR I 8 B 1 AUX ENT

```
PASSWORD
 00000000
```

Enter the new 8-digit password

X X • • • X ENT

```
PASSWORD
 XXXXXXXX
```

Press CLR to clear the display

There are three ways to lock the CPU once the password has been entered.

1. If the CPU power is disconnected, the CPU will be automatically locked against access.
2. If you enter the password with DirectSOFT, the CPU will be automatically locked against access when you exit DirectSOFT.
3. Use AUX 83 to lock the CPU.

When you use ***Direct***SOFT, you will be prompted for a password if the CPU has been locked. If you use the Handheld Programmer, you have to use AUX 82 to unlock the CPU. Once you enter AUX 82, you will be prompted to enter the password.

NOTE*: The DL06 CPUs support multi-level password protection of the ladder program. This allows password protection while not locking the communication port to an operator interface. The multi-level password can be invoked by creating a password with an upper case* ***A*** *followed by seven numeric characters (e.g. A1234567).*

CPU Operation

Achieving the proper control for your equipment or process requires a good understanding of how DL06 CPUs control all aspects of system operation. There are four main areas to understand before you create your application program:

- CPU Operating System — the CPU manages all aspects of system control. A quick overview of all the steps is provided in the next section.
- CPU Operating Modes — The two primary modes of operation are Program Mode and Run Mode.
- CPU Timing — The two important areas we discuss are the I/O response time and the CPU scan time.
- CPU Memory Map — DL06 CPUs offer a wide variety of resources, such as timers, counters, inputs, etc. The memory map section shows the organization and availability of these data types.

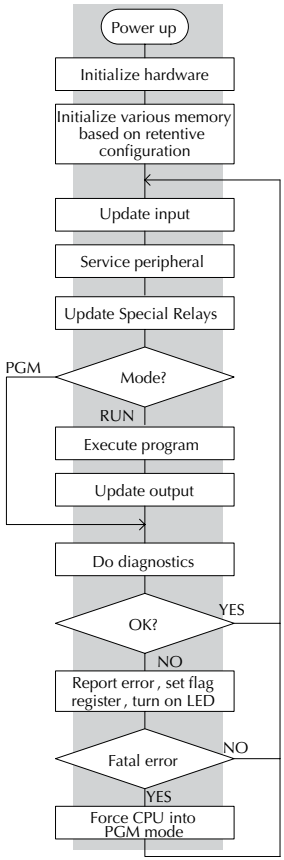
CPU Operating System

At powerup, the CPU initializes the internal electronic hardware. Memory initialization starts with examining the retentive memory settings. In general, the contents of retentive memory is preserved, and non-retentive memory is initialized to zero (unless otherwise specified).

After the one-time powerup tasks, the CPU begins the cyclical scan activity. The flowchart to the right shows how the tasks differ, based on the CPU mode and the existence of any errors. The scan time is defined as the average time around the task loop. Note that the CPU is always reading the inputs, even during program mode. This allows programming tools to monitor input status at any time.

The outputs are only updated in Run mode. In program mode, they are in the off state.

Error detection has two levels. Non-fatal errors are reported, but the CPU remains in its current mode. If a fatal error occurs, the CPU is forced into program mode and the outputs go off.



Program Mode

In Program Mode, the CPU does not execute the application program or update the output points. The primary use for Program Mode is to enter or change an application program. You also use program mode to set up the CPU parameters, such as HSIO features, retentive memory areas, etc.

You can use a programming device, such as ***Direct***SOFT, the D2–HPP (Handheld Programmer) or the CPU mode switch to place the CPU in Program Mode.

Run Mode

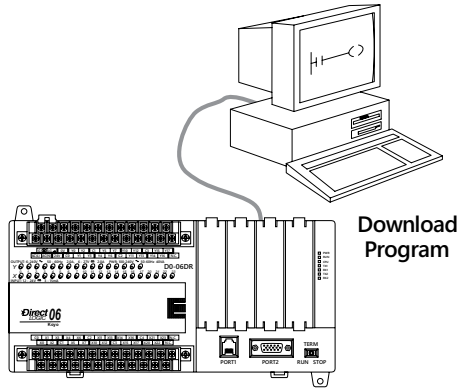
In Run Mode, the CPU executes the application program and updates the I/O system. You can perform many operations during Run Mode. Some of these include:

- Monitor and change I/O point status
- Change timer/counter preset values
- Change variable memory locations

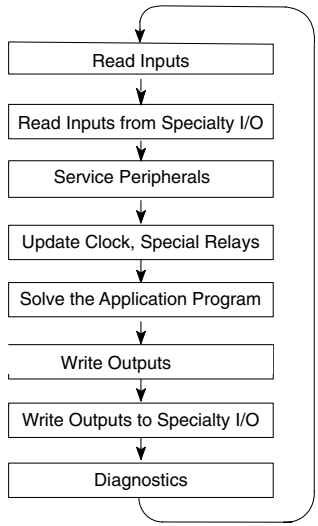
Run Mode operation can be divided into several key areas. For the vast majority of applications, some of these execution segments are more important than others. For example, you need to understand how the CPU updates the I/O points, handles forcing operations, and solves the application program. The remaining segments are not that important for most applications.

You can use ***Direct***SOFT, the D2–HPP (Handheld Programmer) or the CPU mode switch to place the CPU in Run Mode.

You can also edit the program during Run Mode. The Run Mode Edits are not bumpless to the outputs. Instead, the CPU ignores the inputs and maintains the outputs in their last state while it accepts the new program information. If an error is found in the new program, then the CPU will turn all the outputs off and enter the Program Mode. This feature is discussed in more detail in Chapter 9.



Normal Run mode scan



WARNING: Only authorized personnel fully familiar with all aspects of the application should make changes to the program. Changes during Run Mode become effective immediately. Make sure you thoroughly consider the impact of any changes to minimize the risk of personal injury or damage to equipment.

Read Inputs

The CPU reads the status of all inputs, then stores it in the image register. Input image register locations are designated with an X followed by a memory location. Image register data is used by the CPU when it solves the application program.

Of course, an input may change after the CPU has just read the inputs. Generally, the CPU scan time is measured in milliseconds. If you have an application that cannot wait until the next I/O update, you can use Immediate Instructions. These do not use the status of the input image register to solve the application program. The Immediate instructions immediately read the input status directly from the I/O modules. However, this lengthens the program scan since the CPU has to read the I/O point status again. A complete list of the Immediate instructions is included in Chapter 5.

Service Peripherals and Force I/O

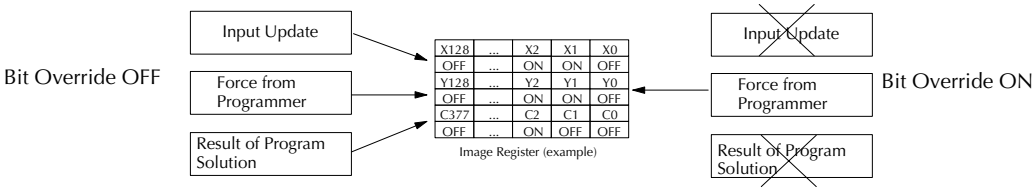
After the CPU reads the inputs from the input modules, it reads any attached peripheral devices. This is primarily a communications service for any attached devices. For example, it would read a programming device to see if any input, output, or other memory type status needs to be modified. There are two basic types of forcing available with the DL06 CPUs:

- Forcing from a peripheral – not a permanent force, good only for one scan
- Bit Override – holds the I/O point (or other bit) in the current state. Valid bits are X, Y, C, T, CT, and S. (These memory types are discussed in more detail later in this chapter).

Regular Forcing — This type of forcing can temporarily change the status of a discrete bit. For example, you may want to force an input on, even though it is really off. This allows you to change the point status that was stored in the image register. This value will be valid until the image register location is written to during the next scan. This is primarily useful during testing situations when you need to force a bit on to trigger another event.

Bit Override — Bit override can be enabled on a point-by-point basis by using AUX 59 from the Handheld Programmer or, by a menu option from within ***Direct***SOFT. Bit override basically disables any changes to the discrete point by the CPU. For example, if you enable bit override for X1, and X1 is off at the time, then the CPU will not change the state of X1. This means that even if X1 comes on, the CPU will not acknowledge the change. So, if you used X1 in the program, it would always be evaluated as **Off** in this case. Of course, if X1 was on when the bit override was enabled, then X1 would always be evaluated as **On**.

There is an advantage available when you use the bit override feature. The regular forcing is not disabled because the bit override is enabled. For example, if you enabled the Bit Override for Y0 and it was off at the time, then the CPU would not change the state of Y0. However, you can still use a programming device to change the status. Now, if you use the programming device to force Y0 on, it will remain on and the CPU will not change the state of Y0. If you then force Y0 off, the CPU will maintain Y0 as off. The CPU will never update the point with the results from the application program or from the I/O update until the bit override is removed. The following diagram shows a brief overview of the bit override feature. Notice the CPU does not update the Image Register when bit override is enabled.



WARNING: Only authorized personnel fully familiar with all aspects of the application should make changes to the program. Make sure you thoroughly consider the impact of any changes to minimize the risk of personal injury or damage to equipment.

CPU Bus Communication

It is possible to transfer data to and from the CPU over the CPU bus on the backplane. This data is more than standard I/O point status. This type of communications can only occur on the CPU (local) base. There is a portion of the execution cycle used to communicate with these modules. The CPU performs both read and write requests during this segment.

Update Clock, Special Relays and Special Registers

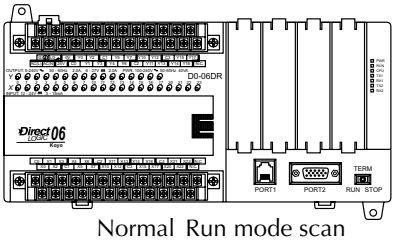
The DL06 CPUs have an internal real-time clock and calendar timer which is accessible to the application program. Special V-memory locations hold this information. This portion of the execution cycle makes sure these locations get updated on every scan. Also, there are several different Special Relays, such as diagnostic relays, for example, that are also updated during this segment.

Solve Application Program

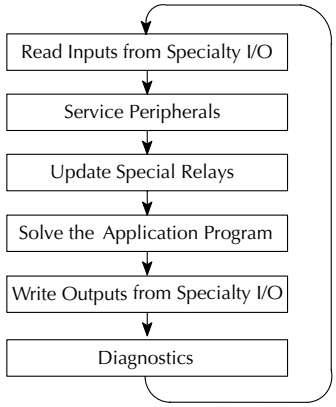
The CPU evaluates each instruction in the application program during this segment of the scan cycle. The instructions define the relationship between the input conditions and the desired output response. The CPU uses the output image register area to store the status of the desired action for the outputs. Output image register locations are designated with a Y followed by a memory location. The actual outputs are updated during the write outputs segment of the scan cycle. There are immediate output instructions available that will update the output points immediately instead of waiting until the write output segment. A complete list of the Immediate instructions is provided in Chapter 5.

The internal control relays (C), the stages (S), and the variable memory (V) are also updated in this segment.

You may recall that you can force various types of points in the system, discussed earlier in this chapter. If any I/O points or memory data have been forced, the output image register also contains this information.



Normal Run mode scan



Solve PID Loop Equations

The DL06 CPU can process up to 8 PID loops. The loop calculations are run as a separate task from the ladder program execution, immediately following it. Only loops which have been configured are calculated, and then only according to a built-in loop scheduler. The sample time (calculation interval) of each loop is programmable. Please refer to Chapter 8, PID Loop Operation, for more on the effects of PID loop calculation on the overall CPU scan time.

Write Outputs

Once the application program has solved the instruction logic and constructed the output image register, the CPU writes the contents of the output image register to the corresponding output points. Remember, the CPU also made sure that any forcing operation changes were stored in the output image register, so the forced points get updated with the status specified earlier.

Write Outputs to Specialty I/O

After the CPU updates the outputs in the local and expansion bases, it sends the output point information that is required by any Specialty modules which are installed. Specialty modules have built-in microprocessors which communicate to the CPU via the backplane. Some of these modules can process data. Refer to the specific Specialty module user manual for detailed information.

Diagnostics

During this part of the scan, the CPU performs all system diagnostics and other tasks such as calculating the **scan time** and resetting the **watchdog timer**. There are many different error conditions that are automatically detected and reported by the DL06 PLCs. Appendix B contains a listing of the various error codes.

Probably one of the more important things that occurs during this segment is the **scan time calculation and watchdog timer control**. The DL06 CPU has a **watchdog timer** that stores the maximum time allowed for the CPU to complete the solve application segment of the scan cycle. If this time is exceeded, the CPU will enter the Program Mode and turn off all outputs. The default value set from the factory is 200 ms. An error is automatically reported. For example, the Handheld Programmer would display the following message "**E003 S/W TIMEOUT**" when the scan overrun occurs.

You can use AUX 53 to view the minimum, maximum, and current scan time. Use AUX 55 to increase or decrease the watchdog timer value.

I/O Response Time

Is Timing Important for Your Application?

I/O response time is the amount of time required for the control system to sense a change in an input point and update a corresponding output point. In the majority of applications, the CPU performs this task in such a short period of time that you may never have to concern yourself with the aspects of system timing. However, some applications do require extremely fast update times. In these cases, you may need to know how to determine the amount of time spent during the various segments of operation.

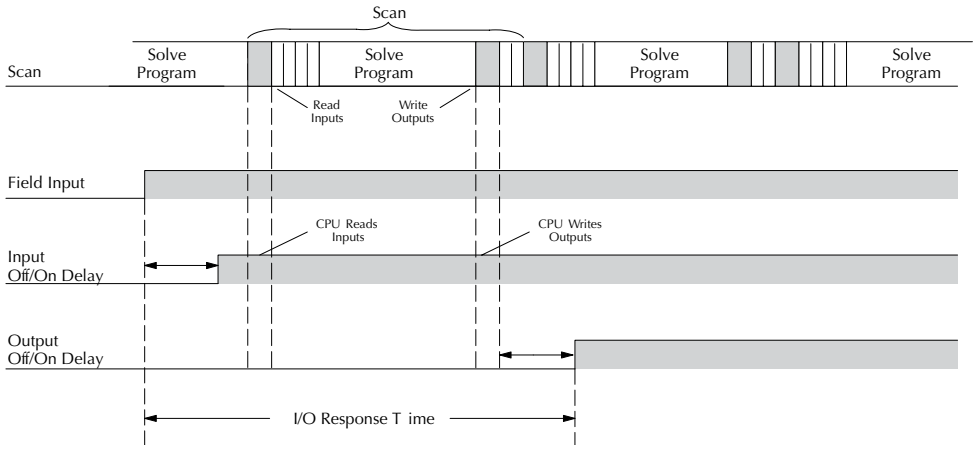
There are four things that can affect the I/O response time.

- The point in the scan cycle when the field input changes states
- Input Off to On delay time
- CPU scan time
- Output Off to On delay time

The next paragraphs show how these items interact to affect the response time.

Normal Minimum I/O Response

The I/O response time is shortest when the input changes just before the Read Inputs portion of the execution cycle. In this case the input status is read, the application program is solved, and the output point gets updated. The following diagram shows an example of the timing for this situation.

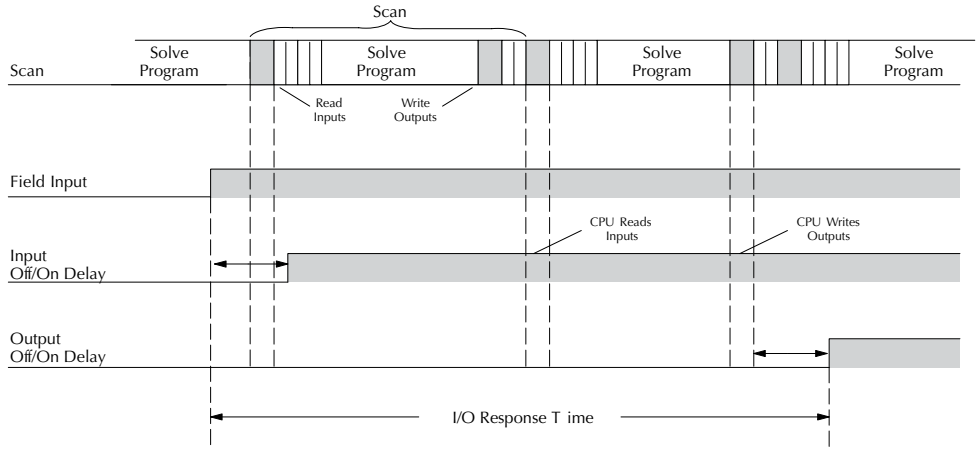


In this case, you can calculate the response time by simply adding the following items:

Input Delay + Scan Time + Output Delay = Response Time

Normal Maximum I/O Response

The I/O response time is longest when the input changes just after the Read Inputs portion of the execution cycle. In this case the new input status is not read until the following scan. The following diagram shows an example of the timing for this situation.



In this case, you can calculate the response time by simply adding the following items:

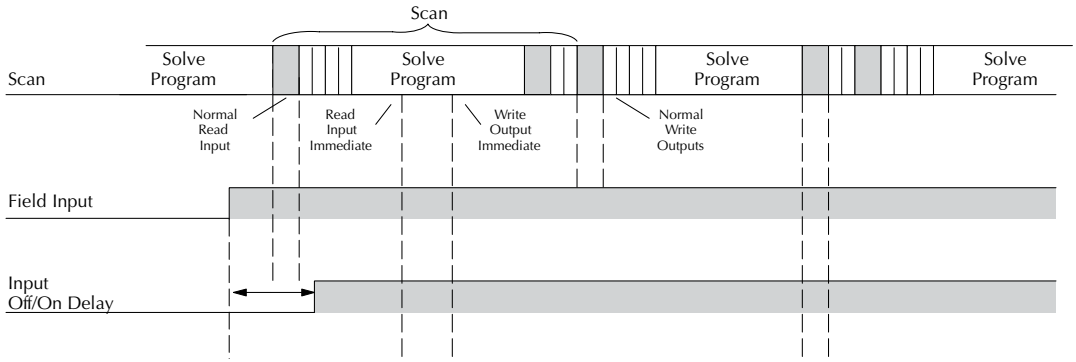
Input Delay +(2 x Scan Time) + Output Delay = Response Time

Improving Response Time

There are a few things you can do to help improve throughput.

- You can choose instructions with faster execution times
- You can use immediate I/O instructions (which update the I/O points during the program execution)
- You can use the HSIO Mode 50 Pulse Catch features designed to operate in high-speed environments. See Appendix E for details on using this feature.
- You can change Mode 60 filter to 0ms for X0, X1, X2, and X3.

Of these three things the Immediate I/O instructions are probably the most important and most useful. The following example shows how an immediate input instruction and immediate output instruction would affect the response time.



In this case, you can calculate the response time by simply adding the following items.

Input Delay + Instruction Execution Time + Output Delay = Response Time

The instruction execution time would be calculated by adding the time for the immediate input instruction, the immediate output instruction, and any other instructions in between the two.

NOTE: *Even though the immediate instruction reads the most current status from I/O, it only uses the results to solve that one instruction. It does not use the new status to update the image register. Therefore, any regular instructions that follow will still use the image register values. Any immediate instructions that follow will access the I/O again to update the status.*

CPU Scan Time Considerations

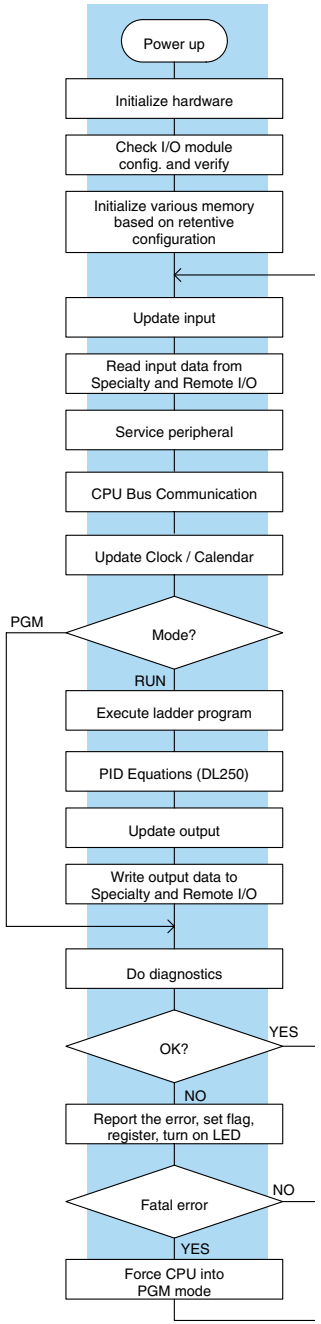
The scan time covers all the cyclical tasks that are performed by the operating system. You can use *Direct*SOFT or the Handheld Programmer to display the minimum, maximum, and current scan times that have occurred since the previous Program Mode to Run Mode transition. This information can be very important when evaluating the performance of a system. As we've shown previously there are several segments that make up the scan cycle. Each of these segments requires a certain amount of time to complete. Of all the segments, the following are the most important:

- Input Update
- Peripheral Service
- Program Execution
- Output Update
- Timed Interrupt Execution

The one you have the most control over is the amount of time it takes to execute the application program. This is because different instructions take different amounts of time to execute. So, if you think you need a faster scan, then you can try to choose faster instructions.

Your choice of I/O type and peripheral devices can also affect the scan time. However, these things are usually dictated by the application.

The following paragraphs provide some general information on how much time some of the segments can require.



Reading Inputs

The time required during each scan to read the input status of built-in inputs is 52.6 µs. Don't confuse this with the I/O response time that was discussed earlier.

Writing Outputs

The time required to write the output status of built-in outputs is 41.1 µs. Don't confuse this with the I/O response time that was discussed earlier.

Service Peripherals

Communication requests can occur at any time during the scan, but the CPU only logs the requests for service until the Service Peripherals portion of the scan. The CPU does not spend any time on this if there are no peripherals connected.

To Log Request (anytime)		DL06
Nothing Connected	Min. & Max	0µs
Port 1	Send Min. / Max.	5.8/11.8 µs
	Rec. Min. / Max.	12.5/25.2 µs
Port 2	Send Min. / Max.	6.2/14.3 µs
	Rec. Min. / Max.	14.2/31.9 µs
LCD	Min. / Max.	4.8/49.2 µs

During the Service Peripherals portion of the scan, the CPU analyzes the communications request and responds as appropriate. The amount of time required to service the peripherals depends on the content of the request.

To Service Request DL06	DL06
Minimum	9 µs
Run Mode Max.	412 µs
Program Mode Max.	2.5 second

CPU Bus Communication

Some specialty modules can also communicate directly with the CPU via the CPU bus. During this portion of the cycle the CPU completes any CPU bus communications. The actual time required depends on the type of modules installed and the type of request being processed.

Update Clock/Calendar, Special Relays, Special Registers

The clock, calendar, and special relays are updated and loaded into special V-memory locations during this time. This update is performed during both Run and Program Modes.

Modes		DL06
Program Mode	Minimum	12.0 µs
	Maximum	12.0 µs
Run Mode	Minimum	20.0 µs
	Maximum	27.0 µs

NOTE: *The Clock/Calendar is updated while there is energy on the super-capacitor. If the super-capacitor is discharged, the real time and date is lost.*

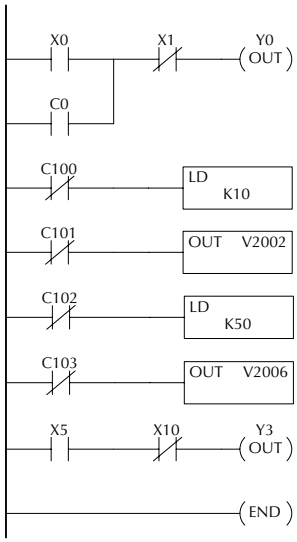
Application Program Execution

The CPU processes the program from address 0 to the END instruction. The CPU executes the program left to right and top to bottom. As each rung is evaluated the appropriate image register or memory location is updated. The time required to solve the application program depends on the type and number of instructions used, and the amount of execution overhead.

Just add the execution times for all the instructions in your program to determine to total execution time. Appendix C provides a complete list of the instruction execution times for the DL06 Micro PLC. For example, the execution time for running the program shown below is calculated as follows:

Instruction	Time
STR X0	.67 µs
OR C0	.51 µs
ANDN X1	.51 µs
OUT Y0	1.82 µs
STRN C100	.67 µs
LD K10	9.00 µs
STRN C101	.67 µs
OUT V2002	9.3 µs
STRN C102	.67 µs
LD K50	9.00 µs
STRN C103	.67 µs
OUT V2006	1.82 µs
STR X5	.67 µs
ANDN X10	.51 µs
OUT Y3	1.82 µs
END	12.80 µs
SUBTOTAL	51.11 µs

Overhead	DL06
Minimum	746.2 µs
Maximum	4352.4 µs



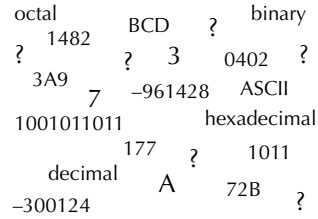
TOTAL TIME = (Program execution time + Overhead) x 1.18

The program above takes only 51.11 µs to execute during each scan. The DL06 spends 0.18ms on internal timed interrupt management, for every 1ms of instruction time. The total scan time is calculated by adding the program execution time to the overhead (shown above) and multiplying the result (ms) by 1.18. Overhead includes all other housekeeping and diagnostic tasks. The scan time will vary slightly from one scan to the next, because of fluctuation in overhead tasks.

Program Control Instructions — the DL06 CPUs offer additional instructions that can change the way the program executes. These instructions include FOR/NEXT loops, Subroutines, and Interrupt Routines. These instructions can interrupt the normal program flow and affect the program execution time. Chapter 5 provides detailed information on how these different types of instructions operate.

PLC Numbering Systems

If you are a new PLC user or are using ***AutomationDirect*** PLCs for the first time, please take a moment to study how our PLCs use numbers. You'll find that each PLC manufacturer has their own conventions on the use of numbers in their PLCs. We want to take just a moment to familiarize you with how numbers are used in ***AutomationDirect*** PLCs. The information you learn here applies to all of our PLCs.

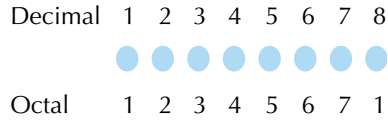


As any good computer does, PLCs store and manipulate numbers in binary form - just ones and zeros. So, why do we have to deal with numbers in so many different forms? Numbers have meaning, and some representations are more convenient than others for particular purposes. Sometimes we use numbers to represent a size or amount of something. Other numbers refer to locations or addresses, or to time. In science we attach engineering units to numbers to give a particular meaning (see Appendix I for numbering system details).

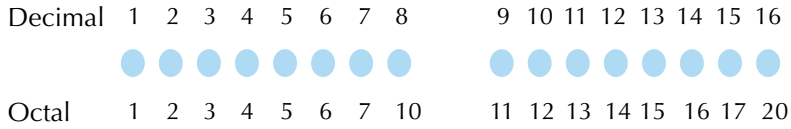
PLC Resources

PLCs offer a fixed amount of resources, depending on the model and configuration. We use the word **resources** to include variable memory (V-memory), I/O points, timers, counters, etc. Most modular PLCs allow you to add I/O points in groups of eight. In fact, all the resources of our PLCs are counted in octal. It's easier for computers to count in groups of eight than ten, because eight is an even power of 2 (see Appendix I for more details).

Octal means simply counting in groups of eight things at a time. In the figure to the right, there are eight circles. The quantity in decimal is 8, but in octal it is 10 (8 and 9 are not valid in octal). In octal, 10 means 1 group of 8 plus 0 (no individuals)

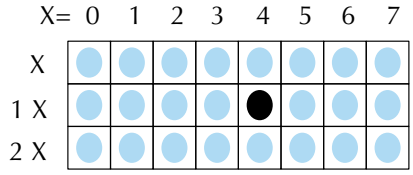


In the figure below, we have two groups of eight circles.
Counting in octal we have 20 items, meaning 2 groups of eight, plus 0 individuals Don't say "twenty", say "two–zero octal". This makes a clear distinction between number systems.



After counting PLC resources, it's time to access PLC resources (there's a difference). The CPU instruction set accesses resources of the PLC using octal addresses. Octal addresses are the same as octal quantities, except they start counting at zero. The number zero is significant to a computer, so we don't skip it.

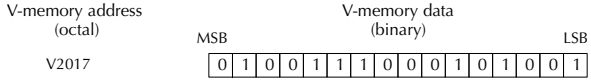
Our circles are in an array of square containers to the right. To access a resource, our PLC instruction will address its location using the octal references shown. If these were counters, **CT14** would access the black circle location.



V–Memory

Variable memory (V-memory) stores data for the ladder program and for configuration settings. V-memory locations and V-memory addresses are the same thing, and are numbered in octal. For example, V2073 is a valid location, while V1983 is not valid (9 and 8 are not valid octal digits).

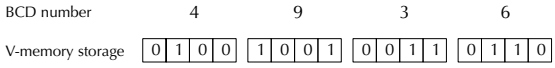
Each V-memory location is one data word wide, meaning 16 bits. For configuration registers, our manuals will show each bit of a V-memory word. The least significant bit (LSB) will be on the right, and the most significant bit (MSB) on the left. We use the word "significant", referring to the relative binary weighting of the bits.



V-memory data is 16-bit binary, but we rarely program the data registers one bit at a time. We use instructions or viewing tools that let us work with decimal, octal, and hexadecimal numbers. All these are converted and stored as binary for us.

A frequently-asked question is "How do I tell if a number is octal, BCD, or hex?" The answer is that we usually cannot tell just by looking at the data ... but it does not really matter. What matters is, the source or mechanism which writes data into a V-memory location and the thing which later reads it must both use the same data type (i.e., octal, hex, binary, or whatever). The V-memory location is just a storage box ... that's all. It does not convert or move the data on its own.

Binary-Coded Decimal Numbers



Since humans naturally count in decimal (10 fingers, 10 toes), we prefer to enter and view PLC data in decimal as well. However, computers are more efficient in using pure binary numbers. A compromise solution between the two is Binary-Coded Decimal (BCD) representation. A BCD digit ranges from 0 to 9, and is stored as four binary bits (a nibble). This permits each V-memory location to store four BCD digits, with a range of decimal numbers from 0000 to 9999.

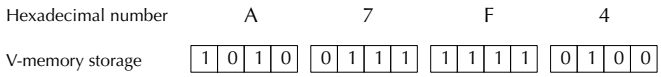
In a pure binary sense, a 16-bit word can represent numbers from 0 to 65535. In storing BCD numbers, the range is reduced to only 0 to 9999. Many math instructions use Binary-Coded Decimal (BCD) data, and ***Direct***SOFT and the handheld programmer allow us to enter and view data in BCD.

Hexadecimal Numbers

Hexadecimal numbers are similar to BCD numbers, except they utilize all possible binary values in each 4-bit digit. They are base-16 numbers so we need 16 different digits. To extend our decimal digits 0 through 9, we use A through F as shown.

Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Hexadecimal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

A 4-digit hexadecimal number can represent all 65536 values in a V-memory word. The range is from 0000 to FFFF (hex). PLCs often need this full range for sensor data, etc. Hexadecimal is just a convenient way for humans to view full binary data.

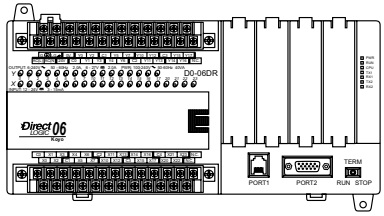


Memory Map

With any PLC system, you generally have many different types of information to process. This includes input device status, output device status, various timing elements, parts counts, etc. It is important to understand how the system represents and stores the various types of data. For example, you need to know how the system identifies input points, output points, data words, etc. The following paragraphs discuss the various memory types used in DL06 Micro PLCs. A memory map overview for the CPU follows the memory descriptions.

Octal Numbering System

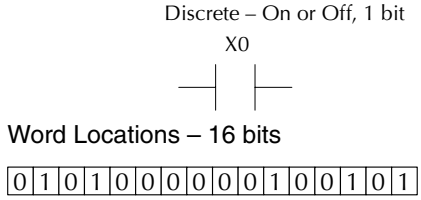
All memory locations and resources are numbered in Octal (base 8). For example, the diagram shows how the octal numbering system works for the discrete input points. Notice the octal system does not contain any numbers with the digits 8 or 9.



Discrete and Word Locations

As you examine the different memory types, you'll notice two types of memory in the DL06, discrete and word memory. Discrete memory is one bit that can be either a 1 or a 0. Word memory is referred to as V-memory (variable) and is a 16-bit location normally used to manipulate data/numbers, store data/numbers, etc.

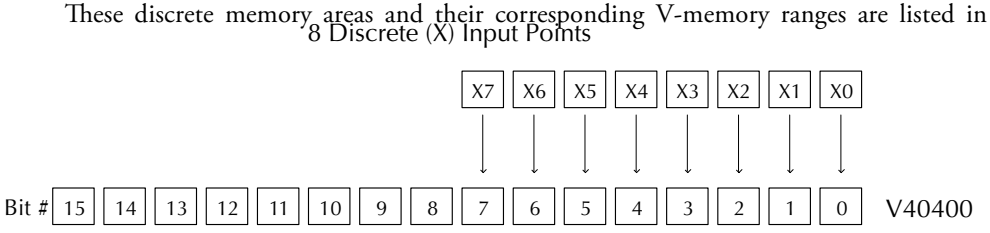
Some information is automatically stored in V-memory. For example, the timer current values are stored in V-memory.



V-memory Locations for Discrete Memory Areas

The discrete memory area is for inputs, outputs, control relays, special relays, stages, timer status bits and counter status bits. However, you can also access the bit data types as a V-memory word. Each V-memory location contains 16 consecutive discrete locations. For example, the following diagram shows how the X input points are mapped into V-memory locations.

These discrete memory areas and their corresponding V-memory ranges are listed in the memory area table for DL06 Micro PLCs on the following pages.



Input Points (X Data Type)

The discrete input points are noted by an X data type. There are 20 discrete input points and 256 discrete input addresses available with DL06 CPUs. In this example, the output point Y0 will be turned on when input X0 energizes.

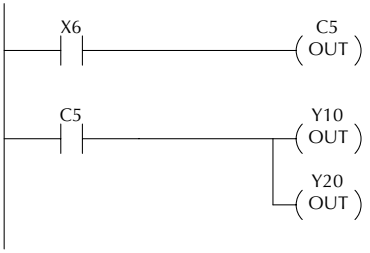
Output Points (Y Data Type)

The discrete output points are noted by a Y data type. There are 16 discrete outputs and 256 discrete output addresses available with DL06 CPUs. In this example, output point Y1 will be turned on when input X1 energizes.

Control Relays (C Data Type)

Control relays are discrete bits normally used to control the user program. The control relays do not represent a real world device, that is, they cannot be physically tied to switches, output coils, etc. There are 1024 control relays internal to the CPU. Because of this, control relays can be programmed as discrete inputs or discrete outputs. These locations are used in programming the discrete memory locations (C) or the corresponding word location which contains 16 consecutive discrete locations.

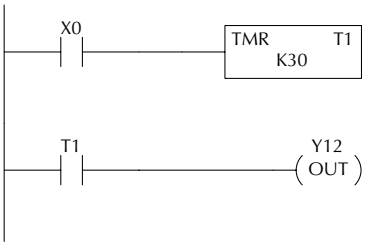
In this example, memory location C5 will energize when input X6 turns on. The second rung shows a simple example of how to use a control relay as an input.



Timers and Timer Status Bits (T Data Type)

There are 256 timers available in the CPU. Timer status bits reflect the relationship between the current value and the preset value of a specified timer. The timer status bit will be on when the current value is equal or greater than the preset value of a corresponding timer.

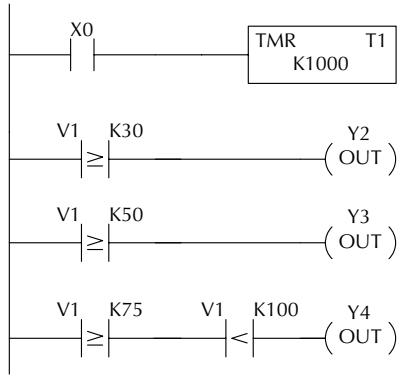
When input X0 turns on, timer T1 will start. When the timer reaches the preset of 3 seconds (K of 30) timer status contact T1 turns on. When T1 turns on, output Y12 turns on. Turning off X0 resets the timer.



Timer Current Values (V Data Type)

As mentioned earlier, some information is automatically stored in V-memory. This is true for the current values associated with timers. For example: V0 holds the current value for Timer 0; V1 holds the current value for Timer 1; and so on. These can also be designated as TA0 (Timer Accumulated) for Timer 0, and TA1 for Timer 1.

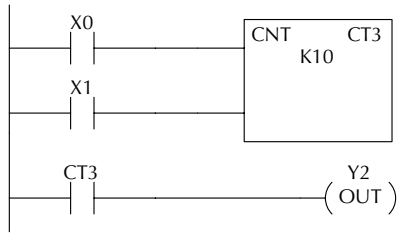
The primary reason for this is programming flexibility. The example shows how you can use relational contacts to monitor several time intervals from a single timer.



Counters and Counter Status Bits (CT Data type)

There are 128 counters available in the CPU. Counter status bits that reflect the relationship between the current value and the preset value of a specified counter. The counter status bit will be on when the current value is equal to or greater than the preset value of a corresponding counter.

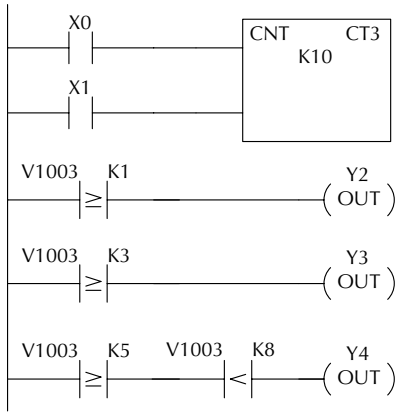
Each time contact X0 transitions from off to on, the counter increments by one. (If X1 comes on, the counter is reset to zero.) When the counter reaches the preset of 10 counts (K of 10) counter status contact CT3 turns on. When CT3 turns on, output Y2 turns on.



Counter Current Values (V Data Type)

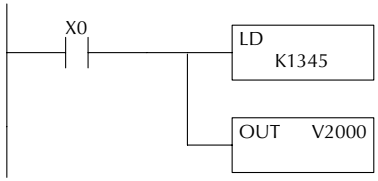
Just like the timers, the counter current values are also automatically stored in V-memory. For example, V1000 holds the current value for Counter CT0, V1001 holds the current value for Counter CT1, etc. These can also be designated as CTA0 (Counter Accumulated) for Counter 0 and CTA01 for Counter 1.

The primary reason for this is programming flexibility. The example shows how you can use relational contacts to monitor the counter values.

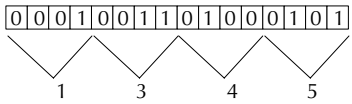


Word Memory (V Data Type)

Word memory is referred to as V-memory (variable) and is a 16-bit location normally used to manipulate data/numbers, store data/numbers, etc. Some information is automatically stored in V-memory. For example, the timer current values are stored in V-memory. The example shows how a four-digit BCD constant is loaded into the accumulator and then stored in a V-memory location.



Word Locations – 16 bits

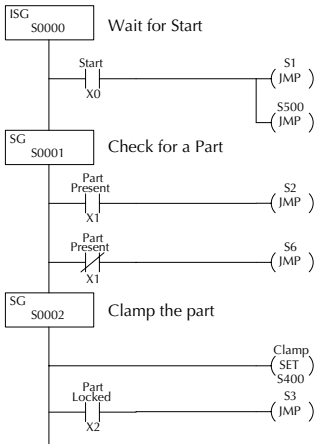


Stages (S Data type)

Stages are used in RLL^{PLUS} programs to create a structured program, similar to a flowchart. Each program Stage denotes a program segment. When the program segment, or Stage, is active, the logic within that segment is executed. If the Stage is off, or inactive, the logic is not executed and the CPU skips to the next active Stage. (See Chapter 7 for a more detailed description of RLL^{PLUS} programming.)

Each Stage also has a discrete status bit that can be used as an input to indicate whether the Stage is active or inactive. If the Stage is active, then the status bit is on. If the Stage is inactive, then the status bit is off. This status bit can also be turned on or off by other instructions, such as the SET or RESET instructions. This allows you to easily control stages throughout the program.

Ladder Representation



Special Relays (SP Data Type)

Special relays are discrete memory locations with predefined functionality. There are many different types of special relays. For example, some aid in program development, others provide system operating status information, etc. Appendix D provides a complete listing of the special relays.

In this example, control relay C10 will energize for 50ms and de-energize for 50ms because SP5 is a predefined relay that will be on for 50ms and off for 50ms.

SP4: 1 second clock
SP5: 100 ms clock
SP6: 50 ms clock

DL06 System V-memory

System Parameters and Default Data Locations (V Data Type)

The DL06 PLCs reserve several V-memory locations for storing system parameters or certain types of system data. These memory locations store things like the error codes, High-Speed I/O data, and other types of system setup information.

System V-memory	Description of Contents	Default Values / Ranges	Read Only R/W
V700-V707	Sets the V-memory location for option card in slot 1	N/A	R/W
V710-V717	Sets the V-memory location for option card in slot 2	N/A	R/W
V720-V727	Sets the V-memory location for option card in slot 3	N/A	R/W
V730-V737	Sets the V-memory location for option card in slot 4	N/A	R/W
V3630–V3707	The default location for multiple preset values for UP/DWN and UP Counter 1 or pulse catch function	N/A	R/W
V3710-V3767	The default location for multiple preset values for UP/DWN and UP Counter 2	N/A	R/W
V7620	DV-1000 Sets the V-memory location that contains the value	V0 – V3760	R/W
V7621	DV-1000 Sets the V-memory location that contains the message	V0 – V3760	R/W
V7622	DV-1000 Sets the total number (1 – 32) of V-memory locations to be displayed	1 - 32	R/W
V7623	DV-1000 Sets the V-memory location containing the numbers to be displayed	V0 – V3760	R/W
V7624	DV-1000 Sets the V-memory location that contains the character code to be displayed	V0 – V3760	R/W
V7625	DV-1000 Contains the function number that can be assigned to each key	V-memory for X, Y, or C	R/W
V7626	DV-1000 Powerup operational mode	0,1, 2, 3, 12	R/W
V7627	Change preset value	0000 to 9999	R/W
V7630	Starting location for the multi–step presets for channel 1. The default value is 3630, which indicates the first value should be obtained from V3630. Since there are 24 presets available, the default range is V3630 – V3707. You can change the starting point if necessary.	Default: V3630 Range: V0- V3710	R/W
V7631	Starting location for the multi–step presets for channel 2. The default value is 3710, which indicates the first value should be obtained from V3710. Since there are 24 presets available, the default range is V3710 – V3767. You can change the starting point if necessary.	Default: V3710 Range: V0- V3710	R/W
V7632	Setup Register for Pulse Output	N/A	R/W
V7633	Sets the desired function code for the high speed counter, interrupt, pulse catch, pulse train, and input filter. This location can also be used to set the power-up in Run Mode option.	Default: 0060 Lower Byte Range: 10 – Counter 20 – Quadrature 30 – Pulse Out 40 – Interrupt 50 – Pulse Catch 60 – Filtered discrete In. Upper Byte Range: Bits 8–11, 14, 15: Unused, Bit 13: Power–up in RUN, only if Mode Switch is in TERM position. Bit 12 is used to enable the low battery indications.	R/W
V7634	X0 Setup Register for High-Speed I/O functions for input X0	Default: 1006	R/W
V7635	X1 Setup Register for High-Speed I/O functions for input X1	Default: 1006	R/W
V7636	X2 Setup Register for High-Speed I/O functions for input X2	Default: 1006	R/W
V7637	X3 Setup Register for High-Speed I/O functions for input X3	Default: 1006	R/W
V7640	PID Loop table beginning address	V1200 - V7377 V10000-V17777	R/W
V7641	Number of PID loops enabled	1-8	R/W
V7642	Error Code - PID Loop Table		R
V7643-V7646	DirectSoft I-Box instructions work area		R
V7647	Timed Interrupt		R/W
V7653	Port 2: Terminate code setting Non-procedure		R/W
V7655	Port 2: Setup for the protocol, time-out, and the response delay time		R/W

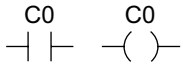
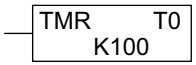
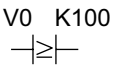
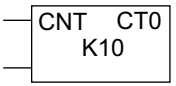
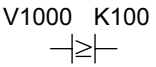
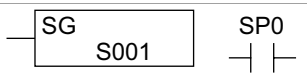
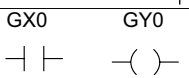
System V-memory	Description of Contents	Default Values / Ranges	Read Only R/W
V7656	Port 2: Setup for the station number, baud rate, STOP bit, and parity		R/W
V7657	Port 2: Setup completion code used to notify the completion of the parameter setup	0400h reset port 2	R/W
V7660	Scan control setup: Keeps the scan control mode		R/W
V7661	Setup timer over counter		R
V7662–V7710	Reserved		R/W
V7711-V7717	DirectSOFT I-Box instructions work area		R
V7720–V7722	Locations for DV–1000 operator interface parameters		R/W
V7720	Location for DV-1000 operation interface Titled Timer preset value pointer		R/W
V7721	DV-1000: Title Counter preset value pointer		R/W
V7722	DV-1000: Hibyte-Titled, Lobyte-Timer preset block size		R/W
V7723–V7725	DirectSOFT I-Box instructions work area		R
V7726	Reserved		R/W
V7727	Version No		R
V7730	D0-DCM Module Slot1 Auto Reset Timeout		R/W
V7731	D0-DCM Module Slot2 Auto Reset Timeout		R/W
V7732	D0-DCM Module Slot3 Auto Reset Timeout		R/W
V7733	D0-DCM Module Slot4 Auto Reset Timeout		R/W
V7734-V7737	Reserved		R/W
V7740	Port 2: Communication Auto Reset Timer Setup	Default: 3030	R/W
V7741	Reserved		R/W
V7742	LCD Various LCD setting flags		R/W
V7743	V Memory address in which the default display message is stored as set		R/W
V7744-V7746	Reserved		R/W
V7747	Location contains a 10 ms counter (0-99). This location increments once every 10 ms		R
V7750	Reserved		R/W
V7751	Fault Message Error Code		R
V7752	I/O Configuration Error: stores the module ID code for the module that does not the current configuration		R
V7753	I/O Configuration Error: stores the module ID code		R
V7754	I/O Configuration Error: identifies the base and slot number		R
V7755	Error code — stores the fatal error code		R
V7756	Error code — stores the major error code		R
V7757	Error code — stores the minor error code		R
V7760–V7762	Reserved		R/W
V7763	Program address where syntax error exists		R
V7764	Syntax error code		R
V7765	Scan counter — stores the total number of scan cycles that have occurred since the last Program Mode to Run Mode transition (in decimal)		R
V7766	Contains the number of seconds on the clock (00-59)		R
V7767	Contains the number of minutes on the clock (00-59)		R
V7770	Contains the number of hours on the clock (00-23)		R
V7771	Contains the day of the week (Mon., Tues., Wed., etc.)		R
V7772	Contains the day of the month (01, 02, etc.)		R
V7773	Contains the month (01 to 12)		R
V7774	Contains the year (00 to 99)		R
V7775	Scan — stores the current scan time (milliseconds)		R
V7776	Scan — stores the minimum scan time that has occurred since the last Program Mode to Run Mode transition (milliseconds)		R
V7777	Scan — stores the maximum scan rate since the last power cycle (milliseconds)		R

DL06 Aliases

An alias is an alternate way of referring to certain memory types, such as timer/counter current values, V-memory locations for I/O points, etc., which simplifies understanding the memory address. The use of the alias is optional, but some users may find the alias to be helpful when developing a program. The table below shows how the aliases can be used.

DL06 Aliases		
Address Start	**Alias Start**	**Example**
V0	TA0	V0 is the timer accumulator value for timer 0, therefore, its alias is TA0. TA1 is the alias for V1, etc..
V1000	CTA0	V1000 is the counter accumulator value for counter 0, therefore, its alias is CTA0. CTA1 is the alias for V1001, etc.
V40000	VGX	V40000 is the word memory reference for discrete bits GX0 through GX17, therefore, its alias is VGX0. V40001 is the word memory reference for discrete bits GX20 through GX 37, therefore, its alias is VGX20.
V40200	VGY	V40200 is the word memory reference for discrete bits GY0 through GY17, therefore, its alias is VGY0. V40201 is the word memory reference for discrete bits GY20 through GY 37, therefore, its alias is VGY20.
V40400	VX0	V40400 is the word memory reference for discrete bits X0 through X17, therefore, its alias is VX0. V40401 is the word memory reference for discrete bits X20 through X37, therefore, its alias is VX20.
V40500	VY0	V40500 is the word memory reference for discrete bits Y0 through Y17, therefore, its alias is VY0. V40501 is the word memory reference for discrete bits Y20 through Y37, therefore, its alias is VY20.
V40600	VC0	V40600 is the word memory reference for discrete bits C0 through C17, therefore, its alias is VC0. V40601 is the word memory reference for discrete bits C20 through C37, therefore, its alias is VC20.
V41000	VS0	V41000 is the word memory reference for discrete bits S0 through S17, therefore, its alias is VS0. V41001 is the word memory reference for discrete bits S20 through S37, therefore, its alias is VS20.
V41100	VT0	V41100 is the word memory reference for discrete bits T0 through T17, therefore, its alias is VT0. V41101 is the word memory reference for discrete bits T20 through T37, therefore, its alias is VT20.
V41140	VCT0	V41140 is the word memory reference for discrete bits CT0 through CT17, therefore, its alias is VCT0. V41141 is the word memory reference for discrete bits CT20 through CT37, therefore, its alias is VCT20.
V41200	VSP0	V41200 is the word memory reference for discrete bits SP0 through SP17, therefore, its alias is VSP0. V41201 is the word memory reference for discrete bits SP20 through SP37, therefore, its alias is VSP20.

DL06 Memory Map

Memory Type	Discrete Memory Reference (octal)	Word Memory Reference (octal)	Decimal	Symbol
Input Points	X0 – X777	V40400 - V40437	512	
Output Points	Y0 – Y777	V40500 – V40537	512	
Control Relays	C0 – C1777	V40600 - V40677	1024	
Special Relays	SP0 – SP777	V41200 – V41237	512	
Timers	T0 – T377	V41100 – V41117	256	
Timer Current Values	None	V0 – V377	256	
Timer Status Bits	T0 – T377	V41100 – V41117	256	
Counters	CT0 – CT177	V41140 – V41147	128	
Counter Current Values	None	V1000 – V1177	128	
Counter Status Bits	CT0 – CT177	V41140 – V41147	128	
Data Words (See Appendix F)	None	V400-V677 V1200 – V7377 V10000 - V17777	192 3200 4096	None specific, used with many instructions.
Data Words EEPROM (See Appendix F)	None	V7400 – V7577	128	None specific, used with many instructions. May be non-volatile if MOV inst. is used. Data can be rewritten to EEPROM at least 100,000 times before it fails.
Stages	S0 – S1777	V41000 – V41077	1024	
Remote I/O (future use) (See Note 1)	GX0-GX3777 GY0-GY3777	V40000-V40177 V40200-V40377	2048 2048	
System parameters	None	V700-V777 V7600 – V7777 V36000-V37777	64 128 1024	None specific, used for various purposes

NOTE 1: *This area can be used for additional Data Words.*
NOTE 2: *The DL06 systems have 20 fixed discrete inputs and 16 fixed discrete outputs, but the total can be increased by up to 64 inputs or 64 outputs, or a combination of both.*

X Input/Y Output Bit Map

This table provides a listing of individual input and output points associated with each V-memory address bit for the DL06's twenty integrated physical inputs and 16 integrated physical outputs in addition to up to 64 inputs and 64 outputs for option cards. Actual available references are X0 to X777 (V40400 – V40437) and Y0 to Y777 (V40500 - V40537).

MSB					DL06 Input (X) and Output (Y) Points											LSB	X Input	Y Output
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		Address	Address
017	016	015	014	013	012	011	010	007	006	005	004	003	002	001	000		V40400	V40500
037	036	035	034	033	032	031	030	027	026	025	024	023	022	021	020		V40401	V40501
057	056	055	054	053	052	051	050	047	046	045	044	043	042	041	040		V40402	V40502
077	076	075	074	073	072	071	070	067	066	065	064	063	062	061	060		V40403	V40503
117	116	115	114	113	112	111	110	107	106	105	104	103	102	101	100		V40404	V40504
137	136	135	134	133	132	131	130	127	126	125	124	123	122	121	120		V40405	V40505
157	156	155	154	153	152	151	150	147	146	145	144	143	142	141	140		V40406	V40506
177	176	175	174	173	172	171	170	167	166	165	164	163	162	161	160		V40407	V40507
217	216	215	214	213	212	211	210	207	206	205	204	203	202	201	200		V40410	V40510
237	236	235	234	233	232	231	230	227	226	225	224	223	222	221	220		V40411	V40511
257	256	255	254	253	252	251	250	247	246	245	244	243	242	241	240		V40412	V40512
277	276	275	274	273	272	271	270	267	266	265	264	263	262	261	260		V40413	V40513
317	316	315	314	313	312	311	310	307	306	305	304	303	302	301	300		V40414	V40514
337	336	335	334	333	332	331	330	327	326	325	324	323	322	321	320		V40415	V40515
357	356	355	354	353	352	351	350	347	346	345	344	343	342	341	340		V40416	V40516
377	376	375	374	373	372	371	370	367	366	365	364	363	362	361	360		V40417	V40517
417	416	415	414	413	412	411	410	407	406	405	404	403	402	401	400		V40420	V40520
437	436	435	434	433	432	431	430	427	426	425	424	423	422	421	420		V40421	V40521
457	456	455	454	453	452	451	450	447	446	445	444	443	442	441	440		V40422	V40522
477	476	475	474	473	472	471	470	467	466	465	464	463	462	461	460		V40423	V40523
517	516	515	514	513	512	511	510	507	506	505	504	503	502	501	500		V40424	V40524
537	536	535	534	533	532	531	530	527	526	525	524	523	522	521	520		V40425	V40525
557	556	555	554	553	552	551	550	547	546	545	544	543	542	541	540		V40426	V40526
577	576	575	574	573	572	571	570	567	566	565	564	563	562	561	560		V40427	V40527
617	616	615	614	613	612	611	610	607	606	605	604	603	602	601	600		V40430	V40530
637	636	635	634	633	632	631	630	627	626	625	624	623	622	621	620		V40431	V40531
657	656	655	654	653	652	651	650	647	646	645	644	643	642	641	640		V40432	V40532
677	676	675	674	673	672	671	670	667	666	665	664	663	662	661	660		V40433	V40533
717	716	715	714	713	712	711	710	707	706	705	704	703	702	701	700		V40434	V40534
737	736	735	734	733	732	731	730	727	726	725	724	723	722	721	720		V40435	V40535
757	756	755	754	753	752	751	750	747	746	745	744	743	742	741	740		V40436	V40536
777	776	775	774	773	772	771	770	767	766	765	764	763	762	761	760		V40437	V40537

Stage Control/Status Bit Map

This table provides a listing of individual Stage control bits associated with each V-memory address bit.

MSB	DL06 Stage (S) Control Bits															LSB	Address
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
017	016	015	014	013	012	011	010	007	006	005	004	003	002	001	000	V41000
037	036	035	034	033	032	031	030	027	026	025	024	023	022	021	020	V41001
057	056	055	054	053	052	051	050	047	046	045	044	043	042	041	040	V41002
077	076	075	074	073	072	071	070	067	066	065	064	063	062	061	060	V41003
117	116	115	114	113	112	111	110	107	106	105	104	103	102	101	100	V41004
137	136	135	134	133	132	131	130	127	126	125	124	123	122	121	120	V41005
157	156	155	154	153	152	151	150	147	146	145	144	143	142	141	140	V41006
177	176	175	174	173	172	171	170	167	166	165	164	163	162	161	160	V41007
217	216	215	214	213	212	211	210	207	206	205	204	203	202	201	200	V41010
237	236	235	234	233	232	231	230	227	226	225	224	223	222	221	220	V41011
257	256	255	254	253	252	251	250	247	246	245	244	243	242	241	240	V41012
277	276	275	274	273	272	271	270	267	266	265	264	263	262	261	260	V41013
317	316	315	314	313	312	311	310	307	306	305	304	303	302	301	300	V41014
337	336	335	334	333	332	331	330	327	326	325	324	323	322	321	320	V41015
357	356	355	354	353	352	351	350	347	346	345	344	343	342	341	340	V41016
377	376	375	374	373	372	371	370	367	366	365	364	363	362	361	360	V41017
417	416	415	414	413	412	411	410	407	406	405	404	403	402	401	400	V41020
437	436	435	434	433	432	431	430	427	426	425	424	423	422	421	420	V41021
457	456	455	454	453	452	451	450	447	446	445	444	443	442	441	440	V41022
477	476	475	474	473	472	471	470	467	466	465	464	463	462	461	460	V41023
517	516	515	514	513	512	511	510	507	506	505	504	503	502	501	500	V41024
537	536	535	534	533	532	531	530	527	526	525	524	523	522	521	520	V41025
557	556	555	554	553	552	551	550	547	546	545	544	543	542	541	540	V41026
577	576	575	574	573	572	571	570	567	566	565	564	563	562	561	560	V41027
617	616	615	614	613	612	611	610	607	606	605	604	603	602	601	600	V41030
637	636	635	634	633	632	631	630	627	626	625	624	623	622	621	620	V41031
657	656	655	654	653	652	651	650	647	646	645	644	643	642	641	640	V41032
677	676	675	674	673	672	671	670	667	666	665	664	663	662	661	660	V41033
717	716	715	714	713	712	711	710	707	706	705	704	703	702	701	700	V41034
737	736	735	734	733	732	731	730	727	726	725	724	723	722	721	720	V41035
757	756	755	754	753	752	751	750	747	746	745	744	743	742	741	740	V41036
777	776	775	774	773	772	771	770	767	766	765	764	763	762	761	760	V41037

This table is continued on the next page.

MSB	DL06 Stage (S) Control Bits (cont'd)														LSB	Address
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
1017	1016	1015	1014	1013	1012	1011	1010	1007	1006	1005	1004	1003	1002	1001	1000	V41040
1037	1036	1035	1034	1033	1032	1031	1030	1027	1026	1025	1024	1023	1022	1021	1020	V41041
1057	1056	1055	1054	1053	1052	1051	1050	1047	1046	1045	1044	1043	1042	1041	1040	V41042
1077	1076	1075	1074	1073	1072	1071	1070	1067	1066	1065	1064	1063	1062	1061	1060	V41043
1117	1116	1115	1114	1113	1112	1111	1110	1107	1106	1105	1104	1103	1102	1101	1100	V41044
1137	1136	1135	1134	1133	1132	1131	1130	1127	1126	1125	1124	1123	1122	1121	1120	V41045
1157	1156	1155	1154	1153	1152	1151	1150	1147	1146	1145	1144	1143	1142	1141	1140	V41046
1177	1176	1175	1174	1173	1172	1171	1170	1167	1166	1165	1164	1163	1162	1161	1160	V41047
1217	1216	1215	1214	1213	1212	1211	1210	1207	1206	1205	1204	1203	1202	1201	1200	V41050
1237	1236	1235	1234	1233	1232	1231	1230	1227	1226	1225	1224	1223	1222	1221	1220	V41051
1257	1256	1255	1254	1253	1252	1251	1250	1247	1246	1245	1244	1243	1242	1241	1240	V41052
1277	1276	1275	1274	1273	1272	1271	1270	1267	1266	1265	1264	1263	1262	1261	1260	V41053
1317	1316	1315	1314	1313	1312	1311	1310	1307	1306	1305	1304	1303	1302	1301	1300	V41054
1337	1336	1335	1334	1333	1332	1331	1330	1327	1326	1325	1324	1323	1322	1321	1320	V41055
1357	1356	1355	1354	1353	1352	1351	1350	1347	1346	1345	1344	1343	1342	1341	1340	V41056
1377	1376	1375	1374	1373	1372	1371	1370	1367	1366	1365	1364	1363	1362	1361	1360	V41057
1417	1416	1415	1414	1413	1412	1411	1410	1407	1406	1405	1404	1403	1402	1401	1400	V41060
1437	1436	1435	1434	1433	1432	1431	1430	1427	1426	1425	1424	1423	1422	1421	1420	V41061
1457	1456	1455	1454	1453	1452	1451	1450	1447	1446	1445	1444	1443	1442	1441	1440	V41062
1477	1476	1475	1474	1473	1472	1471	1470	1467	1466	1465	1464	1463	1462	1461	1460	V41063
1517	1516	1515	1514	1513	1512	1511	1510	1507	1506	1505	1504	1503	1502	1501	1500	V41064
1537	1536	1535	1534	1533	1532	1531	1530	1527	1526	1525	1524	1523	1522	1521	1520	V41065
1557	1556	1555	1554	1553	1552	1551	1550	1547	1546	1545	1544	1543	1542	1541	1540	V41066
1577	1576	1575	1574	1573	1572	1571	1570	1567	1566	1565	1564	1563	1562	1561	1560	V41067
1617	1616	1615	1614	1613	1612	1611	1610	1607	1606	1605	1604	1603	1602	1601	1600	V41070
1637	1636	1635	1634	1633	1632	1631	1630	1627	1626	1625	1624	1623	1622	1621	1620	V41071
1657	1656	1655	1654	1653	1652	1651	1650	1647	1646	1645	1644	1643	1642	1641	1640	V41072
1677	1676	1675	1674	1673	1672	1671	1670	1667	1666	1665	1664	1663	1662	1661	1660	V41073
1717	1716	1715	1714	1713	1712	1711	1710	1707	1706	1705	1704	1703	1702	1701	1700	V41074
1737	1736	1735	1734	1733	1732	1731	1730	1727	1726	1725	1724	1723	1722	1721	1720	V41075
1757	1756	1755	1754	1753	1752	1751	1750	1747	1746	1745	1744	1743	1742	1741	1740	V41076
1777	1776	1775	1774	1773	1772	1771	1770	1767	1766	1765	1764	1763	1762	1761	1760	V41077

Control Relay Bit Map

This table provides a listing of the individual control relays associated with each V-memory address bit.

MSB	DL06 Control Relays (C)															LSB	Address
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
017	016	015	014	013	012	011	010	007	006	005	004	003	002	001	000	V40600
037	036	035	034	033	032	031	030	027	026	025	024	023	022	021	020	V40601
057	056	055	054	053	052	051	050	047	046	045	044	043	042	041	040	V40602
077	076	075	074	073	072	071	070	067	066	065	064	063	062	061	060	V40603
117	116	115	114	113	112	111	110	107	106	105	104	103	102	101	100	V40604
137	136	135	134	133	132	131	130	127	126	125	124	123	122	121	120	V40605
157	156	155	154	153	152	151	150	147	146	145	144	143	142	141	140	V40606
177	176	175	174	173	172	171	170	167	166	165	164	163	162	161	160	V40607
217	216	215	214	213	212	211	210	207	206	205	204	203	202	201	200	V40610
237	236	235	234	233	232	231	230	227	226	225	224	223	222	221	220	V40611
257	256	255	254	253	252	251	250	247	246	245	244	243	242	241	240	V40612
277	276	275	274	273	272	271	270	267	266	265	264	263	262	261	260	V40613
317	316	315	314	313	312	311	310	307	306	305	304	303	302	301	300	V40614
337	336	335	334	333	332	331	330	327	326	325	324	323	322	321	320	V40615
357	356	355	354	353	352	351	350	347	346	345	344	343	342	341	340	V40616
377	376	375	374	373	372	371	370	367	366	365	364	363	362	361	360	V40617
417	416	415	414	413	412	411	410	407	406	405	404	403	402	401	400	V40620
437	436	435	434	433	432	431	430	427	426	425	424	423	422	421	420	V40621
457	456	455	454	453	452	451	450	447	446	445	444	443	442	441	440	V40622
477	476	475	474	473	472	471	470	467	466	465	464	463	462	461	460	V40623
517	516	515	514	513	512	511	510	507	506	505	504	503	502	501	500	V40624
537	536	535	534	533	532	531	530	527	526	525	524	523	522	521	520	V40625
557	556	555	554	553	552	551	550	547	546	545	544	543	542	541	540	V40626
577	576	575	574	573	572	571	570	567	566	565	564	563	562	561	560	V40627
617	616	615	614	613	612	611	610	607	606	605	604	603	602	601	600	V40630
637	636	635	634	633	632	631	630	627	626	625	624	623	622	621	620	V40631
657	656	655	654	653	652	651	650	647	646	645	644	643	642	641	640	V40632
677	676	675	674	673	672	671	670	667	666	665	664	663	662	661	660	V40633
717	716	715	714	713	712	711	710	707	706	705	704	703	702	701	700	V40634
737	736	735	734	733	732	731	730	727	726	725	724	723	722	721	720	V40635
757	756	755	754	753	752	751	750	747	746	745	744	743	742	741	740	V40636
777	776	775	774	773	772	771	770	767	766	765	764	763	762	761	760	V40637

This table is continued on the next page.

MSB	DL06 Control Relays (C) (cont'd)														LSB	Address
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
1017	1016	1015	1014	1013	1012	1011	1010	1007	1006	1005	1004	1003	1002	1001	1000	V40640
1037	1036	1035	1034	1033	1032	1031	1030	1027	1026	1025	1024	1023	1022	1021	1020	V40641
1057	1056	1055	1054	1053	1052	1051	1050	1047	1046	1045	1044	1043	1042	1041	1040	V40642
1077	1076	1075	1074	1073	1072	1071	1070	1067	1066	1065	1064	1063	1062	1061	1060	V40643
1117	1116	1115	1114	1113	1112	1111	1110	1107	1106	1105	1104	1103	1102	1101	1100	V40644
1137	1136	1135	1134	1133	1132	1131	1130	1127	1126	1125	1124	1123	1122	1121	1120	V40645
1157	1156	1155	1154	1153	1152	1151	1150	1147	1146	1145	1144	1143	1142	1141	1140	V40646
1177	1176	1175	1174	1173	1172	1171	1170	1167	1166	1165	1164	1163	1162	1161	1160	V40647

1217	1216	1215	1214	1213	1212	1211	1210	1207	1206	1205	1204	1203	1202	1201	1200	V40650
1237	1236	1235	1234	1233	1232	1231	1230	1227	1226	1225	1224	1223	1222	1221	1220	V40651
1257	1256	1255	1254	1253	1252	1251	1250	1247	1246	1245	1244	1243	1242	1241	1240	V40652
1277	1276	1275	1274	1273	1272	1271	1270	1267	1266	1265	1264	1263	1262	1261	1260	V40653
1317	1316	1315	1314	1313	1312	1311	1310	1307	1306	1305	1304	1303	1302	1301	1300	V40654
1337	1336	1335	1334	1333	1332	1331	1330	1327	1326	1325	1324	1323	1322	1321	1320	V40655
1357	1356	1355	1354	1353	1352	1351	1350	1347	1346	1345	1344	1343	1342	1341	1340	V40656
1377	1376	1375	1374	1373	1372	1371	1370	1367	1366	1365	1364	1363	1362	1361	1360	V40657

1417	1416	1415	1414	1413	1412	1411	1410	1407	1406	1405	1404	1403	1402	1401	1400	V40660
1437	1436	1435	1434	1433	1432	1431	1430	1427	1426	1425	1424	1423	1422	1421	1420	V40661
1457	1456	1455	1454	1453	1452	1451	1450	1447	1446	1445	1444	1443	1442	1441	1440	V40662
1477	1476	1475	1474	1473	1472	1471	1470	1467	1466	1465	1464	1463	1462	1461	1460	V40663
1517	1516	1515	1514	1513	1512	1511	1510	1507	1506	1505	1504	1503	1502	1501	1500	V40664
1537	1536	1535	1534	1533	1532	1531	1530	1527	1526	1525	1524	1523	1522	1521	1520	V40665
1557	1556	1555	1554	1553	1552	1551	1550	1547	1546	1545	1544	1543	1542	1541	1540	V40666
1577	1576	1575	1574	1573	1572	1571	1570	1567	1566	1565	1564	1563	1562	1561	1560	V40667

1617	1616	1615	1614	1613	1612	1611	1610	1607	1606	1605	1604	1603	1602	1601	1600	V40670
1637	1636	1635	1634	1633	1632	1631	1630	1627	1626	1625	1624	1623	1622	1621	1620	V40671
1657	1656	1655	1654	1653	1652	1651	1650	1647	1646	1645	1644	1643	1642	1641	1640	V40672
1677	1676	1675	1674	1673	1672	1671	1670	1667	1666	1665	1664	1663	1662	1661	1660	V40673
1717	1716	1715	1714	1713	1712	1711	1710	1707	1706	1705	1704	1703	1702	1701	1700	V40674
1737	1736	1735	1734	1733	1732	1731	1730	1727	1726	1725	1724	1723	1722	1721	1720	V40675
1757	1756	1755	1754	1753	1752	1751	1750	1747	1746	1745	1744	1743	1742	1741	1740	V40676
1777	1776	1775	1774	1773	1772	1771	1770	1767	1766	1765	1764	1763	1762	1761	1760	V40677

Timer Status Bit Map

This table provides a listing of individual timer contacts associated with each V-memory address bit.

MSB	DL06 Timer (T) Contacts														LSB	Address
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
017	016	015	014	013	012	011	010	007	006	005	004	003	002	001	000	V41100
037	036	035	034	033	032	031	030	027	026	025	024	023	022	021	020	V41101
057	056	055	054	053	052	051	050	047	046	045	044	043	042	041	040	V41102
077	076	075	074	073	072	071	070	067	066	065	064	063	062	061	060	V41103
117	116	115	114	113	112	111	110	107	106	105	104	103	102	101	100	V41104
137	136	135	134	133	132	131	130	127	126	125	124	123	122	121	120	V41105
157	156	155	154	153	152	151	150	147	146	145	144	143	142	141	140	V41106
177	176	175	174	173	172	171	170	167	166	165	164	163	162	161	160	V41107
217	216	215	214	213	212	211	210	207	206	205	204	203	202	201	200	V41110
237	236	235	234	233	232	231	230	227	226	225	224	223	222	221	220	V41111
257	256	255	254	253	252	251	250	247	246	245	244	243	242	241	240	V41112
277	276	275	274	273	272	271	270	267	266	265	264	263	262	261	260	V41113
317	316	315	314	313	312	311	310	307	306	305	304	303	302	301	300	V41114
337	336	335	334	333	332	331	330	327	326	325	324	323	322	321	320	V41115
357	356	355	354	353	352	351	350	347	346	345	344	343	342	341	340	V41116
377	376	375	374	373	372	371	370	367	366	365	364	363	362	361	360	V41117

Counter Status Bit Map

MSB	DL06 Counter (CT) Contacts														LSB	Address
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
017	016	015	014	013	012	011	010	007	006	005	004	003	002	001	000	V41140
037	036	035	034	033	032	031	030	027	026	025	024	023	022	021	020	V41141
057	056	055	054	053	052	051	050	047	046	045	044	043	042	041	040	V41142
077	076	075	074	073	072	071	070	067	066	065	064	063	062	061	060	V41143
117	116	115	114	113	112	111	110	107	106	105	104	103	102	101	100	V41144
137	136	135	134	133	132	131	130	127	126	125	124	123	122	121	120	V41145
157	156	155	154	153	152	151	150	147	146	145	144	143	142	141	140	V41146
177	176	175	174	173	172	171	170	167	166	165	164	163	162	161	160	V41147

This table provides a listing of individual counter contacts associated with each V-memory address bit.

GX and GY I/O Bit Map

This table provides a listing of the individual global I/O points associated with each V-memory address bit.

MSB					DL06 GX and GY I/O Points						LSB					GX	GY
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Address	Address
017	016	015	014	013	012	011	010	007	006	005	004	003	002	001	000	V40000	V40200
037	036	035	034	033	032	031	030	027	026	025	024	023	022	021	020	V40001	V40201
057	056	055	054	053	052	051	050	047	046	045	044	043	042	041	040	V40002	V40202
077	076	075	074	073	072	071	070	067	066	065	064	063	062	061	060	V40003	V40203
117	116	115	114	113	112	111	110	107	106	105	104	103	102	101	100	V40004	V40204
137	136	135	134	133	132	131	130	127	126	125	124	123	122	121	120	V40005	V40205
157	156	155	154	153	152	151	150	147	146	145	144	143	142	141	140	V40006	V40206
177	176	175	174	173	172	171	170	167	166	165	164	163	162	161	160	V40007	V40207
217	216	215	214	213	212	211	210	207	206	205	204	203	202	201	200	V40010	V40210
237	236	235	234	233	232	231	230	227	226	225	224	223	222	221	220	V40011	V40211
257	256	255	254	253	252	251	250	247	246	245	244	243	242	241	240	V40012	V40212
277	276	275	274	273	272	271	270	267	266	265	264	263	262	261	260	V40013	V40213
317	316	315	314	313	312	311	310	307	306	305	304	303	302	301	300	V40004	V40214
337	336	335	334	333	332	331	330	327	326	325	324	323	322	321	320	V40015	V40215
357	356	355	354	353	352	351	350	347	346	345	344	343	342	341	340	V40016	V40216
377	376	375	374	373	372	371	370	367	366	365	364	363	362	361	360	V40007	V40217
417	416	415	414	413	412	411	410	407	406	405	404	403	402	401	400	V40020	V40220
437	436	435	434	433	432	431	430	427	426	425	424	423	422	421	420	V40021	V40221
457	456	455	454	453	452	451	450	447	446	445	444	443	442	441	440	V40022	V40222
477	476	475	474	473	472	471	470	467	466	465	464	463	462	461	460	V40023	V40223
517	516	515	514	513	512	511	510	507	506	505	504	503	502	501	500	V40024	V40224
537	536	535	534	533	532	531	530	527	526	525	524	523	522	521	520	V40025	V40225
557	556	555	554	553	552	551	550	547	546	545	544	543	542	541	540	V40026	V40226
577	576	575	574	573	572	571	570	567	566	565	564	563	562	561	560	V40027	V40227
617	616	615	614	613	612	611	610	607	606	605	604	603	602	601	600	V40030	V40230
637	636	635	634	633	632	631	630	627	626	625	624	623	622	621	620	V40031	V40231
657	656	655	654	653	652	651	650	647	646	645	644	643	642	641	640	V40032	V40232
677	676	675	674	673	672	671	670	667	666	665	664	663	662	661	660	V40033	V40233
717	716	715	714	713	712	711	710	707	706	705	704	703	702	701	700	V40034	V40234
737	736	735	734	733	732	731	730	727	726	725	724	723	722	721	720	V40035	V40235
757	756	755	754	753	752	751	750	747	746	745	744	743	742	741	740	V40036	V40236
777	776	775	774	773	772	771	770	767	766	765	764	763	762	761	760	V40037	V40237

This table is continued on the next page.

NOTE: *This memory area can be used for additional Data Words.*

MSB	DL06 GX and GY I/O Points (cont'd)															LSB	GX	GY
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Address	Address
1017	1016	1015	1014	1013	1012	1011	1010	1007	1006	1005	1004	1003	1002	1001	1000	V40040	V40240
1037	1036	1035	1034	1033	1032	1031	1030	1027	1026	1025	1024	1023	1022	1021	1020	V40041	V40241
1057	1056	1055	1054	1053	1052	1051	1050	1047	1046	1045	1044	1043	1042	1041	1040	V40042	V40242
1077	1076	1075	1074	1073	1072	1071	1070	1067	1066	1065	1064	1063	1062	1061	1060	V40043	V40243
1117	1116	1115	1114	1113	1112	1111	1110	1107	1106	1105	1104	1103	1102	1101	1100	V40044	V40244
1137	1136	1135	1134	1133	1132	1131	1130	1127	1126	1125	1124	1123	1122	1121	1120	V40045	V40245
1157	1156	1155	1154	1153	1152	1151	1150	1147	1146	1145	1144	1143	1142	1141	1140	V40046	V40246
1177	1176	1175	1174	1173	1172	1171	1170	1167	1166	1165	1164	1163	1162	1161	1160	V40047	V40247
1217	1216	1215	1214	1213	1212	1211	1210	1207	1206	1205	1204	1203	1202	1201	1200	V40050	V40250
1237	1236	1235	1234	1233	1232	1231	1230	1227	1226	1225	1224	1223	1222	1221	1220	V40051	V40251
1257	1256	1255	1254	1253	1252	1251	1250	1247	1246	1245	1244	1243	1242	1241	1240	V40052	V40252
1277	1276	1275	1274	1273	1272	1271	1270	1267	1266	1265	1264	1263	1262	1261	1260	V40053	V40253
1317	1316	1315	1314	1313	1312	1311	1310	1307	1306	1305	1304	1303	1302	1301	1300	V40054	V40254
1337	1336	1335	1334	1333	1332	1331	1330	1327	1326	1325	1324	1323	1322	1321	1320	V40055	V40255
1357	1356	1355	1354	1353	1352	1351	1350	1347	1346	1345	1344	1343	1342	1341	1340	V40056	V40256
1377	1376	1375	1374	1373	1372	1371	1370	1367	1366	1365	1364	1363	1362	1361	1360	V40057	V40257
1417	1416	1415	1414	1413	1412	1411	1410	1407	1406	1405	1404	1403	1402	1401	1400	V40060	V40260
1437	1436	1435	1434	1433	1432	1431	1430	1427	1426	1425	1424	1423	1422	1421	1420	V40061	V40261
1457	1456	1455	1454	1453	1452	1451	1450	1447	1446	1445	1444	1443	1442	1441	1440	V40062	V40262
1477	1476	1475	1474	1473	1472	1471	1470	1467	1466	1465	1464	1463	1462	1461	1460	V40063	V40263
1517	1516	1515	1514	1513	1512	1511	1510	1507	1506	1505	1504	1503	1502	1501	1500	V40064	V40264
1537	1536	1535	1534	1533	1532	1531	1530	1527	1526	1525	1524	1523	1522	1521	1520	V40065	V40265
1557	1556	1555	1554	1553	1552	1551	1550	1547	1546	1545	1544	1543	1542	1541	1540	V40066	V40266
1577	1576	1575	1574	1573	1572	1571	1570	1567	1566	1565	1564	1563	1562	1561	1560	V40067	V40267
1617	1616	1615	1614	1613	1612	1611	1610	1607	1606	1605	1604	1603	1602	1601	1600	V40070	V40270
1637	1636	1635	1634	1633	1632	1631	1630	1627	1626	1625	1624	1623	1622	1621	1620	V40071	V40271
1657	1656	1655	1654	1653	1652	1651	1650	1647	1646	1645	1644	1643	1642	1641	1640	V40072	V40272
1677	1676	1675	1674	1673	1672	1671	1670	1667	1666	1665	1664	1663	1662	1661	1660	V40073	V40273
1717	1716	1715	1714	1713	1712	1711	1710	1707	1706	1705	1704	1703	1702	1701	1700	V40074	V40274
1737	1736	1735	1734	1733	1732	1731	1730	1727	1726	1725	1724	1723	1722	1721	1720	V40075	V40275
1757	1756	1755	1754	1753	1752	1751	1750	1747	1746	1745	1744	1743	1742	1741	1740	V40076	V40276
1777	1776	1775	1774	1773	1772	1771	1770	1767	1766	1765	1764	1763	1762	1761	1760	V40077	V40277

This table is continued on the next page.

NOTE: *This memory area can be used for additional Data Words.*

MSB	DL06 GX and GY I/O Points (cont'd)															LSB	GX	GY
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Address	Address
2017	2016	2015	2014	2013	2012	2011	2010	2007	2006	2005	2004	2003	2002	2001	2000	V40100	V40300
2037	2036	2035	2034	2033	2032	2031	2030	2027	2026	2025	2024	2023	2022	2021	2020	V40101	V40301
2057	2056	2055	2054	2053	2052	2051	2050	2047	2046	2045	2044	2043	2042	2041	2040	V40102	V40302
2077	2076	2075	2074	2073	2072	2071	2070	2067	2066	2065	2064	2063	2062	2061	2060	V40103	V40303
2117	2116	2115	2114	2113	2112	2111	2110	2107	2106	2105	2104	2103	2102	2101	2100	V40104	V40304
2137	2136	2135	2134	2133	2132	2131	2130	2127	2126	2125	2124	2123	2122	2121	2120	V40105	V40305
2157	2156	2155	2154	2153	2152	2151	2150	2147	2146	2145	2144	2143	2142	2141	2140	V40106	V40306
2177	2176	2175	2174	2173	2172	2171	2170	2167	2166	2165	2164	2163	2162	2161	2160	V40107	V40307
2217	2216	2215	2214	2213	2212	2211	2210	2207	2206	2205	2204	2203	2202	2201	2200	V40110	V40310
2237	2236	2235	2234	2233	2232	2231	2230	2227	2226	2225	2224	2223	2222	2221	2220	V40111	V40311
2257	2256	2255	2254	2253	2252	2251	2250	2247	2246	2245	2244	2243	2242	2241	2240	V40112	V40312
2277	2276	2275	2274	2273	2272	2271	2270	2267	2266	2265	2264	2263	2262	2261	2260	V40113	V40313
2317	2316	2315	2314	2313	2312	2311	2310	2307	2306	2305	2304	2303	2302	2301	2300	V40114	V40314
2337	2336	2335	2334	2333	2332	2331	2330	2327	2326	2325	2324	2323	2322	2321	2320	V40115	V40315
2357	2356	2355	2354	2353	2352	2351	2350	2347	2346	2345	2344	2343	2342	2341	2340	V40116	V40316
2377	2376	2375	2374	2373	2372	2371	2370	2367	2366	2365	2364	2363	2362	2361	2360	V40117	V40317
2417	2416	2415	2414	2413	2412	2411	2410	2407	2406	2405	2404	2403	2402	2401	2400	V40120	V40320
2437	2436	2435	2434	2433	2432	2431	2430	2427	2426	2425	2424	2423	2422	2421	2420	V40121	V40321
2457	2456	2455	2454	2453	2452	2451	2450	2447	2446	2445	2444	2443	2442	2441	2440	V40122	V40322
2477	2476	2475	2474	2473	2472	2471	2470	2467	2466	2465	2464	2463	2462	2461	2460	V40123	V40323
2517	2516	2515	2514	2513	2512	2511	2510	2507	2506	2505	2504	2503	2502	2501	2500	V40124	V40324
2537	2536	2535	2534	2533	2532	2531	2530	2527	2526	2525	2524	2523	2522	2521	2520	V40125	V40325
2557	2556	2555	2554	2553	2552	2551	2550	2547	2546	2545	2544	2543	2542	2541	2540	V40126	V40326
2577	2576	2575	2574	2573	2572	2571	2570	2567	2566	2565	2564	2563	2562	2561	2560	V40127	V40327
2617	2616	2615	2614	2613	2612	2611	2610	2607	2606	2605	2604	2603	2602	2601	2600	V40130	V40330
2637	2636	2635	2634	2633	2632	2631	2630	2627	2626	2625	2624	2623	2622	2621	2620	V40131	V40331
2657	2656	2655	2654	2653	2652	2651	2650	2647	2646	2645	2644	2643	2642	2641	2640	V40132	V40332
2677	2676	2675	2674	2673	2672	2671	2670	2667	2666	2665	2664	2663	2662	2661	2660	V40133	V40333
2717	2716	2715	2714	2713	2712	2711	2710	2707	2706	2705	2704	2703	2702	2701	2700	V40134	V40334
2737	2736	2735	2734	2733	2732	2731	2730	2727	2726	2725	2724	2723	2722	2721	2720	V40135	V40335
2757	2756	2755	2754	2753	2752	2751	2750	2747	2736	2735	2734	2733	2732	2731	2730	V40136	V40336
2777	2776	2775	2774	2773	2772	2771	2770	2767	2766	2765	2764	2763	2762	2761	2760	V40137	V40337

This table is continued on the next page.

NOTE: *This memory area can be used for additional Data Words.*

MSB	DL06 GX and GY I/O Points (cont'd)															LSB	GX	GY
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Address	Address
3017	3016	3015	3014	3013	3012	3011	3010	3007	3006	3005	3004	3003	3002	3001	3000	V40140	V40340
3037	3036	3035	3034	3033	3032	3031	3030	3027	3026	3025	3024	3023	3022	3021	3020	V40141	V40341
3057	3056	3055	3054	3053	3052	3051	3050	3047	3046	3045	3044	3043	3042	3041	3040	V40142	V40342
3077	3076	3075	3074	3073	3072	3071	3070	3067	3066	3065	3064	3063	3062	3061	3060	V40143	V40343
3117	3116	3115	3114	3113	3112	3111	3110	3107	3106	3105	3104	3103	3102	3101	3100	V40144	V40344
3137	3136	3135	3134	3133	3132	3131	3130	3127	3126	3125	3124	3123	3122	3121	3120	V40145	V40345
3157	3156	3155	3154	3153	3152	3151	3150	3147	3146	3145	3144	3143	3142	3141	3140	V40146	V40346
3177	3176	3175	3174	3173	3172	3171	3170	3167	3166	3165	3164	3163	3162	3161	3160	V40147	V40347
3217	3216	3215	3214	3213	3212	3211	3210	3207	3206	3205	3204	3203	3202	3201	3200	V40150	V40350
3237	3236	3235	3234	3233	3232	3231	3230	3227	3226	3225	3224	3223	3222	3221	3220	V40151	V40351
3257	3256	3255	3254	3253	3252	3251	3250	3247	3246	3245	3244	3243	3242	3241	3240	V40152	V40352
3277	3276	3275	3274	3273	3272	3271	3270	3267	3266	3265	3264	3263	3262	3261	3260	V40153	V40353
3317	3316	3315	3314	3313	3312	3311	3310	3307	3306	3305	3304	3303	3302	3301	3300	V40154	V40354
3337	3336	3335	3334	3333	3332	3331	3330	3327	3326	3325	3324	3323	3322	3321	3320	V40155	V40355
3357	3356	3355	3354	3353	3352	3351	3350	3347	3346	3345	3344	3343	3342	3341	3340	V40156	V40356
3377	3376	3375	3374	3373	3372	3371	3370	3367	3366	3365	3364	3363	3362	3361	3360	V40157	V40357
3417	3416	3415	3414	3413	3412	3411	3410	3407	3406	3405	3404	3403	3402	3401	3400	V40160	V40360
3437	3436	3435	3434	3433	3432	3431	3430	3427	3426	3425	3424	3423	3422	3421	3420	V40161	V40361
3457	3456	3455	3454	3453	3452	3451	3450	3447	3446	3445	3444	3443	3442	3441	3440	V40162	V40362
3477	3476	3475	3474	3473	3472	3471	3470	3467	3466	3465	3464	3463	3462	3461	3460	V40163	V40363
3517	3516	3515	3514	3513	3512	3511	3510	3507	3506	3505	3504	3503	3502	3501	3500	V40164	V40364
3537	3536	3535	3534	3533	3532	3531	3530	3527	3526	3525	3524	3523	3522	3521	3520	V40165	V40365
3557	3556	3555	3554	3553	3552	3551	3550	3547	3546	3545	3544	3543	3542	3541	3540	V40166	V40366
3577	3576	3575	3574	3573	3572	3571	3570	3567	3566	3565	3564	3563	3562	3561	3560	V40167	V40367
3617	3616	3615	3614	3613	3612	3611	3610	3607	3606	3605	3604	3603	3602	3601	3600	V40170	V40370
3637	3636	3635	3634	3633	3632	3631	3630	3627	3626	3625	3624	3623	3622	3621	3620	V40171	V40371
3657	3656	3655	3654	3653	3652	3651	3650	3647	3646	3645	3644	3643	3642	3641	3640	V40172	V40372
3677	3676	3675	3674	3673	3672	3671	3670	3667	3666	3665	3664	3663	3662	3661	3660	V40173	V40373
3717	3716	3715	3714	3713	3712	3711	3710	3707	3706	3705	3704	3703	3702	3701	3700	V40174	V40374
3737	3736	3735	3734	3733	3732	3731	3730	3727	3726	3725	3724	3723	3722	3721	3720	V40175	V40375
3757	3756	3755	3754	3753	3752	3751	3750	3747	3746	3745	3744	3743	3742	3741	3740	V40176	V40376
3777	3776	3775	3774	3773	3772	3771	3770	3767	3766	3765	3764	3763	3762	3761	3760	V40177	V40377

NOTE: *This memory area can be used for additional Data Words.*

System design and Configuration

In This Chapter...

DL06 System Design Strategies

I/O System Configurations

The DL06 PLCs offer a number of different I/O configurations. Choose the configuration that is right for your application, and keep in mind that the DL06 PLCs offer the ability to add I/O with the use of option cards. Although remote I/O isn't available, there are many option cards available. For instance:

- Various A/C and D/C I/O modules
- Combination I/O modules
- Analog I/O modules
- Combination Analog I/O modules

A DL06 system can be developed using several different arrangements using the option modules. See our DL05/06 Options Modules User Manual (D0-OPTIONS-M) on the website, **www.automationdirect.com** for detailed selection information.

Networking Configurations

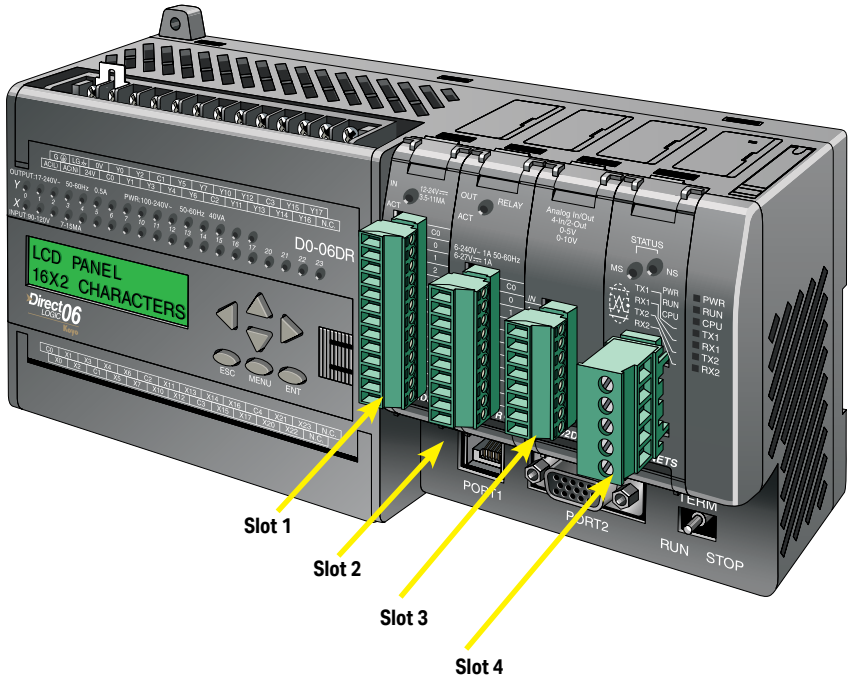
The DL06 PLCs offers the following ways to add networking:

- Ethernet Communications Module s connects a DL06 to high-speed peer-to-peer networks. Any PLC can initiate communications with any other PLC or operator interfaces, such as C-more, when using the ECOM modules.
- Data Communications Modules s connects a DL06 to devices using either DeviceNet or Profibus to link to master controllers, as well as a D0-DCM.
- Communications Port 1 s The DL06 has a 6-pin RJ12 connector on Port 1 that supports (as slave) K-sequence, MODBUS RTU or DirectNET protocols.
- Communications Port 2 s The DL06 has a 15-pin connector on Port 2 that supports either master/slave MODBUS RTU or DirectNET protocols, or K-sequence protocol as slave. (MRX and MWX instructions allow you to enter native MODBUS addressing in your ladder program with no need to perform octal to decimal conversions). Port 2 can also be used for ASCII IN/OUT communications.

Module Placement

Slot Numbering

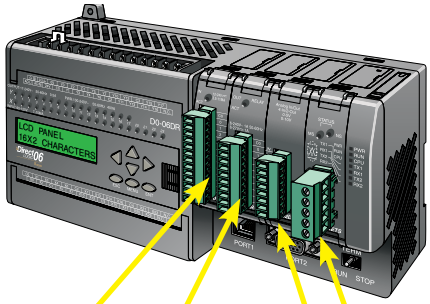
The DL06 has four slots, which are numbered as follows:



Automatic I/O Configuration

The DL06 CPUs automatically detect any installed I/O modules (including specialty modules) at powerup, and establish the correct I/O configuration and addresses. This applies to modules located in the local base. For most applications, you will never have to change the configuration.

I/O addresses use octal numbering, starting at X100 and Y100 in the slot next to the CPU. The addresses are assigned in groups of 8, or 16 depending on the number of points for the I/O module. The discrete input and output modules can be mixed in any order. The following diagram shows the I/O numbering convention for an example system. Both the Handheld Programmer and *Direct*SOFT 5 provide AUX functions that allow you to automatically configure the I/O. For example, with the Handheld Programmer AUX 46 executes an automatic configuration, which allows the CPU to examine the installed modules and determine the I/O configuration and addressing. With *Direct*SOFT 5, the PLC Configure I/O menu option would be used.



	Slot 1	Slot 2	Slot 3	Slot 4
Automatic	8pt. Input X100–X107	16pt. Output Y100–Y117	16pt. Input X110–X127	8pt. Input X130–X137
Manual	8pt. Input X100–X107	16pt. Output Y100–Y117	16pt. Input X200–X217	8pt. Input X120–X127

Manual I/O Configuration

It may never become necessary, but DL06 CPUs allow manual I/O address assignments for any I/O slot(s) . You can manually modify an auto configuration to match arbitrary I/O numbering. For example, two adjacent input modules can have starting addresses at X100 and X200.Use *Direct*SOFT 5 PLC Configure I/O menu option to assign manual I/O address. In automatic configuration, the addresses are assigned on 8-point boundaries. Manual configuration, however, assumes that all modules are at least 16 points, so you can only assign addresses that are a multiple of 20 (octal). You can still use 8 point modules, but 16 addresses will be assigned and the upper eight addresses will be unused.

WARNING: If you manually configure an I/O slot, the I/O addressing for the other modules may change. This is because the DL06 CPUs do not allow you to assign duplicate I/O addresses. You must always correct any I/O configuration errors before you place the CPU in RUN mode. Uncorrected errors can cause unpredictable machine operation that can result in a risk of personal injury or damage to equipment.

Power Budgeting

The DL06 has four option card slots. To determine whether the combination of cards you select will have sufficient power, you will need to perform a power budget calculation.

Power supplied

Power is supplied from two sources, the internal base unit power supply and, if required, an external supply (customer furnished). The D0-06xx (AC powered) PLCs supply a limited amount of 24VDC power. The 24VDC output can be used to power external devices.

For power budgeting, start by considering the power supplied by the base unit. All DL06 PLCs supply the same amount of 5VDC power. Only the AC units offer 24VDC auxiliary power. Be aware of the trade-off between 5VDC power and 24VDC power. The amount of 5VDC power available depends on the amount of 24VDC power being used, and the amount of 24VDC power available depends on the amount of 5VDC power consumed. Determine the amount of internally supplied power from the table on the following page.

Power required by base unit

Because of the different I/O configurations available in the DL06 family, the power consumed by the base unit itself varies from model to model. Subtract the amount of power required by the base unit from the amount of power supplied by the base unit. Be sure to subtract 5VDC and 24VDC amounts.

Power required by option cards

Next, subtract the amount of power required by the option cards you are planning to use. Again, remember to subtract both 5VDC and 24VDC. If your power budget analysis shows surplus power available, you should have a workable configuration.

DL06 Power Supplied by Base Units		
Part Number	5 VDC (mA)	24 VDC (mA)
D0-06xx	<1500mA	300mA
	<2000mA	200mA
D0-06xx-D	1500mA	none

If the 5VDC loading is less than 2000mA, but more than 1500mA, then available 24VDC supply current is 200mA. If the 5VDC loading is less than 1500mA, then the available 24VDC current is 300mA.

DL06 Base Unit Power Required		
Part Number	5 VDC (mA)	24 VDC (mA)
D0-06AA	800mA	none
D0-06AR	900mA	none
D0-06DA	800mA	none
D0-06DD1	600mA	280mA, note 1
D0-06DD2	600mA	none
D0-06DR	950mA	none
D0-06DD1-D	600mA	280mA, note 1
D0-06DD2-D	600mA	none
D0-06DR-D	950mA	none

Power Budgeting Example			
Power Source		5VDC power (mA)	24VDC power (mA)
D0-06DD1 (select row A or row B)	A	1500mA	300mA
	B	2000mA	200mA
Current Required		5VDC power (mA)	24VDC power (mA)
D0-06DD1		600mA	280mA, note 1
D0-16ND3		35mA	0
D0-10TD1		150mA	0
D0-08TR		280mA	0
F0-4AD2DA-2		100mA	0
D0-06LCD		50mA	0
Total Used		1215mA	280mA
Remaining	A	285mA	20mA
	B	785mA	note 2

NOTE: *See the DL05/DL06 OPTIONS manual for the module data for your project.*

DL06 Power Consumed by Option Cards		
Part Number	5 VDC (mA)	24 VDC (mA)
D0-07CDR	130mA	none
D0-08CDD1	100mA	none
D0-08TR	280mA	none
D0-10ND3	35mA	none
D0-10ND3F	35mA	none
D0-10TD1	150mA	none
D0-10TD2	150mA	none
D0-16ND3	35mA	none
D0-16TD1	200mA	none
D0-16TD2	200mA	none
D0-DCM	250mA	none
D0-DEVNETS	45mA	none
F0-04TRS	250mA	none
F0-08NA-1	5mA	none
F0-04AD-1	50mA	none
F0-04AD-2	75mA	none
F0-04DAH-1	25mA	150mA
F0-04DAH-2	25mA	30mA
F0-08ADH-1	25mA	25mA
F0-08ADH-2	25mA	25mA
F0-08DAH-1	25mA	220mA
F0-08DAH-2	25mA	30mA
F0-2AD2DA-2	50mA	30mA
F0-4AD2DA-1	100mA	40mA
F0-4AD2DA-2	100mA	none
F0-04RTD	70mA	none
F0-04THM	30mA	none
F0-CP128	150mA	none
H0-CTRIO(2)	250mA	none
H0-ECOM	250mA	none
H0-ECOM100	300mA	none
H0-PSCM	530mA	none

DL06 Power Consumed by Other Devices		
Part Number	5 VDC (mA)	24 VDC (mA)
D0-06LCD	50mA	none
D2-HPP	200mA	none
DV-1000	150mA	none
EA1-S3ML	210mA	none
EA1-S3MLW	210mA	none

NOTE 1: *Auxiliary 24VDC used to power V+ terminal of D0-06DD1/-D sinking outputs.*
NOTE 2: *If the PLC's auxiliary 24VDC power source is used to power the sinking outputs, use power choice A, above.*

Configuring the DL06's Comm Ports

This section describes how to configure the CPU's built-in networking ports for either MODBUS or *Direct*NET. This will allow you to connect the DL06 PLC system directly to MODBUS networks using the RTU protocol, or to other devices on a *Direct*NET network. MODBUS masters on the network must be capable of issuing the MODBUS commands to read or write the appropriate data. For details on the MODBUS protocol, please refer to the Gould MODBUS Protocol reference Guide (P1–MBUS–300 Rev. B). In the event a more recent version is available, check with your MODBUS supplier before ordering the documentation. For more details on *Direct*NET, order our *Direct*NET manual, part number DA–DNET–M.

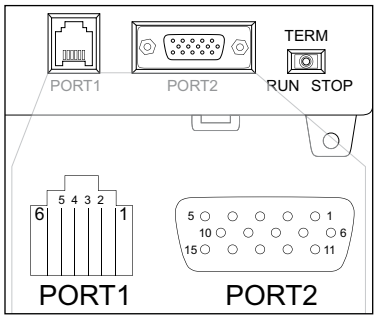
NOTE: *For information about the MODBUS protocol see the Group Schneider Web site at: www.schneiderautomation.com. At the main menu, select Support/Services, Modbus, Modbus Technical Manuals, PI-MBUS-300 Modbus Protocol Reference Guide or search for PIMBUS300. For more information about the DirectNET protocol, order our DirectNET user manual, part number DA–DNET–M, or download it free from our Web site: www.automationdirect.com. Select Documentation/Misc./DA-DNET-M.*

DL06 Port Specifications

Communications Port 1	
Port 1	Connects to HPP, *Direct*SOFT 5, operator interfaces, etc.
	6-pin, RS232C
	Communication speed (baud): 9600 (fixed)
	Parity: odd (fixed)
	Station Address: 1 (fixed)
	8 data bits
	1 start, 1 stop bit
	Asynchronous, half-duplex, DTE
	Protocol (auto-select): K-sequence (slave only), *Direct*NET (slave only), MODBUS (slave only)

Communications Port 2	
Port 2	Connects to HPP, *Direct*SOFT 5, operator interfaces, etc.
	15-pin, multifunction port, RS232C, RS422, RS485
	Communication speed (baud): 300, 600, 1200, 2400, 4800, 9600, 19200, 38400
	Parity: odd (default), even, none
	Station Address: 1 (default)
	8 data bits
	1 start, 1 stop bit
	Asynchronous, half-duplex, DTE
	Protocol (auto-select): K-sequence (slave only), *Direct*NET (master/slave), MODBUS (master/slave), non-sequence/print/ASCII in/out

DL06 Port Pinouts



Port 1 Pin Descriptions		
1	0V	Power (-) connection (GND)
2	5V	Power (+) connection
3	RXD	Receive data (RS-232C)
4	TXD	Transmit data (RS-232C)
5	5V	Power (+) connection
6	0V	Power (-) connection (GND)

Port 2 Pin Descriptions		
1	5V	Power (+) connection
2	TXD	Transmit data (RS-232C)
3	RXD	Receive data (RS-232C)
4	RTS	Ready to send (RS-232C)
5	CTS	Clear to send (RS232C)
6	RXD-	Receive data (-) (RS-422/485)
7	0V	Power (-) connection (GND)
8	0V	Power (-) connection (GND)
9	TXD+	Transmit data (+) (RS-422/485)
10	TXD-	Transmit data (-) (RS-422/485)
11	RTS+	Ready to send (+) (RS-422/485)
12	RTS-	Ready to send (-) (RS-422/485)
13	RXD+	Receive data (+) (RS-422/485)
14	CTS+	Clear to send (+) (RS-422/485)
15	CTS-	Clear to send (-) (RS-422/485)

Choosing a Network Specification

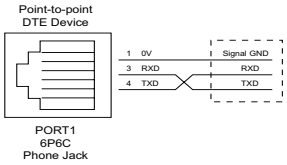
The DL06 PLC's multi-function port gives you the option of using RS-232C, RS-422, or RS-485 specifications. First, determine whether the network will be a 2-wire RS–232C type, a 4-wire RS–422 type, or a 2-wire/4-wire RS-485 type.

The RS–232C specification is simple to implement for networks of shorter distances (15 meters max) and where communication is only required between two devices. The RS–422 and RS-485 signals are for networks that cover longer distances (1000 meters max.) and for multi-drop networks (from 2 to 247 devices).

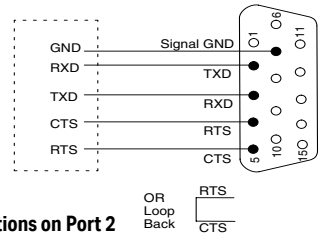
***NOTE:** Termination resistors are required at both ends of RS–422 and RS-485 networks. It is necessary to select resistors that match the impedance rating of the cable (between 100 and 500 ohms).*

RS-232 Network

Normally, the RS-232 signals are used for shorter distances (15 meters maximum), for communications between two devices.



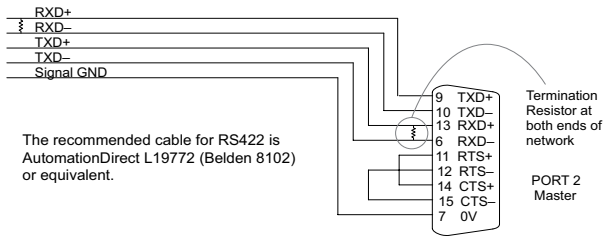
Connections on Port 1



Connections on Port 2

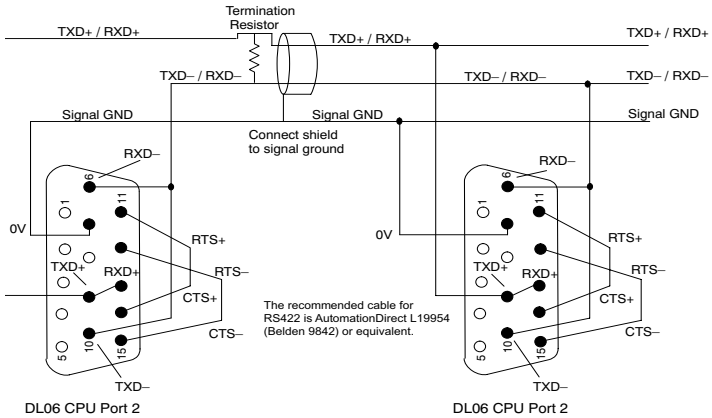
RS-422 Network

RS-422 signals are for long distances (1000 meters maximum). Use terminator resistors at both ends of RS-422 network wiring, matching the impedance rating of the cable (between 100 and 500 ohms).



RS-485 Network

RS-485 signals are for longer distances (1000 meters max) and for multi-drop networks. Use termination resistors at both ends of RS-485 network wiring, matching the impedance rating of the cable (between 100 and 500 ohms).



Connecting to MODBUS and DirectNET Networks

MODBUS Port Configuration

In *Direct*SOFT, choose the PLC menu, then Setup, then "Secondary Comm Port".

- **Port**: From the port number list box at the top, choose "Port 2".
- **Protocol**: Check the box to the left of "MODBUS" (use AUX 56 on the HPP, and select "MBUS"), and then you'll see the box below.

- **Timeout**: amount of time the port will wait after it sends a message to get a response before logging an error.
- **RTS ON / OFF Delay Time:** The RTS ON Delay Time specifies the time the DL06 waits to send the data after it has raised the RTS signal line. The RTS OFF Delay Time specifies the time the DL06 waits to release the RTS signal line after the data has been sent. *When using the DL06 on a multi-drop network, the RTS ON Delay time must be set to **5ms** or more and the RTS OFF Delay time must be set to **2ms** or more. If you encounter problems, the time can be increased.*
- **Station Number**: For making the CPU port a MODBUS master, choose "1". The possible range for MODBUS slave numbers is from 1 to 247, but the DL06 network instructions used in Master mode will access only slaves 1 to 99. Each slave must have a unique number. At powerup, the port is automatically a slave, unless and until the DL06 executes ladder logic network instructions which use the port as a master. Thereafter, the port reverts back to slave mode until ladder logic uses the port again.
- **Baud Rate**: The available baud rates include 300, 600, 1200, 2400, 4800, 9600, 19200, and 38400 baud. Choose a higher baud rate initially, reverting to lower baud rates if you experience data errors or noise problems on the network. Important: You must configure the baud rates of all devices on the network to the same value. Refer to the appropriate product manual for details.
- **Stop Bits**: Choose 1 or 2 stop bits for use in the protocol.
- **Parity**: Choose none, even, or odd parity for error checking.
- **Echo Suppression:** Select the appropriate wiring configuration used on Port 2.

Then click the button indicated to send the Port configuration to the CPU, and click Close.

DirectNET Port Configuration

In *Direct*SOFT, choose the PLC menu, then Setup, then "Secondary Comm Port".

- **Port**: From the port number list box, choose "Port 2 ".
- **Protocol**: Check the box to the left of "*Direct*NET" (use AUX 56 on the HPP, then select "DNET"), and then you'll see the dialog below.

- **Timeout**: Amount of time the port will wait after it sends a message to get a response before logging an error.
- **RTS ON / OFF Delay Time:** The RTS ON Delay Time specifies the time the DL06 waits to send the data after it has raised the RTS signal line. The RTS OFF Delay Time specifies the time the DL06 waits to release the RTS signal line after the data has been sent. *When using the DL06 on a multi-drop network, the RTS ON Delay time must be set to* ***5ms*** *or more and the RTS OFF Delay time must be set to* ***2ms*** *or more. If you encounter problems, the time can be increased.*
- **Station Number**: For making the CPU port a *Direct*NET master, choose "1". The allowable range for *Direct*NET slaves is from 1 to 90 (each slave must have a unique number). At powerup, the port is automatically a slave, unless and until the DL06 executes ladder logic instructions which attempt to use the port as a master. Thereafter, the port reverts back to slave mode until ladder logic uses the port again.
- **Baud Rate**: The available baud rates include 300, 600, 1200, 2400, 4800, 9600, 19200, and 38400 baud. Choose a higher baud rate initially, reverting to lower baud rates if you experience data errors or noise problems on the network. Important: You must configure the baud rates of all devices on the network to the same value.
- **Stop Bits**: Choose 1 or 2 stop bits for use in the protocol.
- **Parity**: Choose none, even, or odd parity for error checking.
- **Format**: Choose between hex or ASCII formats.

Then click the button indicated to send the Port configuration to the CPU, and click Close.

Non–Sequence Protocol (ASCII In/Out and PRINT)

Non-Sequence Port Configuration

Configuring port 2 on the DL06 for Non–Sequence allows the CPU to use port 2 to either read or write raw ASCII strings using the ASCII instructions. See the ASCII In/Out instructions and the PRINT instruction in chapter 5.

In *Direct*SOFT, choose the PLC menu, then Setup, then "Secondary Comm Port".

- Port: From the port number list box at the top, choose "Port 2".
- Protocol: Check the box to the left of "Non–Sequence".
- Timeout: Amount of time the port will wait after it sends a message to get a response before logging an error.

- RTS On Delay Time: The amount of time between raising the RTS line and sending the data.
- RTS Off Delay Time: The amount of time between resetting the RTS line after sending the data.
- Data Bits: Select either 7–bits or 8–bits to match the number of data bits specified for the connected devices.
- Baud Rate: The available baud rates include 300, 600, 900, 2400, 4800, 9600, 19200, and 38400 baud. Choose a higher baud rate initially, reverting to lower baud rates if you experience data errors or noise problems on the network. Important: You must configure the baud rates of all devices on the network to the same value. Refer to the appropriate product manual for details.
- Stop Bits: Choose 1 or 2 stop bits to match the number of stop bits specified for the connected devices.
- Parity: Choose none, even, or odd parity for error checking. Be sure to match the parity specified for the connected devices.
- Echo Suppression: Select the appropriate radio button based on the wiring configuration used on port 2.
- Xon/Xoff Flow Control: Choose this selection if you have Port 2 wired for Hardware Flow Control (Xon/Xoff) with RTS and CTS signal connected between all devices.
- RTS Flow Control: Choose this selection if you have Port 2 RTS signal wired between all devices.
- Memory Address: Please choose a memory address with 64 words of contiguous free memory for use by Non-Sequence Protocol.

Click the button indicated to send the port configuration to the CPU, and click Close.

Network Slave Operation

This section describes how other devices on a network can communicate with a CPU port that you have configured as a *Direct*NET slave or MODBUS slave (DL06). A MODBUS host must use the MODBUS RTU protocol to communicate with the DL06 as a slave. The host software must send a MODBUS function code and MODBUS address to specify a PLC memory location the DL06 comprehends. The *Direct*NET host uses normal I/O addresses to access applicable DL06 CPU and system. No CPU ladder logic is required to support either MODBUS slave or *Direct*NET slave operation.

NOTE: *For more intformation on DirectNET proprietary protocol, see the DirectNET reference manual, DA-DNET-M, available on our website.*

MODBUS Function Codes Supported

MODBUS Function Code	Function	DL06 Data Types Available
01	Read a group of coils	Y, CR, T, CT
02	Read a group of inputs	X, SP
05	Set / Reset a single coil	Y, CR, T, CT
15	Set / Reset a group of coils Y,	CR, T, CT
03, 04	Read a value from one or more registers	V
06	Write a value into a single register	V
16	Write a value into a group of registers	V

The MODBUS function code determines whether the access is a read or a write, and whether to access a single data point or a group of them. The DL06 supports the MODBUS function codes described below.

Determining the MODBUS Address

There are typically two ways that most host software conventions allow you to specify a PLC memory location. These are:

- By specifying the MODBUS data type and address
- By specifying a MODBUS address only

Word Data Types			
Registers	PLC Range (Octal)	Input/Holding (484 Mode)*	Input/Holding (584/984 Mode)*
V-Memory (Timers)	V0 - V377	3001 / 4001	30001 / 40001
V-Memory (Counters)	V1000 - V1177	3513 / 4513	30513 / 40513
V-Memory (Data Words)	V400 - V677	3257 / 4257	30257 / 40257
	V1200 - V7377	3641 / 4641	30641 / 40641
	V10000 - V17777	-	34097 / 44097
* Modbus: Function 04			

If Your Host Software Requires the Data Type and Address

Many host software packages allow you to specify the MODBUS data type and the MODBUS address that corresponds to the PLC memory location. This is the easiest method, but not all packages allow you to do it this way.

The actual equation used to calculate the address depends on the type of PLC data you are using. The PLC memory types are split into two categories for this purpose.

- Discrete – X, SP, Y, CR, S, T, C (contacts)
- Word – V, Timer current value, Counter current value

In either case, you basically convert the PLC octal address to decimal and add the appropriate MODBUS address (if required). The table below shows the exact equation used for each group of data.

DL06 Memory Type	QTY (Decimal)	PLC Range (Octal)	MODBUS Address Range (Decimal)	MODBUS Data Type
For Discrete Data Types .... Convert PLC Addr. to Dec. + Start of Range + Data Type				
Inputs (X)	512	X0 – X777	2048 – 2559	Input
Special Relays(SP)	512	SP0 – SP777	3072 – 3583	Input
Outputs (Y)	512	Y0 – Y777	2048 – 2559	Coil
Control Relays (CR)	1024	C0 – C1777	3072 – 4095	Coil
Timer Contacts (T)	256	T0 – T377	6144 – 6399	Coil
Counter Contacts (CT)	128	CT0 – CT177	6400 – 6527	Coil
Stage Status Bits(S)	1024	S0 – S1777	5120 – 6143	Coil
For Word Data Types .... Convert PLC Addr. to Dec. + Data Type				
Timer Current Values (V)	256	V0 – V377	0 – 255	Input Register
Counter Current Values (V)	128	V1000 – V1177	512 – 639	Input Register
V-Memory, user data (V)	3200	V1200 – V7377	640 – 3839	Holding Register
	4096	V10000 - V17777	4096 - 8191	Holding Register
V-Memory, non-volatile (V)	128	V7400 – V7577	3840 – 3967	Holding Register

The following examples show how to generate the MODBUS address and data type for hosts which require this format.

Example 1: V2100

Find the MODBUS address for User V location V2100.

1. Find V-memory in the table.
2. Convert V2100 into decimal (1088).
3. Use the MODBUS data type from the table.

Holding Reg 1088

V-memory, user data (V)	3200	V1200 – V7377	640 – 3839	Holding Register

Example 2: Y20

Find the MODBUS address for output Y20.

1. Find Y outputs in the table.
2. Convert Y20 into decimal (16).
3. Add the starting address for the range (2048).
4. Use the MODBUS data type from the table.

Coil 2064

Outputs (V)	256	Y0 – Y377	2048 - 2303	Coil

Example 3: T10 Current Value

Find the MODBUS address to obtain the current value from Timer T10.

1. Find Timer Current Values in the table.
2. Convert T10 into decimal (8).
3. Use the MODBUS data type from the table.

Input Reg. 8

Timer Current Values (V)	128	V0 – V177	0 - 127	Input Register

Example 4: C54

Find the MODBUS address for Control Relay C54.

1. Find Control Relays in the table.
2. Convert C54 into decimal (44).
3. Add the starting address for the range (3072).
4. Use the MODBUS data type from the table.

Coil 3116

Control Relays (CR)	512	C0 – C77	3072 – 3583	Coil

If Your MODBUS Host Software Requires an Address ONLY

Some host software does not allow you to specify the MODBUS data type and address. Instead, you specify an address only. This method requires another step to determine the address, but it's still fairly simple. Basically, MODBUS also separates the data types by address ranges as well. So this means an address alone can actually describe the type of data and location. This is often referred to as "adding the offset". One important thing to remember here is that two different addressing modes may be available in your host software package. These are:

- 484 Mode
- 584/984 Mode

We recommend that you use the 584/984 addressing mode if your host software allows you to choose. This is because the 584/984 mode allows access to a higher number of memory locations within each data type. If your software only supports 484 mode, then there may be some PLC memory locations that will be unavailable. The actual equation used to calculate the address depends on the type of PLC data you are using. The PLC memory types are split into two categories for this purpose.

- Discrete – X, SP, Y, CR, S, T (contacts), C (contacts)
- Word – V, Timer current value, Counter current value

In either case, you basically convert the PLC octal address to decimal and add the appropriate MODBUS addresses (as required). The table below shows the exact equation used for each group of data.

Discrete Data Types				
DL06 Memory Type	**PLC Range (Octal)**	**Address (484 Mode)**	**Address (584/984 Mode)**	**MODBUS Data Type**
Global Inputs (GX)	GX0-GX1746	1001 - 1999	10001 - 10999	Input
	GX1747-GX3777	–	11000 - 12048	–
Inputs (X)	X0 – X777	–	12049 - 12304	Input
Special Relays (SP)	SP0 – SP777	–	13073 - 13584	Input
Global Outputs (GY)	GY0 - GX3777	1 - 2048	1 - 2048	–
Outputs (Y)	Y0 – Y777	2049 - 2560	2049 - 2560	Output
Control Relays (CR)	C0 – C1777	3073 - 4096	3073 - 3584	Output
Timer Contacts (T)	T0 – T377	6145 - 6400	6145 - 6400	Output
Counter Contacts (CT)	CT0 – CT177	6401 - 6656	6401 - 6656	Output
Stage Status Bits (S)	S0 – S1777	5121 - 6144	5121 - 6144	Output

Word Data Types			
Registers	**PLC Range (Octal)**	**Input/Holding (484 Mode)***	**Input/Holding (584/984 Mode)***
V-memory (Timers)	V0 - V377	3001/4001	30001/40001
V-memory (Counters)	V1000 - V1177	3513/4513	30513/40513
V-memory (Data Words)	V1200 - V1377	3641/4641	30641/40641
	V1400 - V1746	3769/4769	30769/40769
	V1747 - V1777	---	31000/41000
	V2000 - V7377	---	41025
	V10000 - V17777	---	44097

***MODBUS: Function 04**

1. Refer to your PLC user manual for the correct memory mapping size of your PLC. Some of the addresses shown above might not pertain to your particular CPU.

2. For an automated MODBUS/Koyo address conversion utility, go to our website w**ww.automationdirect.com**, and down load download the EXCEL file **modbus_conversion.xls** located at: Tech Support > Technical Support Home page.

Example 1: V2100 584/984 Mode

Find the MODBUS address for User V location V2100. PLC Address (Dec.) + Mode Address

1. Find V-memory in the table. V2100 = 1088 decimal
2. Convert V2100 into decimal (1088). 1088 + 40001 = **41089**
3. Add the MODBUS starting address for the mode (40001).

For Word Data Types....	PLC Address (Dec.)	+	Appropriate Mode Address			
Timer Current Values (V)	128	V0 – V177	0 – 127	3001	30001	Input Register
Counter Current Values (V)	128	V1200 – V7377	640 – 3839	3001	30001	Input Register
V-memory, user data (V)	1024	V2000 – V3777	1024 – 2047	4001	40001	Holding Register

Example 2: Y20 584/984 Mode

Find the MODBUS address for output Y20. PLC Addr. (Dec.) + Start Address + Mode

1. Find Y outputs in the table. Y20 = 16 decimal
2. Convert Y20 into decimal (16). 16 + 2048 + 1 = **2065**
3. Add the starting address for the range (2048).
4. Add the MODBUS address for the mode (1).

Outputs (Y)	320	Y0 - Y477	2048 – 2367	1	1	Coil
Control Relays (CR)	256	C0 - C377	3072 – 3551	1	1	Coil
Timer Contacts (T)	128	T0 - T177	6144 – 6271	1	1	Coil

Example 3: T10 Current Value 484 Mode

Find the MODBUS address to obtain the Current value from Timer T10.

PLC Address (Dec.) + Mode Address

TA10 = 8 decimal

1. Find Timer Current Values in the table.
2. Convert T10 into decimal (8).
3. Add the MODBUS starting address for the mode (3001).

8 + 3001 = **3009**

For Word Data Types....	PLC Address (Dec.)	+	Appropriate Mode Address			
Timer Current Values (V)	128	V0 – V177	0 – 127	3001	30001	Input Register
Counter Current Values (V)	128	V1200 – V7377	512 – 639	3001	30001	Input Register
V-memory, user data (V)	1024	V2000 – V3777	1024 – 2047	4001	40001	Holding Register

Example 4: C54 584/984 Mode

Find the MODBUS address for Control Relay C54.

PLC Addr. (Dec.) + Start Address + Mode

1. Find Control Relays in the table.
2. Convert C54 into decimal (44).
3. Add the starting address for the range (3072).
4. Add the MODBUS address for the mode (1).

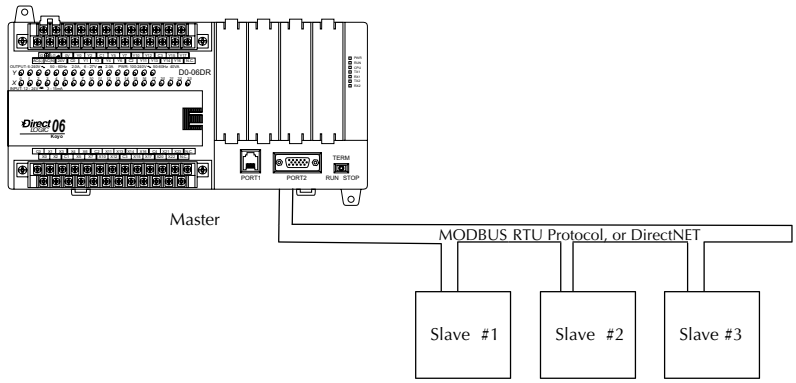
C54 = 44 decimal

44 + 3072 + 1 = **3117**

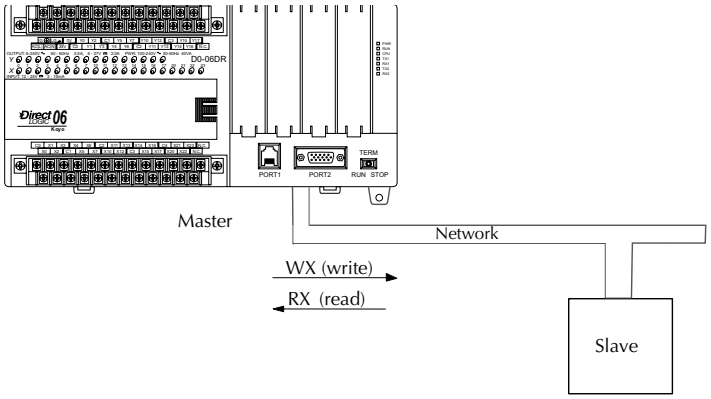
Outputs (Y)	320	Y0 – Y477	2048 – 2367	1	1	Coil
Control Relays (CR)	256	C0 – C377	3072 – 3551	1	1	Coil
Timer Contacts (T)	128	T0 – T177	6144 – 6271	1	1	Coil

Network Master Operation

This section describes how the DL06 can communicate on a MODBUS or *Direct*NET network as a master. For MODBUS networks, it uses the MODBUS RTU protocol, which must be interpreted by all the slaves on the network. Both MODBUS and *Direct*Net are single master/multiple slave networks. The master is the only member of the network that can initiate requests on the network. This section teaches you how to design the required ladder logic for network master operation.



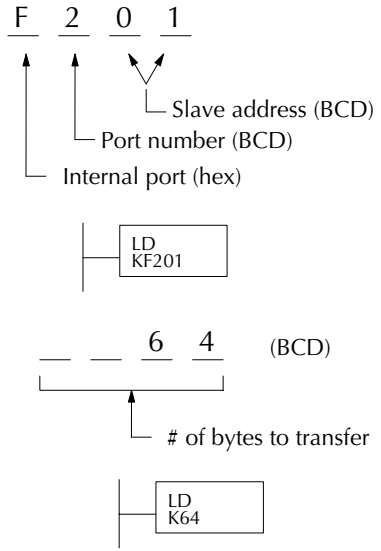
When using the DL06 PLC as the master station, simple RLL instructions are used to initiate the requests. The WX instruction initiates network write operations, and the RX instruction initiates network read operations. Before executing either the WX or RX commands, we will need to load data related to the read or write operation onto the CPU's accumulator stack. When the WX or RX instruction executes, it uses the information on the stack combined with data in the instruction box to completely define the task, which goes to the port.



The following step-by-step procedure will provide you the information necessary to set up your ladder program to receive data from a network slave.

Step 1: Identify Master Port # and Slave

The first Load (LD) instruction identifies the communications port number on the network master (DL06) and the address of the slave station. This instruction can address up to 99 MODBUS slaves, or 90 *Direct*NET slaves. The format of the word is shown to the right. The "F2" in the upper byte indicates the use of the right port of the DL06 PLC, port number 2. The lower byte contains the slave address number in BCD (01 to 99).



Step 2: Load Number of Bytes to Transfer

The second Load (LD) instruction determines the number of bytes which will be transferred between the master and slave in the subsequent WX or RX instruction. The value to be loaded is in BCD format (decimal), from 1 to 128 bytes.

The number of bytes specified also depends on the type of data you want to obtain. For example, the DL06 Input points can be accessed by V-memory locations or as X input locations. However, if you only want X0 – X27, you'll have to use the X input data type because the V-memory locations can only be accessed in 2-byte increments. The following table shows the byte ranges for the various types of *Direct*LOGIC products.

DL05 / 06 / 205 / 350 / 405 Memory	Bits per unit	Bytes
V-memory T / C current value	16 16	2 2
Inputs (X, SP)	8	1
Outputs (Y, C, Stage, T/C bits)	8	1
Scratch Pad Memory	8	1
Diagnostic Status	8	1

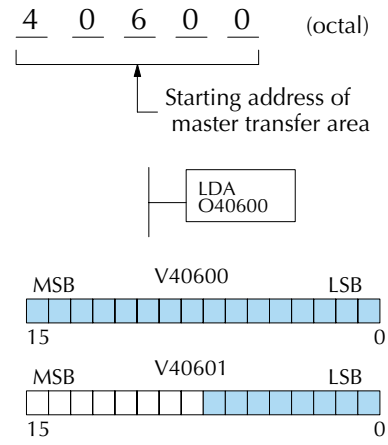
DL330 / 340 Memory	Bits per unit	Bytes
Data registers T / C accumulator	8 16	1 2
I/O, internal relays, shift register bits, T/C bits, stage bits	1	1
Scratch Pad Memory	8	1
Diagnostic Status(5 word R/W)	16	10

Step 3: Specify Master Memory Area

The third instruction in the RX or WX sequence is a Load Address (LDA) instruction. Its purpose is to load the starting address of the memory area to be transferred. Entered as an octal number, the LDA instruction converts it to hex and places the result in the accumulator.

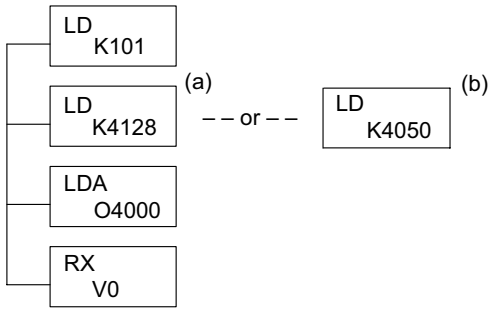
For a WX instruction, the DL06 CPU sends the number of bytes previously specified from its memory area beginning at the LDA address specified.

For an RX instruction, the DL06 CPU reads the number of bytes previously specified from the slave, placing the received data into its memory area beginning at the LDA address specified.



NOTE: *Since V-memory words are always 16 bits, you may not always use the whole word. For example, if you only specify 3 bytes and you are reading Y outputs from the slave, you will only get 24 bits of data. In this case, only the 8 least significant bits of the last word location will be modified. The remaining 8 bits are not affected.*

When using MODBUS, the RX instructions use function 3 by default, to read MODBUS holding registers (Address 40001). The DL05/DL06, DL250-1/260, DL350, DL454 support function 04, read input register (Address 30001). To use function 04, put the number "4" into the most significant position (4xxx) of the total number of bytes. Four digits must be entered for the instruction to work properly with this mode.

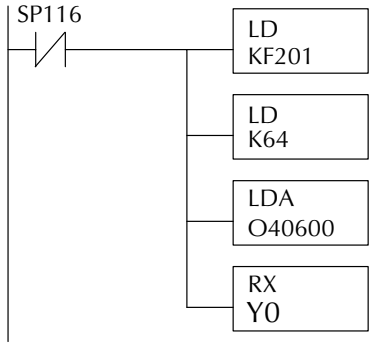


The (a) K4128 indicates the instruction will read 128 bytes of MODBUS input registers (30001). The (b) K4050 indicates the instruction will read 50 bytes of MODBUS input registers (30001). The value of 4 in the most significant position will cause the RX to use MODBUS function 4 (30001 range).

Step 4: Specify Slave Memory Area

The last instruction in our sequence is the WX or RX instruction itself. Use WX to write to the slave, and RX to read from the slave. All four of our instructions are shown to the right. In the last instruction, you must specify the starting address and a valid data type for the slave.

- DirectNET slaves – specify the same address in the WX and RX instruction as the slave's native I/O address
- MODBUS DL405, DL205, or DL06 slaves – specify the same address in the WX and RX instruction as the slave's native I/O address
- MODBUS 305 slaves – use the following table to convert DL305 addresses to MODBUS addresses



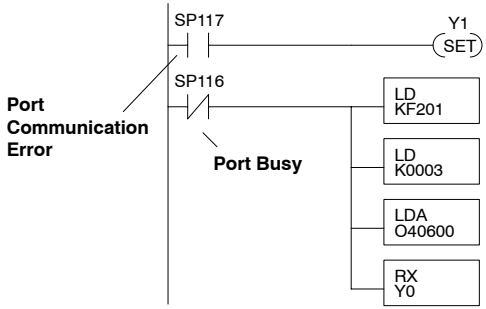
DL305 Series CPU Memory Type–to–MODBUS Cross Reference (excluding 350 CPU)					
PLC Memory Type	PLC Base Address	MODBUS Base Address	PLC Memory Type	PLC Base Address	MODBUS Base Address
TMR/CNT Current Values	R600	V0	TMR/CNT Status Bits	CT600	GY600
I/O Points	IO 000	GY0	Control Relays	CR160	GY160
Data Registers	R401,R400	V100	Shift Registers	SR400	GY400
Stage Status Bits (D3-330P only)	S0	GY200			

Communications from a Ladder Program

Typically network communications will last longer than 1 scan. The program must wait for the communications to finish before starting the next transaction.

Port 2, which can be a master, has two Special Relay contacts associated with it (see Appendix D for comm port special relays).One indicates "Port busy"(SP116), and the other indicates "Port Communication Error"(SP117). The example above shows the use of these contacts for a network master that only reads a device (RX). The "Port Busy" bit is on while the PLC communicates with the slave. When the bit is off the program can initiate the next network request.

The "Port Communication Error" bit turns on when the PLC has detected an error. Use of this bit is optional. When used, it should be ahead of any network instruction boxes since the error bit is reset when an RX or WX instruction is executed

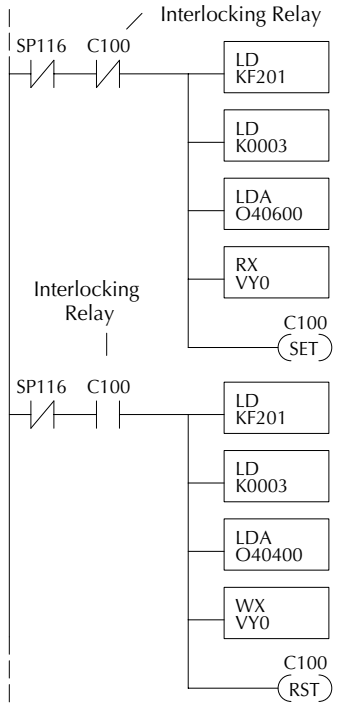


Multiple Read and Write Interlocks

If you are using multiple reads and writes in the RLL program, you have to interlock the routines to make sure all the routines are executed. If you don't use the interlocks, then the CPU will only execute the first routine. This is because each port can only handle one transaction at a time.

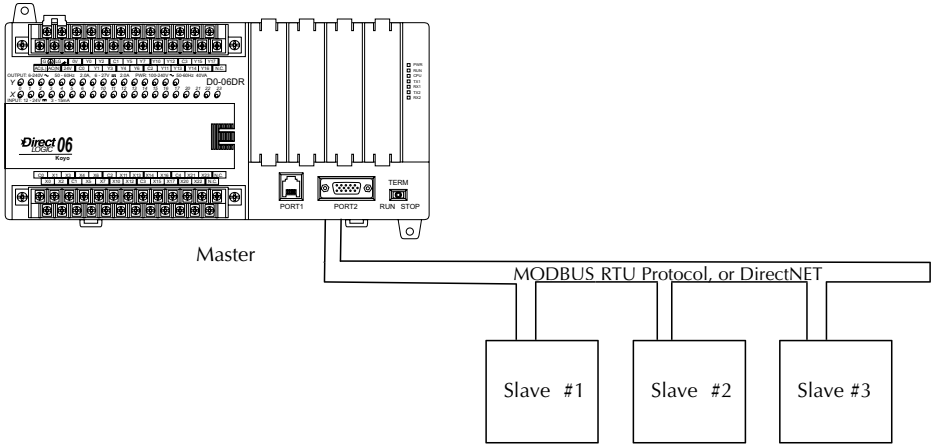
In the example to the right, after the RX instruction is executed, C100 is set. When the port has finished the communication task, the second routine is executed and C100 is reset.

If you're using RLL^{PLUS} Stage Programming, you can put each routine in a separate program stage to ensure proper execution and switch from stage to stage allowing only one of them to be active at a time.



Network Master Operation (using MRX and MWX Instructions)

This section describes how the DL06 can communicate on a MODBUS RTU network as a master using the MRX and MWX read/write instructions. These instructions allow you to enter native MODBUS addressing in your ladder logic program with no need to perform octal to decimal conversions. MODBUS is a single master/multiple slave network. The master is the only member of the network that can initiate requests on the network. This section teaches you how to design the required ladder logic for network master operation.



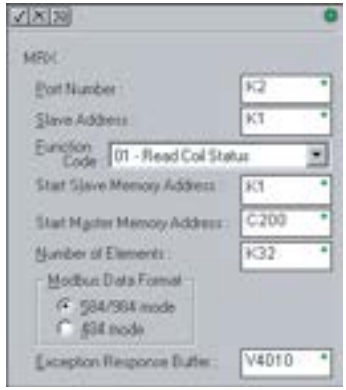
MODBUS Function Codes Supported

The MODBUS function code determines whether the access is a read or a write, and whether to access a single data point or a group of them. The DL06 supports the MODBUS function codes described below.

MODBUS Function Code	Function	DL06 Data Types Available
01	Read a group of coils	Y, CR, T, CT
02	Read a group of inputs	X, SP
05	Set / Reset a single coil (slave only)	Y, CR, T, CT
15	Set / Reset a group of coils	Y, CR, T, CT
03, 04	Read a value from one or more registers	V
06	Write a value into a single register (slave only)	V
07	Read Exception Status	V
08	Diagnostics	V
16	Write a value into a group of registers	V

MODBUS Read from Network(MRX)

The MODBUS Read from Network (MRX) instruction is used by the DL06 network master to read a block of data from a connected slave device and to write the data into V–memory addresses within the master. The instruction allows the user the to specify the MODBUS Function Code, slave station address, starting master and slave memory addresses, number of elements to transfer, MODBUS data format and the Exception Response Buffer.



- **Port Number:** must be DL06 Port 2 (K2)
- **Slave Address:** specify a slave station address (0–247)
- **Function Code**: The following MODBUS function codes are supported by the MRX instruction:
  - 01 – Read a group of coils
  - 02 – Read a group of inputs
  - 03 – Read holding registers
  - 04 – Read input registers
  - 07 – Read Exception status
  - 08 – Diagnostics
- **Start Slave Memory Address**: specifies the starting slave memory address of the data to be read. See the table on the following page.
- **Start Master Memory Address**: specifies the starting memory address in the master where the data will be placed. See the table on the following page.
- **Number of Elements**: specifies how many coils, input, holding registers or input register will be read. See the table on the following page.
- **MODBUS Data Format**: specifies MODBUS 584/984 or 484 data format to be used
- **Exception Response Buffer**: specifies the master memory address where the Exception Response will be placed. See the table on the following page.

MRX Slave Memory Address

MRX Slave Address Ranges		
Function Code	**MODBUS Data Format**	**Slave Address Range(s)**
01 – Read Coil	484 Mode	1–999
01 – Read Coil	584/984 Mode	1–65535
02 – Read Input Status	484 Mode	1001–1999
02 – Read Input Status	584/984 Mode	10001–19999 (5 digit) or 100001–165535 (6 digit)
03 – Read Holding Register	484 Mode	4001–4999
03 – Read Holding Register	584/984	40001–49999 (5 digit) or 4000001–465535 (6 digit)
04 – Read Input Register	484 Mode	3001–3999
04 – Read Input Register	584/984 Mode	30001–39999 (5 digit) or 3000001–365535 (6 digit)
07 – Read Exception Status	484 and 584/984 Mode	n/a
08 – Diagnostics	484 and 584/984 Mode	0–65535

MRX Master Memory Addresses

MRX Master Memory Address Ranges	
Operand Data Type	**DL06 Range**
Inputs X	0–1777
Outputs Y	0–1777
Control Relays C	0–3777
Stage Bits S	0–1777
Timer Bits T	0–377
Counter Bits CT	0–377
Special Relays SP	0–777
V–memory V	All
Global Inputs GX	0–3777
Global Outputs GY	0–3777

MRX Number of Elements

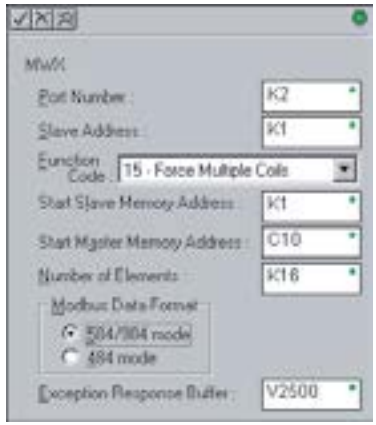
MRX Number of Elements		
Operand Data Type		**DL06 Range**
V–memory	V	All
Constant	K	1–2000

MRX Exception Response Buffer

MRX Exception Response Buffer		
Operand Data Type		**DL06 Range**
V–memory	V	All

MODBUS Write to Network (MWX)

The MODBUS Write to Network (MWX) instruction is used to write a block of data from the network masters's (DL06) memory to MODBUS memory addresses within a slave device on the network. The instruction allows the user to specify the MODBUS Function Code, slave station address, starting master and slave memory addresses, number of elements to transfer, MODBUS data format and the Exception Response Buffer.



- **Port Number**: must be DL06 Port 2 (K2)
- **Slave Address**: specify a slave station address (0–247)
- **Function Code**: The following MODBUS function codes are supported by the MWX instruction:

  05 – Force Single coil

  06 – Preset Single Register

  08 – Diagnostics

  15 – Force Multiple Coils

  16 – Preset Multiple Registers
- **Start Slave Memory Address**: specifies the starting slave memory address where the data will be written.
- **Start Master Memory Address:** specifies the starting address of the data in the master that is to written to the slave.
- **Number of Elements**: specifies how many consecutive coils or registers will be written to. This field is only active when either function code 15 or 16 is selected.
- **MODBUS Data Format**: specifies MODBUS 584/984 or 484 data format to be used.
- **Exception Response Buffer**: specifies the master memory address where the Exception Response will be placed.

MWX Slave Memory Address

MWX Slave Address Ranges		
Function Code	**MODBUS Data Format**	**Slave Address Range(s)**
05 – Force Single Coil	484 Mode	1–999
05 – Force Single Coil	584/984 Mode	1–65535
06 – Preset Single Register	484 Mode	4001–4999
06 – Preset Single Register	84/984 Mode	40001–49999 (5 digit) or 400001–465535 (6 digit)
08 – Diagnostics	484 and 584/984 Mode	0–65535
15 – Force Multiple Coils	484	1–999
15 – Force Multiple Coils	585/984 Mode	1–65535
16 – Preset Multiple Registers	484 Mode	4001–4999
16 – Preset Multiple Registers	584/984 Mode	40001–49999 (5 digit) or 4000001–465535 (6 digit)

MWX Master Memory Addresses

MWX Master Memory Address Ranges		
Operand Data Type		**DL06 Range**
Inputs	X	0–777
Outputs	Y	0–777
Control Relays	C	0–1777
Stage Bits	S	0–1777
Timer Bits	T	0–377
Counter Bits	CT	0–177
Special Relays	SP	0–777
V–memory	V	All
Global Inputs	GX	0–3777
Global Outputs	GY	0–3777

MWX Number of Elements

MWX Number of Elements		
Operand Data Type		**DL06 Range**
V–memory	V	All
Constant	K	1–2000

MWX Exception Response Buffer

MWX Exception Response Buffer		
Operand Data Type		DL06 Range
V–memory	V	All

MRX/MWX Example in DirectSOFT

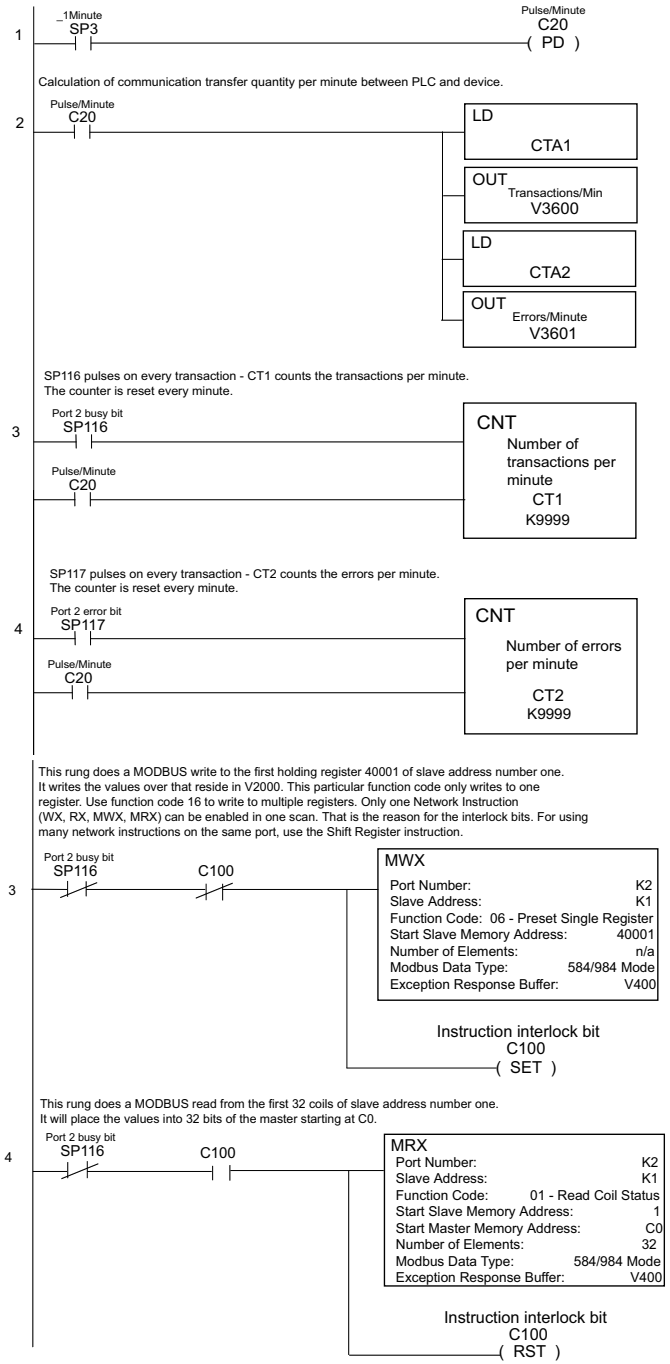
DL06 port 2 has two Special Relay contacts associated with it (see Appendix D for comm port special relays). One indicates "Port busy"(SP116), and the other indicates "Port Communication Error"(SP117). The "Port Busy" bit is on while the PLC communicates with the slave. When the bit is off the program can initiate the next network request. The "Port Communication Error" bit turns on when the PLC has detected an error and use of this bit is optional. When used, it should be ahead of any network instruction boxes since the error bit is reset when an MRX or MWX instruction is executed. Typically network communications will last longer than 1 CPU scan. The program must wait for the communications to finish before starting the next transaction.

The "Port Communication Error" bit turns on when the PLC has detected an error. Use of this bit is optional. When used, it should be ahead of any network instruction boxes since the error bit is reset when an RX or WX instruction is executed.

Multiple Read and Write Interlocks

If you are using multiple reads and writes in the RLL program, you have to interlock the routines to make sure all the routines are executed. If you don't use the interlocks, then the CPU will only execute the first routine. This is because each port can only handle one transaction at a time. In the example below, after the MRX instruction is executed, C100 is set. When the port has finished the communication task, the second routine is executed and C100 is reset. If you're using RLL^{PLUS} Stage Programming, you can put each routine in a separate program stage to ensure proper execution and switch from stage to stage allowing only one of them to be active at a time.

See example on the next page.


1
_1Minute
SP3
Pulse/Minute
C20
(PD)
Calculation of communication transfer quantity per minute between PLC and device.
2
Pulse/Minute
C20
LD
CTA1
OUT
Transactions/Min
V3600
LD
CTA2
OUT
Errors/Minute
V3601
SP116 pulses on every transaction - CT1 counts the transactions per minute.
The counter is reset every minute.
3
Port 2 busy bit
SP116
Pulse/Minute
C20
CNT
Number of transactions per minute
CT1
K9999
SP117 pulses on every transaction - CT2 counts the errors per minute.
The counter is reset every minute.
4
Port 2 error bit
SP117
Pulse/Minute
C20
CNT
Number of errors per minute
CT2
K9999
This rung does a MODBUS write to the first holding register 40001 of slave address number one.
It writes the values over that reside in V2000. This particular function code only writes to one
register. Use function code 16 to write to multiple registers. Only one Network Instruction
(WX, RX, MWX, MRX) can be enabled in one scan. That is the reason for the interlock bits. For using
many network instructions on the same port, use the Shift Register instruction.
3
Port 2 busy bit
SP116
C100
MWX
Port Number: K2
Slave Address: K1
Function Code: 06 - Preset Single Register
Start Slave Memory Address: 40001
Number of Elements: n/a
Modbus Data Type: 584/984 Mode
Exception Response Buffer: V400
Instruction interlock bit
C100
(SET)
This rung does a MODBUS read from the first 32 coils of slave address number one.
It will place the values into 32 bits of the master starting at C0.
4
Port 2 busy bit
SP116
C100
MRX
Port Number: K2
Slave Address: K1
Function Code: 01 - Read Coil Status
Start Slave Memory Address: 1
Start Master Memory Address: C0
Number of Elements: 32
Modbus Data Type: 584/984 Mode
Exception Response Buffer: V400
Instruction interlock bit
C100
(RST)

STANDARD RLL INSTRUCTIONS

In This Chapter...

Introduction

DL06 Micro PLCs offer a wide variety of instructions to perform many different types of operations. This chapter shows you how to use each standard Relay Ladder Logic (RLL) instruction. In addition to these instructions, you may also need to refer to the Drum instruction in Chapter 6, the Stage programming instructions in Chapter 7, PID in Chapter 8, LCD in Chapter 10 and programming for analog modules in D0-OPTIONS-M.

There are two ways to quickly find the instruction you need.

- If you know the instruction category (Boolean, Comparative Boolean, etc.), just use the title at the top of the page to find the pages that discuss the instructions in that category.
- If you know the individual instruction name, use the following table to find the page(s) that discusses the instruction.

Using Boolean Instructions

Do you ever wonder why so many PLC manufacturers always quote the scan time for a 1K Boolean program? Simple. Most programs utilize many Boolean instructions. These are typically very simple instructions designed to join input and output contacts in various series and parallel combinations. Our ***Direct***SOFT software is a similar program. It uses graphic symbols to develop a program; therefore, you don't necessarily have to know the instruction mnemonics in order to develop your program.

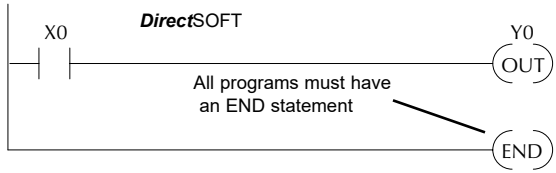
Many of the instructions in this chapter are not program instructions used in ***Direct***SOFT, but are implied. In other words, they are not actually keyboard commands, however, they can be seen in a Mnemonic View of the program once the ***Direct***SOFT program has been developed and accepted (compiled). Each instruction listed in this chapter will have a small chart to indicate how the instruction is used with ***Direct***SOFT and the HPP.

DS	Implied
HPP	Used

The following paragraphs show how these instructions are used to build simple ladder programs.

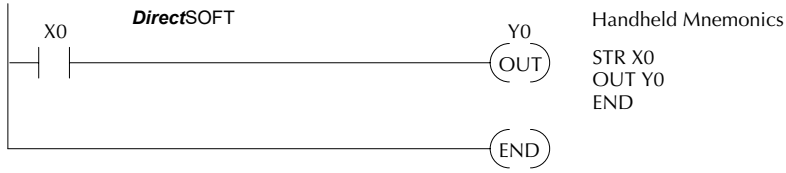
END Statement

All DL06 programs require an END statement as the last instruction. This tells the CPU that this is the end of the program. Normally, any instructions placed after the END statement will not be executed. There are exceptions to this, such as interrupt routines, etc. This chapter will discuss the instruction set in detail.



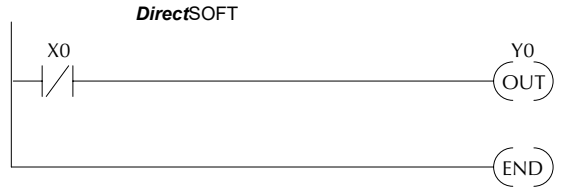
Simple Rungs

You use a contact to start rungs that contain both contacts and coils. The boolean instruction that does this is called a Store or, STR instruction. The output point is represented by the Output or, OUT instruction. The following example shows how to enter a single contact and a single output coil.



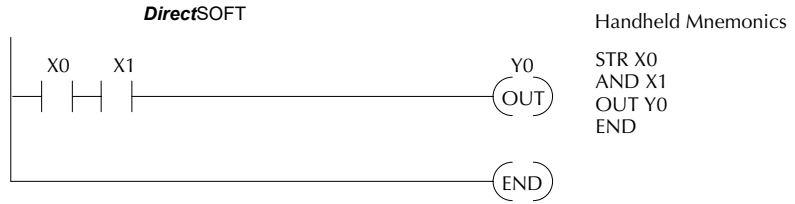
Normally Closed Contact

Normally closed contacts are also very common. This is accomplished with the Store Not, or STRN instruction. The following example shows a simple rung with a normally closed contact.



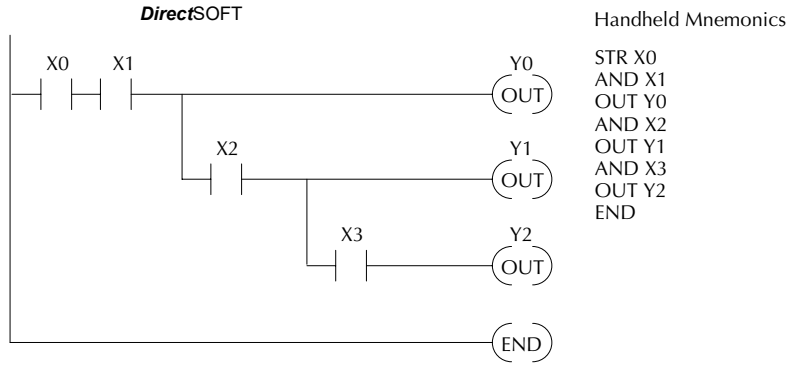
Contacts in Series

Use the AND instruction to join two or more contacts in series. The following example shows two contacts in series and a single output coil. The instructions used would be STR X0, AND X1, followed by OUT Y0.



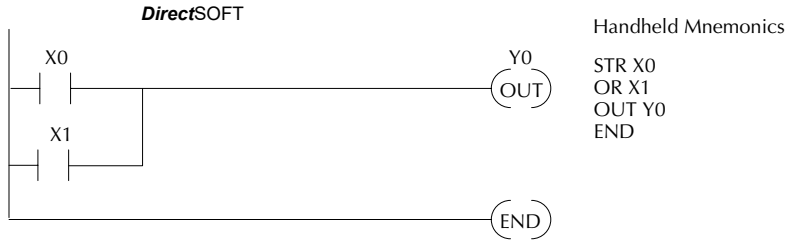
Midline Outputs

Sometimes, it is necessary to use midline outputs to get additional outputs that are conditional on other contacts. The following example shows how you can use the AND instruction to continue a rung with more conditional outputs.



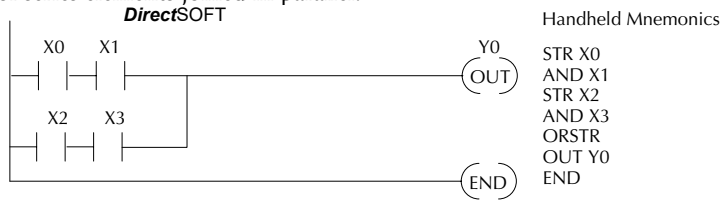
Parallel Elements

You may also have to join contacts in parallel. The OR instruction allows you to do this. The following example shows two contacts in parallel and a single output coil. The instructions would be STR X0, OR X1, followed by OUT Y0.



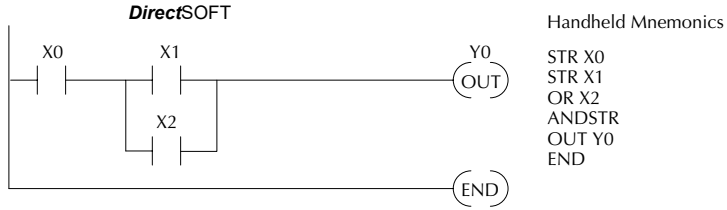
Joining Series Branches in Parallel

Quite often, it is necessary to join several groups of series elements in parallel. The Or Store (ORSTR) instruction allows this operation. The following example shows a simple network consisting of series elements joined in parallel.



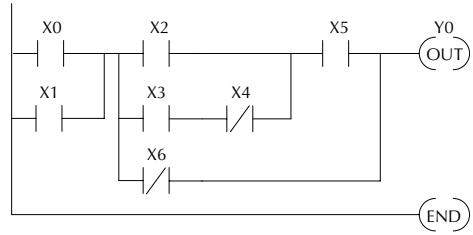
Joining Parallel Branches in Series

You can also join one or more parallel branches in series. The And Store (ANDSTR) instruction allows this operation. The following example shows a simple network with contact branches in series with parallel contacts.



Combination Networks

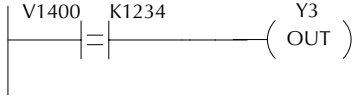
You can combine the various types of series and parallel branches to solve almost any application problem. The example at right shows a simple combination network.



Comparative Boolean

Some PLC manufacturers make it really difficult to do a simple comparison of two numbers. Some of them require you to move the data all over the place before you can actually perform the comparison. The DL06 Micro PLCs provide Comparative Boolean instructions that allow you to quickly and easily solve this problem. The Comparative Boolean provides evaluation of two BCD values using boolean contacts. The valid evaluations are: equal to, not equal to, equal to or greater than, and less than.

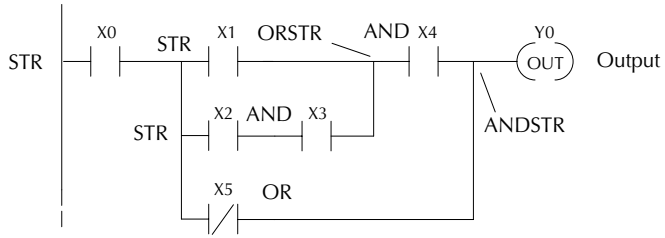
In the example, when the BCD value in V-memory location V1400 is equal to the constant value 1234, Y3 will energize.



Boolean Stack

There are limits to how many elements you can include in a rung. This is because the DL06 PLCs use an 8-level boolean stack to evaluate the various logic elements. The boolean stack is a temporary storage area that solves the logic for the rung. Each time the program encounters a STR instruction, the instruction is placed on the top of the stack. Any other STR instructions already on the boolean stack are pushed down a level. The ANDSTR, and ORSTR instructions combine levels of the boolean stack when they are encountered. An error will occur during program compilation if the CPU encounters a rung that uses more than the eight levels of the boolean stack.

The following example shows how the boolean stack is used to solve boolean logic.



STR X0

1	STR X0
2	
3	
4	

STR X1

1	STR X1
2	STR X0
3	
4	

STR X2

1	STR X2
2	STR X1
3	STR X0
4	

AND X3

1	X2 AND X3
2	STR X1
3	STR X0
4	

ORSTR

1	X1 or (X2 AND X3)
2	STR X0
3	

AND X4

1	X4 AND {X1 or (X2 AND X3)}
2	STR X0
3	

ORNOT X5

1	NOT X5 OR X4 AND {X1 OR (X2 AND X3)}
2	STR X0
3	

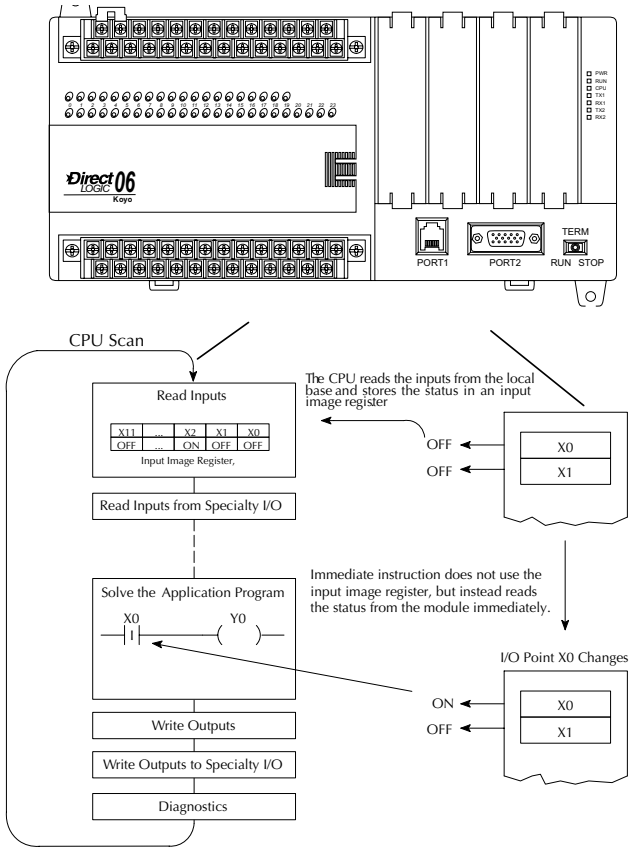
ANDSTR

1	XO AND (NOT X5 or X4) AND {X1 or (X2 AND X3)}
2	
3	

Immediate Boolean

The DL06 Micro PLCs can usually complete an operation cycle in a matter of milliseconds. However, in some applications you may not be able to wait a few milliseconds until the next I/O update occurs. The DL06 PLCs offer Immediate input and outputs which are special boolean instructions that allow reading directly from inputs and writing directly to outputs during the program execution portion of the CPU cycle. You may recall that this is normally done during the input or output update portion of the CPU cycle. The immediate instructions take longer to execute because the program execution is interrupted while the CPU reads or writes the I/O point. This function is not normally done until the read inputs or the write outputs portion of the CPU cycle.

***NOTE**: Even though the immediate input instruction reads the most current status from the input point, it only uses the results to solve that one instruction. It does not use the new status to update the image register. Therefore, any regular instructions that follow will still use the image register values. Any immediate instructions that follow will access the I/O again to update the status. The immediate output instruction will write the status to the I/O and update the image register.*

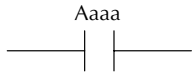


Boolean Instructions

Store (STR)

DS	Used
HPP	Used

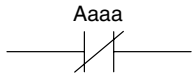
The Store instruction begins a new rung or an additional branch in a rung with a normally open contact. Status of the contact will be the same state as the associated image register point or memory location.



Store Not (STRN)

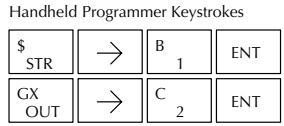
DS	Used
HPP	Used

The Store Not instruction begins a new rung or an additional branch in a rung with a normally closed contact. Status of the contact will be opposite the state of the associated image register point or memory location.

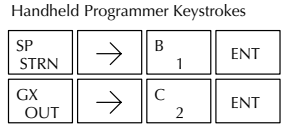


Operand Data Type		DL06 Range
	A	aaa
Inputs	X	0–777
Outputs	Y	0–777
Control Relays	C	0–1777
Stage	S	0–1777
Timer	T	0–377
Counter C	CT	0–177
Special Relay	SP	0–777

In the following Store example, when input X1 is on, output Y2 will energize.

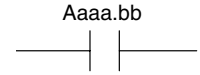


In the following Store Not example, when input X1 is off output Y2 will energize.



Store Bit-of-Word (STRB)

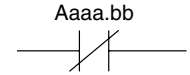
DS	Used
HPP	Used



The Store Bit-of-Word instruction begins a new rung or an additional branch in a rung with a normally open contact. Status of the contact will be the same state as the bit referenced in the associated memory location.

Store Not Bit-of-Word (STRNB)

DS	Used
HPP	Used



The Store Not instruction begins a new rung or an additional branch in a rung with a normally closed contact. Status of the contact will be opposite the state of the bit referenced in the associated memory location.

Operand Data Type		DL06 Range	
	A	aaa	bb
V-memory	B	See memory map	0 to 15
Pointer	PB	See memory map	0 to 15

These instructions look like the STR and STRN instructions only the address is different. Take note how the address is set up in the following Store Bit-of-Word example.

When bit 12 of V-memory location V1400 is on, output Y2 will energize.

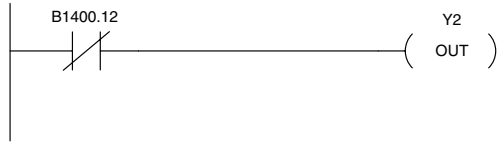
DirectSOFT

Handheld Programmer Keystrokes

STR	SHFT	B	→	V	1	4	0	0
→	K	1	2	ENT				
OUT	→	2	ENT					

In the following Store Not Bit-of-Word example, when bit 12 of V-memory location V1400 is off, output Y2 will energize.

DirectSOFT

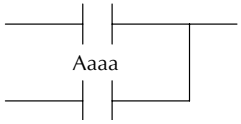


Handheld Programmer Keystrokes

STRN	SHFT	B	→	V	1	4	0	0
→	K	1	2	ENT				
OUT	→	2	ENT					

Or (OR)

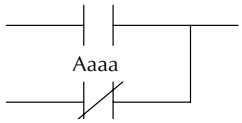
DS	Implied
HPP	Used



The Or instruction will logically OR a normally open contact in parallel with another contact in a rung. The status of the contact will be the same state as the associated image register point or memory location.

Or Not (ORN)

DS	Implied
HPP	Used



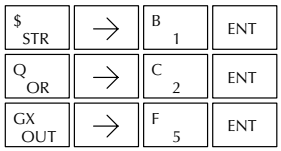
The Or Not instruction will logically OR a normally closed contact in parallel with another contact in a rung. The status of the contact will be opposite the state of the associated image register point or memory location.

Operand Data Type		DL06 Range
	A	aaa
Inputs	X	0-777
Outputs	Y	0-777
Control Relays	C	0–1777
Stage	S	0–1777
Timer	T	0–377
Counter	CT	0–177
Special Relay	SP	0-777

In the following Or example, when input X1 or X2 is on, output Y5 will energize.

DirectSOFT

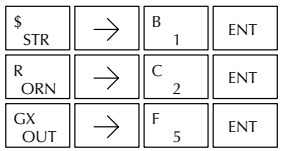
Handheld Programmer Keystrokes



In the following Or Not example, when input X1 is on or X2 is off, output Y5 will energize.

DirectSOFT

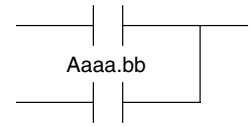
Handheld Programmer Keystrokes



Or Bit-of-Word (OR)

DS	Implied
HPP	Used

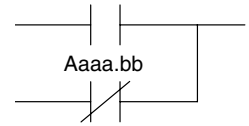
The Or Bit-of-Word instruction will logically OR a normally open contact in parallel with another contact in a rung. Status of the contact will be the same state as the bit referenced in the associated memory location.



Or Not Bit-of-Word (ORN)

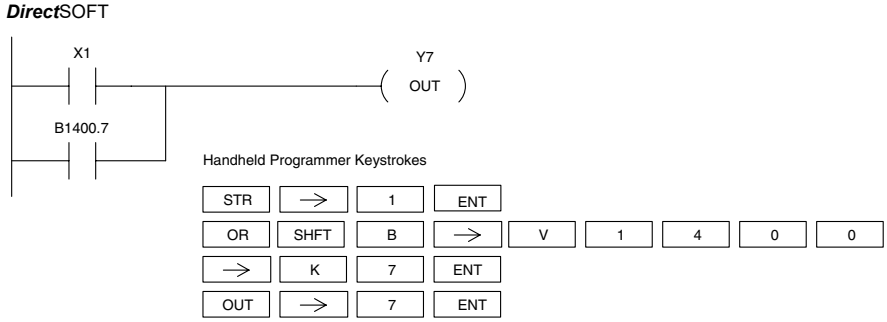
DS	Implied
HPP	Used

The Or Not Bit-of-Word instruction will logically OR a normally closed contact in parallel with another contact in a rung. Status of the contact will be opposite the state of the bit referenced in the associated memory location.

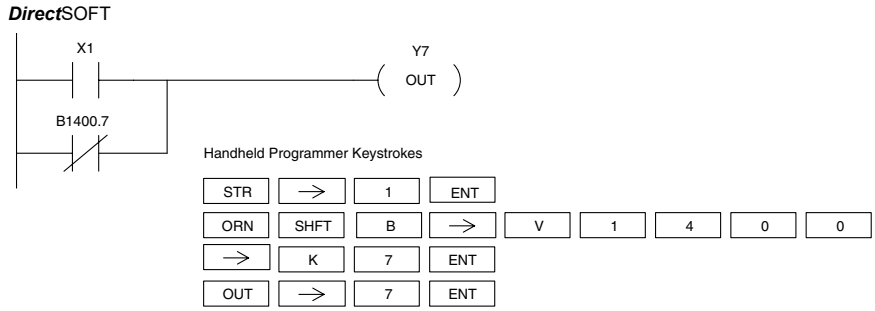


Operand Data Type		DL06 Range	
	A	aaa	bb
V-memory	B	See memory map	0 to 15
Pointer	PB	See memory map	0 to 15

In the following Or Bit-of-Word example, when input X1 or bit 7 of V1400 is on, output Y5 will energize.



In the following Or Bit-of-Word example, when input X1 is on or bit 7 of V1400 is off, output Y7 will energize.



AND (AND)

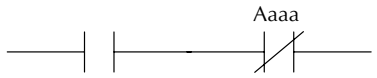
DS	Implied
HPP	Used

The AND instruction logically ands a normally open contact in series with another contact in a rung. The status of the contact will be the same state as the associated image register point or memory location.

AND NOT (ANDN)

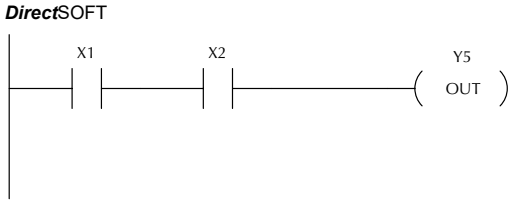
DS	Implied
HPP	Used

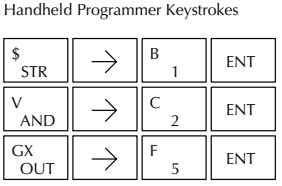
The AND NOT instruction logically ands a normally closed contact in series with another contact in a rung. The status of the contact will be opposite the state of the associated image register point or memory location



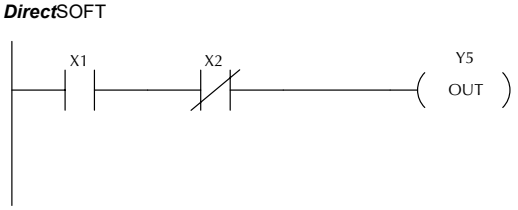
Operand Data Type		DL06 Range
	A	aaa
Inputs	X	0–777
Outputs	Y	0–777
Control Relays	C	0–1777
Stage	S	0–1777
Timer	T	0–377
Counter	CT	0–177
Special Relay	SP	0–777

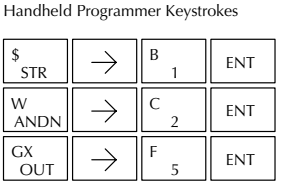
In the following And example, when input X1 and X2 are on output Y5 will energize.





In the following And Not example, when input X1 is on and X2 is off output Y5 will energize.





AND Bit-of-Word (AND)

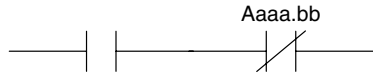
DS	Implied
HPP	Used

The And Bit-of-Word instruction logically ands a normally open contact in series with another contact in a rung. The status of the contact will be the same state as the bit referenced in the associated memory location.

AND Not Bit-of-Word (ANDN)

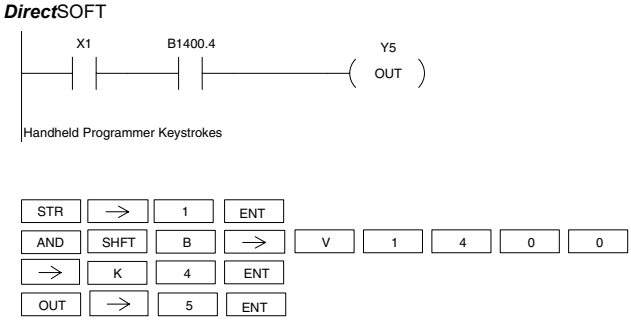
DS	Implied
HPP	Used

The And Not Bit-of-Word instruction logically ands a normally closed contact in series with another contact in a rung. The status of the contact will be opposite the state of the bit referenced in the associated memory location.

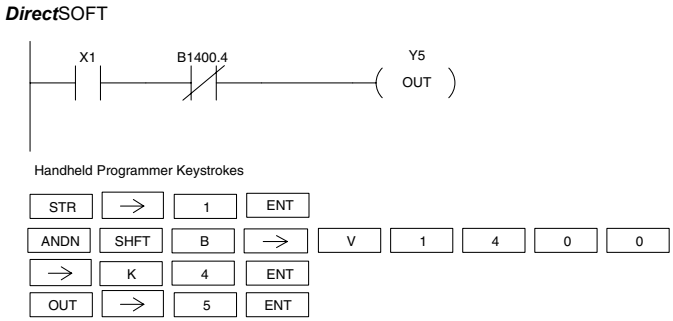


Operand Data Type		DL06 Range	
	A	aaa	bb
V-memory	B	See memory map	0 to 15
Pointer	PB	See memory map	0 to 15

In the following And Bit-of-Word example, when input X1 and bit 4 of V1400 is on output Y5 will energize.



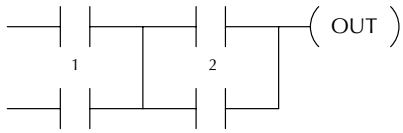
In the following And Not Bit-of-Word example, when input X1 is on and bit 4 of V1400 is off output Y5 will energize.



And Store (ANDSTR)

DS	Implied
HPP	Used

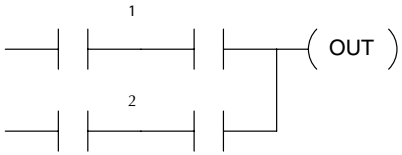
The And Store instruction logically ands two branches of a rung in series. Both branches must begin with the Store instruction.



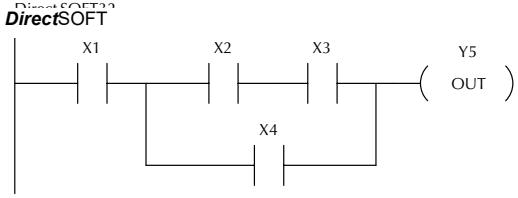
OR Store (ORSTR)

DS	Implied
HPP	Used

The Or Store instruction logically ORs two branches of a rung in parallel. Both branches must begin with the Store instruction.



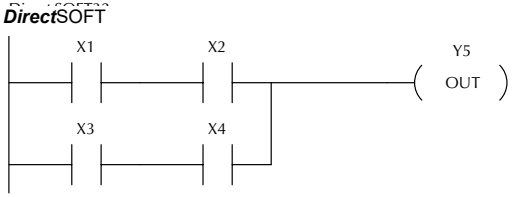
In the following And Store example, the branch consisting of contacts X2, X3, and X4 have been anded with the branch consisting of contact X1.



Handheld Programmer Keystrokes

$ STR	→	B 1	ENT
$ STR	→	C 2	ENT
V AND	→	D 3	ENT
Q OR	→	E 4	ENT
L ANDST	ENT		
GX OUT	→	F 5	ENT

In the following Or Store example, the branch consisting of X1 and X2 have been OR'd with the branch consisting of X3 and X4.



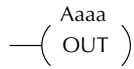
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT
V AND	→	C 2	ENT
$ STR	→	D 3	ENT
V AND	→	E 4	ENT
M ORST	ENT		
GX OUT	→	F 5	ENT

Out (OUT)

DS	Used
HPP	Used

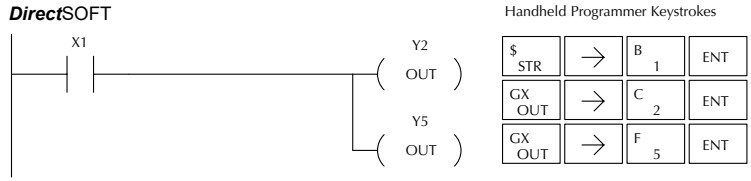
The Out instruction reflects the status of the rung (on/off) and outpu the discrete (on/off) state to the specified image register point or memc location.



Multiple Out instructions referencing the same discrete location should not be used since only the last Out instruction in the program will control the physical output point. Instead, use the next instruction, the Or Out.

Operand Data Type		DL06 Range
	A	aaa
Inputs	X	0–777
Outputs	Y	0–777
Control Relays	C	0–1777

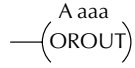
In the following Out example, when input X1 is on, output Y2 and Y5 will energize.



Or Out (OROUT)

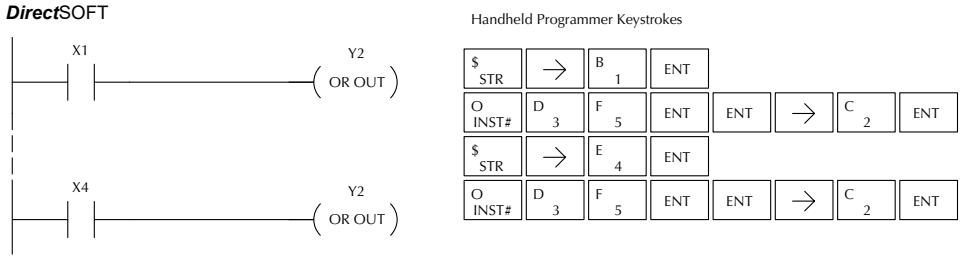
DS	Usied
HPP	Used

The Or Out instruction allows more than one rung of discrete logic to control a single output. Multiple Or Out instructions referencing the same output coil may be used, since *all* contacts controlling the output are logically OR'd together. If the status of *any* rung is on, the output will also be on.



Operand Data Type		DL06 Range
	A	aaa
Inputs	X	0–777
Outputs	Y	0-777
Control Relays	C	0–1777

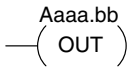
In the following example, when X1 or X4 is on, Y2 will energize.



Out Bit-of-Word (OUT)

DS	Used
HPP	Used

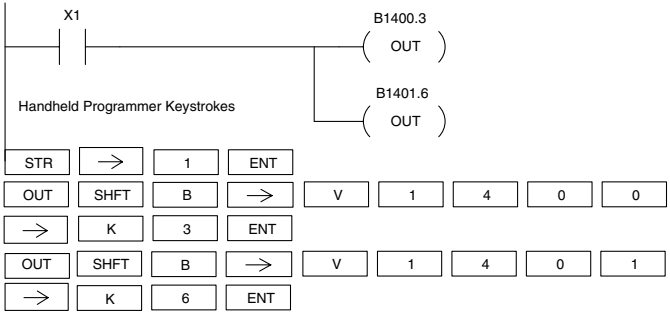
The Out Bit-of-Word instruction reflects the status of the rung (on/off) and outputs the discrete (on/off) state to the specified bit in the referenced memory location. Multiple Out Bit-of-Word instructions referencing the same bit of the same word generally should not be used since only the last Out instruction in the program will control the status of the bit.



Operand Data Type		DL06 Range	
	A	aaa	bb
V-memory	B	See memory map	0 to 15
Pointer	PB	See memory map	0 to 15

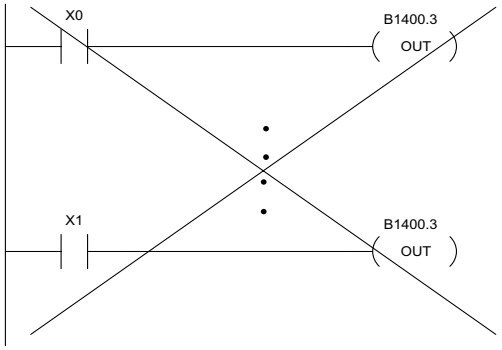
NOTE: *If the Bit-of-Word is entered as V1400.3 in* ***DirectSOFT****, it will be converted to B1400.3. Bit-of-Word can also be entered as B1400.3.*

DirectSOFT



In the following Out Bit-of-Word example, when input X1 is on, bit 3 of V1400 and bit 6 of V1401 will turn on.

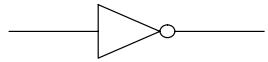
The following Out Bit-of-Word example contains two Out Bit-of-Word instructions using the same bit in the same memory word. The final state bit 3 of V1400 is ultimately controlled by the last rung of logic referencing it. X1 will override the logic state controlled by X0. To avoid this situation, multiple outputs using the same location must not be used in programming.



Not (NOT)

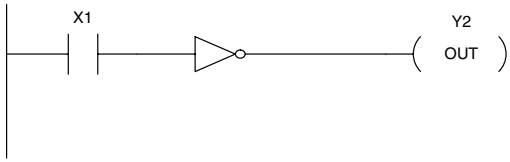
DS	Used
HPP	Used

The Not instruction inverts the status of the rung at the point o the instruction.



In the following example, when X1 is off, Y2 will energize. This is because the Not instruction inverts the status of the rung at the Not instruction.

DirectSOFT



Handheld Programmer Keystrokes

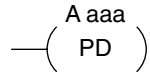
$ STR	→	B 1	ENT	
SHFT	N TMR	O INST#	T MLR	ENT
GX OUT	→	C 2	ENT	

***NOTE**: **Direct**SOFT Release 1.1i and later supports the use of the NOT instruction. The above example rung is merely intended to show the visual representation of the NOT instruction. The NOT instruction can only be selected in **DirectSOFT** from the Instruction Browser. The rung cannot be created or displayed in **Direct**SOFT versions earlier than 1.1i.*

Positive Differential (PD)

DS	Used
HPP	Used

The Positive Differential instruction is typically known as a one shot. When the input logic produces an off to on transition, the output will energize for one CPU scan.



In the following example, every time X1 makes an Off-to-On transition, C0 will energize for one scan.

Operand Data Type		DL06 Range
	A	aaa
Inputs	X	0–777
Outputs	Y	0–777
Control Relays	C	0–1777

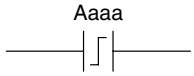
DirectSOFT

Handheld Programmer Keystrokes

$ STR	→	B 1	ENT			
SHFT	P CV	SHFT	D 3	→	A 0	

Store Positive Differential (STRPD)

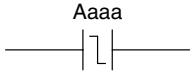
DS	Used
HPP	Used



The Store Positive Differential instruction begins a new rung or an additional branch in a rung with a contact. The contact closes for one CPU scan when the state of the associated image register point makes an Off-to-On transition. Thereafter, the contact remains open until the next Off-to-On transition (the symbol inside the contact represents the transition). This function is sometimes called a "one-shot". This contact will also close on a program-to-run transition if it is within a retentive range.

Store Negative Differential (STRND)

DS	Used
HPP	Used



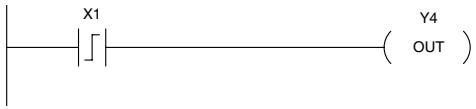
The Store Negative Differential instruction begins a new rung or an additional branch in a rung with a contact. The contact closes for one CPU scan when the state of the associated image register point makes an On-to-Off transition. Thereafter, the contact remains open until the next On-to-Off transition (the symbol inside the contact represents the transition).

NOTE: *When using* ***DirectSOFT****, these instructions can only be entered from the Instruction Browser.*

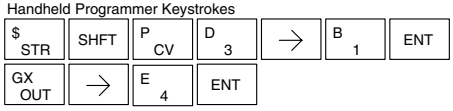
Operand Data Type	A	DL06 Range
		aaa
Inputs	X	0–777
Outputs	Y	0–777
Control Relays	C	0–1777
Stage	S	0–1777
Timer	T	0–377
Counter	CT	0–177

In the following example, each time X1 makes an Off-to-On transition, Y4 will energize for one scan.

DirectSOFT



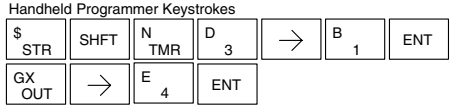
Handheld Programmer Keystrokes



In the following example, each time X1 makes an On-to-Off transition, Y4 will energize for one scan.

DirectSOFT

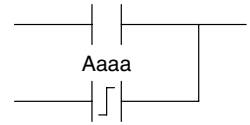
Handheld Programmer Keystrokes



Or Positive Differential (ORPD)

DS	Implied
HPP	Used

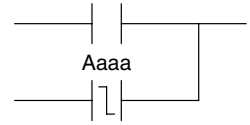
The Or Positive Differential instruction logically ors a contact in parallel with another contact in a rung. The status of the contact will be open until the associated image register point makes an Off-to-On transition, closing it for one CPU scan. Thereafter, it remains open until another Off-to-On transition.



Or Negative Differential (ORND)

DS	Implied
HPP	Used

The Or Negative Differential instruction logically ors a contact in parallel with another contact in a rung. The status of the contact will be open until the associated image register point makes an On-to-Off transition, closing it for one CPU scan. Thereafter, it remains open until another On-to-Off transition.

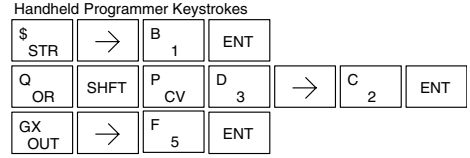


Operand Data Type		DL06 Range
	A	aaa
Inputs	X	0–777
Outputs	Y	0–777
Control Relays	C	0–1777
Stage	S	0–1777
Timer	T	0–377
Counter	CT	0–177

In the following example, Y 5 will energize whenever X1 is on, or for one CPU scan when X2 transitions from Off to On.

DirectSOFT

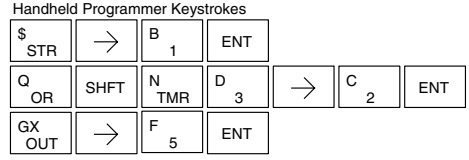
Handheld Programmer Keystrokes



In the following example, Y 5 will energize whenever X1 is on, or for one CPU scan when X2 transitions from On to Off.

DirectSOFT

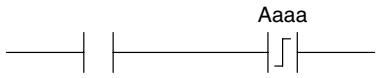
Handheld Programmer Keystrokes



And Positive Differential (ANDPD)

DS	Implied
HPP	Used

The And Positive Differential instruction logically ands a normally open contact in series with another contact in a rung. The status of the contact will be open until the associated image register point makes an Off-to-On transition, closing it for one CPU scan. Thereafter, it remains open until another Off-to-On transition.



And Negative Differential (ANDND)

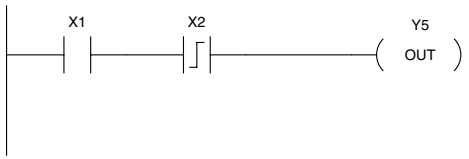
DS	Implied
HPP	Used

The And Negative Differential instruction logically ands a normally open contact in series with another contact 5-22in a rung. The status of the contact will be open until the associated image register point makes an On-to-Off transition, closing it for one CPU scan. Thereafter, it remains open until another On-to-Off transition.

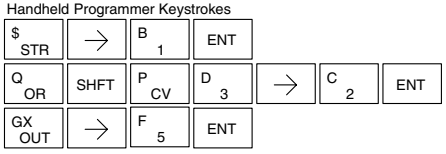
Operand Data Type		DL06 Range
	A	aaa
Inputs	X	0–777
Outputs	Y	0–777
Control Relays	C	0–1777
Stage	S	0–1777
Timer	T	0–377
Counter	CT	0–177

In the following example, Y5 will energize for one CPU scan whenever X1 is on and X2 transitions from Off to On.

*Direct*SOFT

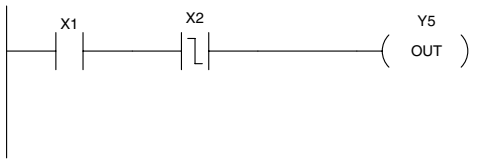


Handheld Programmer Keystrokes

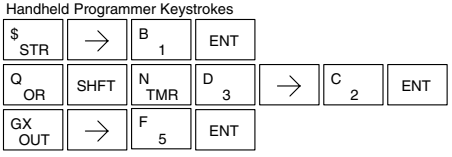


In the following example, Y5 will energize for one CPU scan whenever X1 is on and X2 transitions from On to Off.

*Direct*SOFT



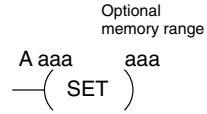
Handheld Programmer Keystrokes



Set (SET)

DS	Used
HPP	Used

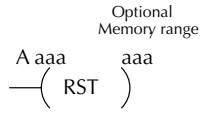
The Set instruction sets or turns on an image register point/memory location or a consecutive range of image register points/memory locations. Once the point/location is set it will remain on until it is reset using the Reset instruction. It is not necessary for the input controlling the Set instruction to remain on.



Reset (RST)

DS	Used
HPP	Used

The Reset instruction resets or turns off an image register point/memory location or a range of image registers points/memory locations. Once the point/location is reset, it is not necessary for the input to remain on.



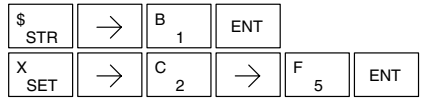
In the following example when X1 is on, Y2 through Y5 will energize.

Operand Data Type		DL06 Range
	A	aaa
Inputs	X	0–777
Outputs	Y	0–777
Control Relays	C	0–1777
Stage	S	0–1777
Timer	T	0–377
Counter	CT	0–177

In the following example when X1 is on, Y2 through Y5 will be reset or de–energized.

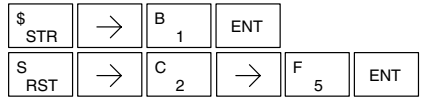
DirectSOFT

Handheld Programmer Keystrokes



DirectSOFT

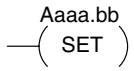
Handheld Programmer Keystrokes



Set Bit-of-Word (SET)

DS	Used
HPP	Used

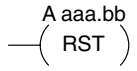
The Set Bit-of-Word instruction sets or turns on a bit in a V-memory location. Once the bit is set, it will remain on until it is reset using the Reset Bit-of-Word instruction. It is not necessary for the input controlling the Set Bit-of-Word instruction to remain on.



Reset Bit-of-Word (RST)

DS	Used
HPP	Used

The Reset Bit-of-Word instruction resets or turns off a bit in a V-memory location. Once the bit is reset it is not necessary for the input to remain on.



Operand Data Type		DL06 Range	
	A	aaa	bb
V-memory	B	See memory map	0 to 15
Pointer	PB	See memory map	0 to 15

In the following example when X1 turns on, bit 1 in V1400 is set to the on state.

DirectSOFT

Handheld Programmer Keystrokes

STR	→	1	ENT					
SET	SHFT	B	→	V	1	4	0	0
→	K	1	ENT					

In the following example, when X2 turns on, bit 1 in V1400 is reset to the off state.

DirectSOFT

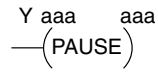
Handheld Programmer Keystrokes

STR	→	2	ENT					
RST	SHFT	B	→	V	1	4	0	0
→	K	1	ENT					

Pause (PAUSE)

DS	Used
HPP	Used

The Pause instruction disables the output update on a range of outputs. The ladder program will continue to run and update the image register. However, the outputs in the range specified in the Pause instruction will be turned off at the output points.

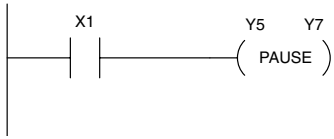


Operand Data Type		DL06 Range
	A	aaa
Outputs	Y	0–777

In the following example, when X1 is ON, Y5–Y7 will be turned OFF. The execution of the ladder program will not be affected.

DirectSOFT

DirectSOFT32



Since the D2–HPP Handheld Programmer does not have a specific Pause key, you can use the corresponding instruction number for entry (#960), or type each letter of the command.

Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
O INST#	J 9	G 6	A 0	ENT	ENT	→	D 3	→	F 5	ENT

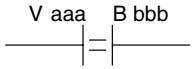
In some cases, you may want certain output points in the specified pause range to operate normally. In that case, use Aux 58 to over-ride the Pause instruction.

Comparative Boolean

Store If Equal (STRE)

DS	Implied
HPP	Used

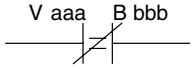
The Store If Equal instruction begins a new rung or additional branch in a rung with a normally open comparative contact. The contact will be on when Vaaa equals Bbbb .



Store If Not Equal (STRNE)

DS	Implied
HPP	Used

The Store If Not Equal instruction begins a new rung or additional branch in a rung with a normally closed comparative contact. The contact will be on when Vaaa does not equal Bbbb.



Operand Data Type		DL06 Range	
	B	aaa	bbb
V-memory	V	See memory map	See memory map
Pointer	P	See memory map	See memory map
Constant	K	—	0–9999

In the following example, when the BCD value in V-memory location V2000 = 4933, Y3 will energize.

DirectSOFT

Handheld Programmer Keystrokes

$ STR	SHFT	E 4	→	C 2	A 0	A 0	A 0
→	E 4	J 9	D 3	D 3	ENT		
GX OUT	→	D 3	ENT				

In the following example, when the value in V-memory location V2000≠5060, Y3 will energize.

DirectSOFT

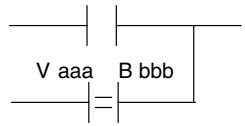
Handheld Programmer Keystrokes

SP STRN	SHFT	E 4	→	C 2	A 0	A 0	A 0
→	F 5	A 0	G 6	A 0	ENT		
GX OUT	→	D 3	ENT				

Or If Equal (ORE)

DS	Implied
HPP	Used

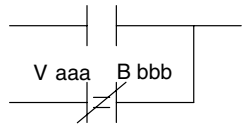
The Or If Equal instruction connects a normally open comparative contact in parallel with another contact. The contact will be on when Vaaa = Bbbb.



Or If Not Equal (ORNE)

DS	Implied
HPP	Used

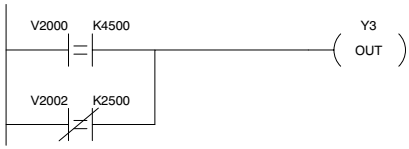
The Or If Not Equal instruction connects a normally closed comparative contact in parallel with another contact. The contact will be on when Vaaa does not equal Bbbb.



Operand Data Type		DL06 Range	
	B	aaa	bbb
V-memory	V	See memory map	See memory map
Pointer	P	See memory map	See memory map
Constant	K	—	0–9999

In the following example, when the BCD value in V-memory location V2000 = 4500 or V2002 ≠ 2500, Y3 will energize.

*Direct*SOFT

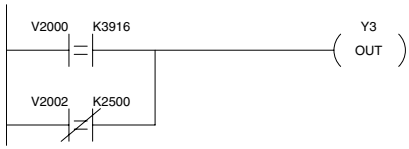


Handheld Programmer Keystrokes

$ STR	SHFT	E 4	→	C 2	A 0	A 0	A 0	→
E 4	F 5	A 0	A 0	ENT				
Q OR	SHFT	E 4	→	C 2	A 0	A 0	C 2	→
C 2	D 3	E 4	F 5	ENT				
GX OUT	→	D 3	ENT					

In the following example, when the BCD value in V-memory location V2000 = 3916 or V2002 ≠ 2500, Y3 will energize.

*Direct*SOFT



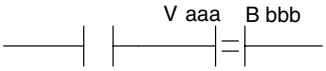
Handheld Programmer Keystrokes

$ STR	SHFT	E 4	→	C 2	A 0	A 0	A 0	→
D 3	J 9	B 1	G 6	ENT				
R ORN	SHFT	E 4	→	C 2	A 0	A 0	C 2	→
C 2	F 5	A 0	A 0	ENT				
GX OUT	→	D 3	ENT					

And If Equal (ANDE)

DS	Implied
HPP	Used

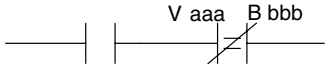
The And If Equal instruction connects a normally open comparative contact in series with another contact. The contact will be on when Vaaa = Bbbb.



And If Not Equal (ANDNE)

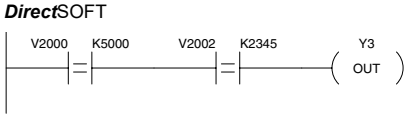
DS	Implied
HPP	Used

The And If Not Equal instruction connects a normally closed comparative contact in series with another contact. The contact will be on when Vaaa does not equal Bbbb.



Operand Data Type		DL06 Range	
	B	aaa	bbb
V-memory	V	See memory map	See memory map
Pointer	P	See memory map	See memory map
Constant	K	—	0–9999

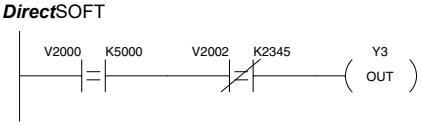
In the following example, when the BCD value in V-memory location V2000 = 5000 and V2002 = 2345, Y3 will energize.



Handheld Programmer Keystrokes

$ STR	SHFT	E 4	→	C 2	A 0	A 0	A 0	→
F 5	A 0	A 0	A 0	ENT				
V AND	SHFT	E 4	→	C 2	A 0	A 0	C 2	→
C 2	D 3	E 4	F 5	ENT				
GX OUT	→	D 3	ENT					

In the following example, when the BCD value in V-memory location V2000 = 5000 and V2002 ≠ 2345, Y3 will energize.



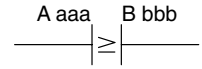
Handheld Programmer Keystrokes

$ STR	SHFT	E 4	→	C 2	A 0	A 0	A 0	→
F 5	A 0	A 0	A 0	ENT				
V AND	SHFT	E 4	→	C 2	A 0	A 0	C 2	→
C 2	D 3	E 4	F 5	ENT				
GX OUT	→	D 3	ENT					

Store (STR)

DS	Implied
HPP	Used

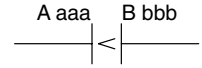
The Comparative Store instruction begins a new rung or additional branch in a rung with a normally open comparative contact. The contact will be on when Aaaa is equal to or greater than Bbbb.



Store Not (STRN)

DS	Implied
HPP	Used

The Comparative Store Not instruction begins a new rung or additional branch in a rung with a normally closed comparative contact. The contact will be on when Aaaa < Bbbb.

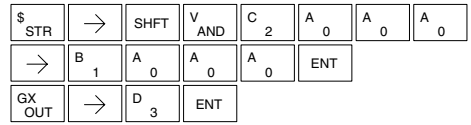


Operand Data Type		DL06 Range	
	A/B	aaa	bbb
V-memory	V	See memory map	See memory map
Pointer	p	See memory map	See memory map
Constant	K	—	0–9999
Timer	TA	0–377	
Counter	CTA	0–177	

In the following example, when the BCD value in V-memory location V2000 ≥ 1000, Y3 will energize.

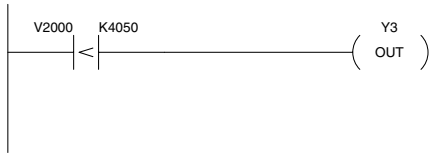
DirectSOFT

Handheld Programmer Keystrokes

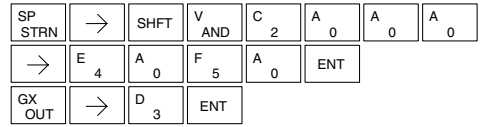


In the following example, when the value in V-memory location V2000 < 4050, Y3 will energize.

DirectSOFT



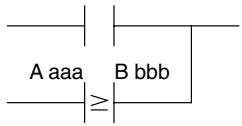
Handheld Programmer Keystrokes



Or (OR)

DS	Implied
HPP	Used

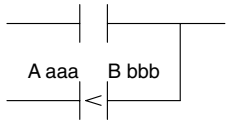
The Comparative Or instruction connects a normally open comparative contact in parallel with another contact. The contact will be on when Aaaa is equal to or greater than Bbbb.



Or Not (ORN)

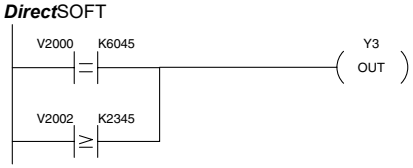
DS	Implied
HPP	Used

The Comparative Or Not instruction connects a normally closed comparative contact in parallel with another contact. The contact will be on when Aaaa < Bbbb.



Operand Data Type		DL06 Range	
	A/B	aaa	bbb
V-memory	V	See memory map	See memory map
Pointer	p	See memory map	See memory map
Constant	K	—	0–9999
Timer	TA	0–377	
Counter	CTA	0–177	

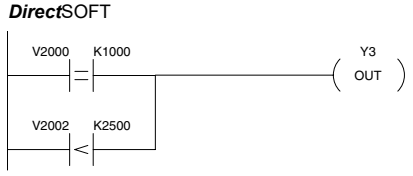
In the following example, when the BCD value in V-memory location V2000 = 6045 or V2002 ≥ 2345, Y3 will energize.



Handheld Programmer Keystrokes

$ STR	SHFT	E 4	→	C 2	A 0	A 0	A 0	→
G 6	A 0	E 4	F 5	ENT				
Q OR	→	SHFT	V AND	C 2	A 0	A 0	C 2	→
C 2	D 3	E 4	F 5	ENT				
GX OUT	→	D 3	ENT					

In the following example when the BCD value in V-memory location V2000 = 1000 or V2002 < 2500, Y3 will energize.



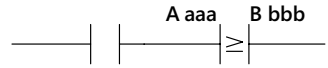
Handheld Programmer Keystrokes

$ STR	SHFT	E 4	→	C 2	A 0	A 0	A 0	→
B 1	A 0	A 0	A 0	ENT				
R ORN	→	SHFT	V AND	C 2	A 0	A 0	C 2	→
C 2	F 5	A 0	A 0	ENT				
GX OUT	→	D 3	ENT					

And (AND)

DS	Implied
HPP	Used

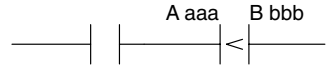
The Comparative And instruction connects a normally open comparative contact in series with another contact. The contact will be on when Aaaa is equal to or greater than Bbbb.



And Not (ANDN)

DS	Implied
HPP	Used

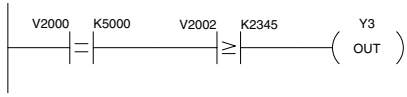
The Comparative And Not instruction connects a normally closed comparative contact in series with another contact. The contact will be on when Aaaa < Bbbb.



Operand Data Type		DL06 Range	
	A/B	aaa	bbb
V-memory	V	See memory map	See memory map
Pointer	p	See memory map	See memory map
Constant	K	—	0–9999
Timer	TA	0–377	
Counter	CTA	0–177	

In the following example, when the value in BCD V-memory location V2000 = 5000, and V2002 ≥ 2345, Y3 will energize.

DirectSOFT

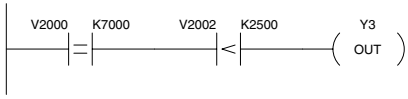


Handheld Programmer Keystrokes

$ STR	SHFT	E 4	→	C 2	A 0	A 0	A 0	→
F 5	A 0	A 0	A 0	ENT				
V AND	→	SHFT	V AND	C 2	A 0	A 0	C 2	→
C 2	D 3	E 4	F 5	ENT				
GX OUT	→	D 3	ENT					

In the following example, when the value in V-memory location V2000 = 7000 and V2002 < 2500, Y3 will energize.

DirectSOFT



Handheld Programmer Keystrokes

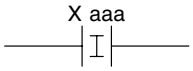
$ STR	SHFT	E 4	→	C 2	A 0	A 0	A 0	→
H 7	A 0	A 0	A 0	ENT				
W ANDN	→	SHFT	V AND	C 2	A 0	A 0	C 2	→
C 2	F 5	A 0	A 0	ENT				
GX OUT	→	SHFT	Y AND	D 3	ENT			

Immediate Instructions

Store Immediate (STRI)

DS	Implied
HPP	Used

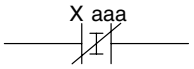
The Store Immediate instruction begins a new rung or additional branch in a rung. The status of the contact will be the same as the status of the associated input point *at the time the instruction is executed*. The image register is not updated.



Store Not Immediate (STRNI)

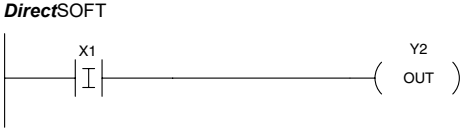
DS	Implied
HPP	Used

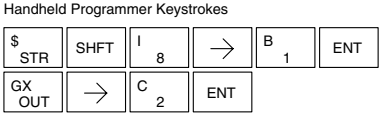
The Store Not Immediate instruction begins a new rung or additional branch in a rung. The status of the contact will be opposite the status of the associated input point *at the time the instruction is executed*. The image register is not updated.



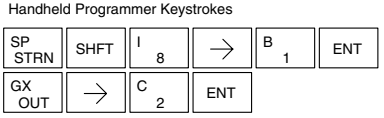
Operand Data Type		DL06 Range
		aaa
Inputs	X	0–777

In the following example, when X1 is on, Y2 will energize.





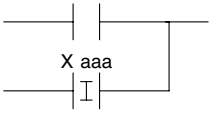
In the following example, when X1 is off, Y2 will energize.



Or Immediate (ORI)

DS	Implied
HPP	Used

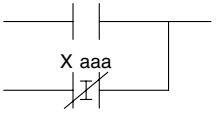
The Or Immediate connects two contacts in parallel. The status of the contact will be the same as the status of the associated input point *at the time the instruction is executed*. The image register is not updated.



Or Not Immediate (ORNI)

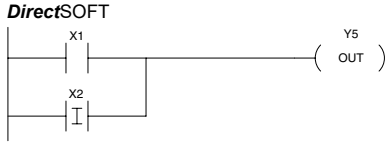
DS	Implied
HPP	Used

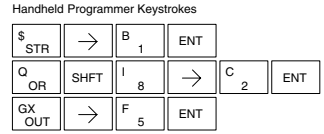
The Or Not Immediate connects two contacts in parallel. The status of the contact will be opposite the status of the associated input point *at the time the instruction is executed*. The image register is not updated.



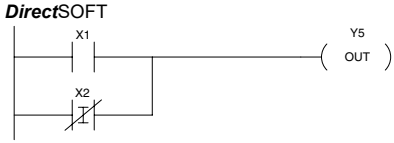
Operand Data Type		DL06 Range
		aaa
Inputs	X	0–777

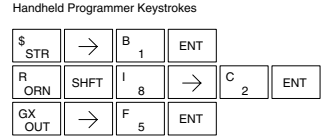
In the following example, when X1 or X2 is on, Y5 will energize.





In the following example, when X1 is on or X2 is off, Y5 will energize.

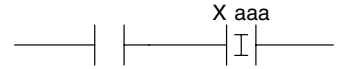




And Immediate (ANDI)

DS	Implied
HPP	Used

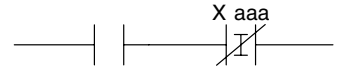
The And Immediate instruction connects two contacts in series. The status of the contact will be the same as the status of the associated input point *at the time the instruction is executed*. The image register is not updated.



And Not Immediate (ANDNI)

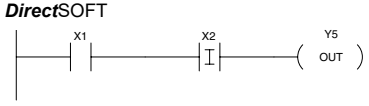
DS	Implied
HPP	Used

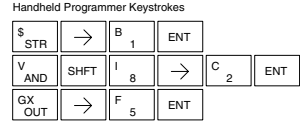
The And Not Immediate instruction connects two contacts in series. The status of the contact will be opposite the status of the associated input point *at the time the instruction is executed*. The image register is not updated.



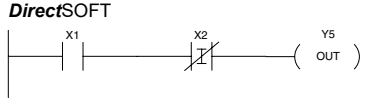
Operand Data Type		DL06 Range
		aaa
Inputs	X	0–777

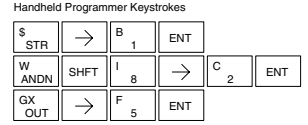
In the following example, when X1 and X2 are on, Y5 will energize.





In the following example, when X1 is on and X2 is off, Y5 will energize.

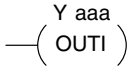




Out Immediate (OUTI)

DS	Used
HPP	Used

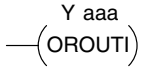
The Out Immediate instruction reflects the status of the rung (on/off) and outputs the discrete (on/off) status to the specified module output point and the image register *at the time the instruction is executed*. If multiple Out Immediate instructions referencing the same discrete point are used, it is possible for the module output status to change multiple times in a CPU scan. See Or Out Immediate.



Or Out Immediate (OROUTI)

DS	Used
HPP	Used

The Or Out Immediate instruction has been designed to use more than 1 rung of discrete logic to control a single output. Multiple Or Out Immediate instructions referencing the same output coil may be used, since all contacts controlling the output are OR'd together. If the status of any rung is on *at the time the instruction is executed*, the output will also be on.

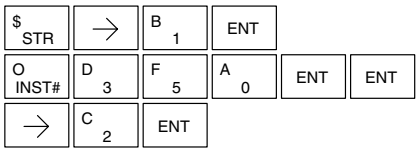


Operand Data Type		DL06 Range
		aaa
Outputs	Y	0–777

In the following example, when X1 is on, output point Y2 on the output module will turn on. For instruction entry on the Handheld Programmer, you can use the instruction number (#350) as shown, or type each letter of the command.

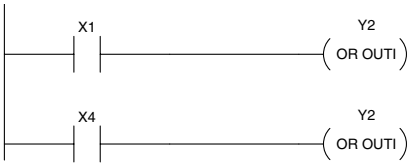
DirectSOFT

Handheld Programmer Keystrokes

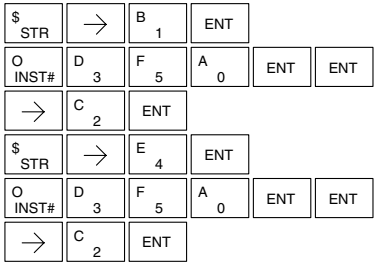


In the following example, when X1 or X4 is on, Y2 will energize.

DirectSOFT



Handheld Programmer Keystrokes



Out Immediate Formatted (OUTIF)

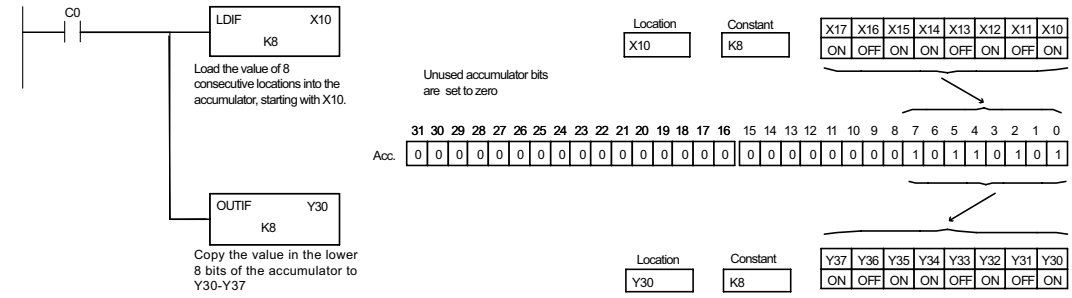
DS	Used
HPP	Used

The Out Immediate Formatted instruction outputs a 1–32 bit binary value from the accumulator to specified output points *at the time the instruction is executed*. Accumulator bits that are not used by the instruction are set to zero.

Operand Data Type		DL06 Range
		aaa
Outputs	Y	0-777
Constant	K	1-32

In the following example, when C0 is on,the binary pattern for X10 –X17 is loaded into the accumulator using the Load Immediate Formatted instruction. The binary pattern in the accumulator is written to Y30–Y37 using the Out Immediate Formatted instruction. This technique is useful to quickly copy an input pattern to outputs (without waiting for the CPU scan).

DirectSOFT



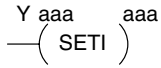
Handheld Programmer Keystrokes

$ STR	→	NEXT	NEXT	NEXT	NEXT	A 0	ENT				
SHFT	L ANDST	D 3	I 8	F 5	→	B 1	A 0	→	I 8	ENT	
GX OUT	SHFT	I 8	F 5	→	D 3	A 0	→	I 8	ENT		

Set Immediate (SETI)

DS	Used
HPP	Used

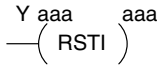
The Set Immediate instruction immediately sets, or turns on an output or a range of outputs in the image register and the corresponding output point(s) *at the time the instruction is executed*. Once the outputs are set, it is not necessary for the input to remain on. The Reset Immediate instruction can be used to reset the outputs.



Reset Immediate (RSTI)

DS	Used
HPP	Used

The Reset Immediate instruction immediately resets, or turns off, an output or a range of outputs in the image register and the output point(s) *at the time the instruction is executed.* Once the outputs are reset, it is not necessary for the input to remain on.

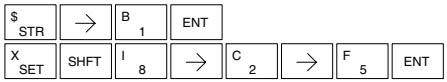


Operand Data Type		DL06 Range
		aaa
Ouputs	Y	0–777

In the following example, when X1 is on, Y2 through Y5 will be set on in the image register and on the corresponding output points.

DirectSOFT

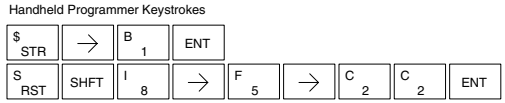
Handheld Programmer Keystrokes



In the following example, when X1 is on, Y5 through Y22 will be reset (off) in the image register and on the corresponding output module(s).

DirectSOFT

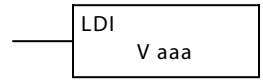
Handheld Programmer Keystrokes



Load Immediate (LDI)

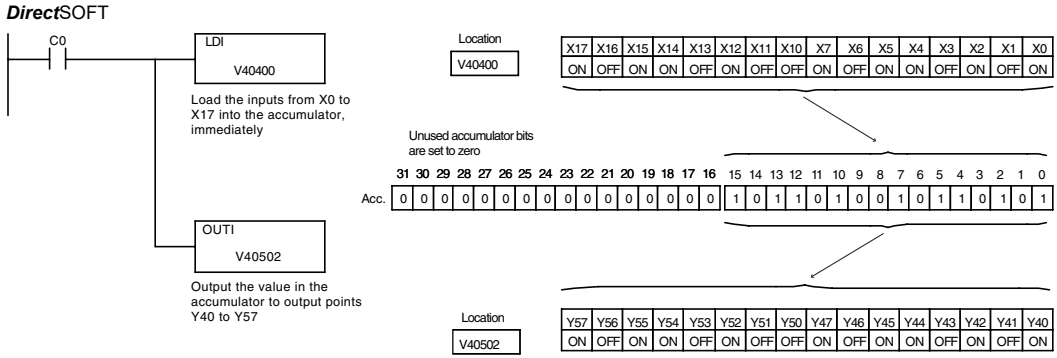
DS	Used
HPP	Used

The Load Immediate instruction loads a 16-bit V-memory value into the accumulator. The valid address range includes all input point addresses on the local base. The value reflects the current status of the input points *at the time the instruction is executed*. This instruction may be used instead of the LDIF instruction, which requires you to specify the number of input points.



Operand Data Type		DL06 Range
		aaa
Inputs	V	40400-40437

In the following example, when C0 is on, the binary pattern of X0–X17 will be loaded into the accumulator using the Load Immediate instruction. The Out Immediate instruction could be used to copy the 16 bits in the accumulator to output points, such as Y40–Y57. This technique is useful to quickly copy an input pattern to output points (without waiting for a full CPU scan to occur).



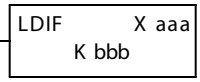
Handheld Programmer Keystrokes

$ STR	→	NEXT	NEXT	NEXT	NEXT	A 0	ENT			
SHFT	L ANDST	D 3	I 8	→	E 4	A 0	E 4	A 0	A 0	ENT
GX OUT	SHFT	I 8	→	NEXT	E 4	A 0	F 5	A 0	C 2	ENT

Load Immediate Formatted (LDIF)

DS	Used
HPP	Used

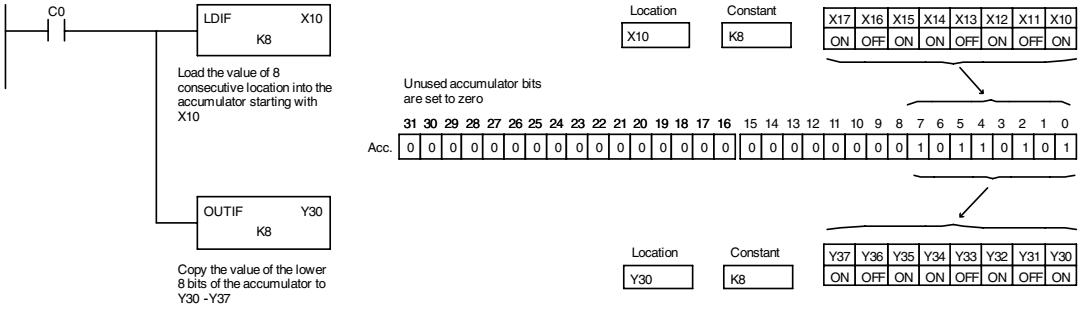
The Load Immediate Formatted instruction loads a 1–32 bit binary value into the accumulator. The value reflects the current status of the input module(s) *at the time the instruction is executed*. Accumulator bits that are not used by the instruction are set to zero.



Operand Data Type		DL06 Range	
		aaa	bbb
Inputs	X	0-777	--
Constant	K	--	1-32

In the following example, when C0 is on, the binary pattern of X10–X17 will be loaded into the accumulator using the Load Immediate Formatted instruction. The Out Immediate Formatted instruction could be used to copy the specified number of bits in the accumulator to the specified outputs on the output module, such as Y30–Y37. This technique is useful to quickly copy an input pattern to outputs (without waiting for the CPU scan).

***Direct*SOFT**



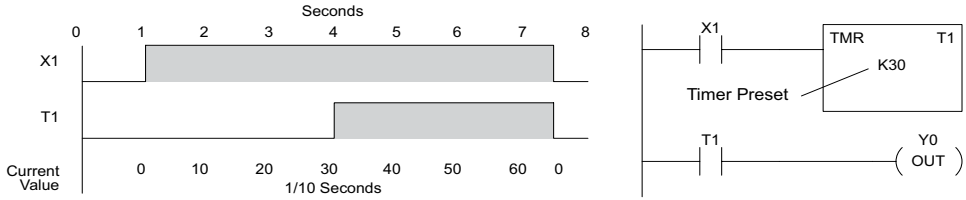
Handheld Programmer Keystrokes

$ STR	→	NEXT	NEXT	NEXT	NEXT	A 0	ENT			
SHFT	L ANDST	D 3	I 8	F 5	→	B 1	A 0	→	I 8	ENT
GX OUT	SHFT	I 8	F 5	→	D 3	A 0	→	I 8	ENT	

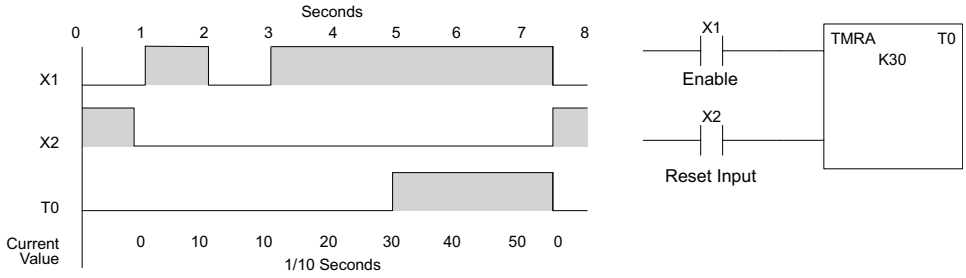
Timer, Counter and Shift Register Instructions

Using Timers

Timers are used to time an event for a desired period. The single input timer will time as long as the input is on. When the input changes from on to off, the timer current value is reset to 0. There is a tenth of a second and a hundredth of a second timer available with a maximum time of 999.9 and 99.99 seconds respectively. There is a discrete bit associated with each timer to indicate that the current value is equal to or greater than the preset value. The timing diagram below shows the relationship between the timer input, associated discrete bit, current value and timer preset.



There are those applications that need an accumulating timer, meaning it has the ability to time, stop, and then resume from where it previously stopped. The accumulating timer works similarly to the regular timer, but two inputs are required. The enable input starts and stops the timer. When the timer stops, the elapsed time is maintained. When the timer starts again, the timing continues from the elapsed time. When the reset input is turned on, the elapsed time is cleared and the timer will start at 0 when it is restarted. There is a tenth of a second and a hundredth of a second timer available with a maximum time of 9999999.9 and 999999.99 seconds respectively. The timing diagram below shows the relationship between the timer input, timer reset, associated discrete bit, current value and timer preset.

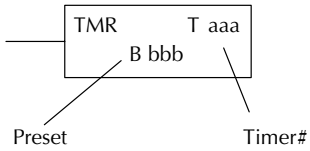


NOTE: *Decimal points are not used in these timers, but the decimal point is implied. The preset and current value for all four timers is in BCD format.*

Timer (TMR) and Timer Fast (TMRF)

DS	Used
HPP	Used

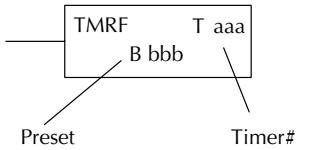
The Timer instruction is a 0.1 second single input timer that times to a maximum of 999.9 seconds. The Timer Fast instruction is a 0.01 second single input timer that times up to a maximum of 99.99 seconds. These timers will be enabled if the input logic is true (on) and will be reset to 0 if the input logic is false (off). Both timers use single word BCD values for the preset and current value. The decimal place is implied.



Instruction Specifications

Timer Reference (Taaa): Specifies the timer number.

Preset Value (Bbbb): Constant value (K) or a V-memory location specified in BCD.



Current Value: Timer current values, in BCD format, are accessed by referencing the associated V or T memory location*. For example, the timer current value for T3 physically resides in V-memory location V3.

Discrete Status Bit: The discrete status bit is referenced by the associated T memory location. Operating as a "timer done bit", it will be on if the current value is equal to or greater than the preset value. For example, the discrete status bit for Timer 2 is T2.

***NOTE**: A V-memory preset is required only if the ladder program or an Operator Interface unit must change the preset.*

Operand Data Type		DL06 Range	
	A/B	aaa	bbb
Timers	T	0–777	—
V-memory for preset values	V	—	400-677 1200–7377 7400–7577 10000-17777
Pointers (preset only)	P	—	400-677 1200–7377 7400–7577* 10000-17777
Constants (preset only)	K	—	0–9999
Timer discrete status bits	T/V	0–377 or V41100–41117	
Timer current values	V /T**	0–377	

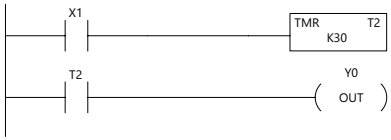
***NOTE:** *May be non-volatile if MOV instruction is used.*
*** With the HPP, both the Timer discrete status bits and current value are accessed with the same data reference. DirectSOFT uses separate references, such as "T2" for discrete status bit for Timer T2, and "TA2" for the current value of Timer T2.*

You can perform functions when the timer reaches the specified preset using the discrete status bit. Or, use comparative contacts to perform functions at different time intervals, based on one timer. The examples on the following page show these two methods of programming timers.

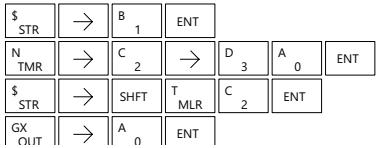
Timer Example Using Discrete Status Bits

In the following example, a single input timer is used with a preset of 3 seconds. The timer discrete status bit (T2) will turn on when the timer has timed for 3 seconds. The timer is reset when X1 turns off, turning the discrete status bit off and resetting the timer current value to 0.

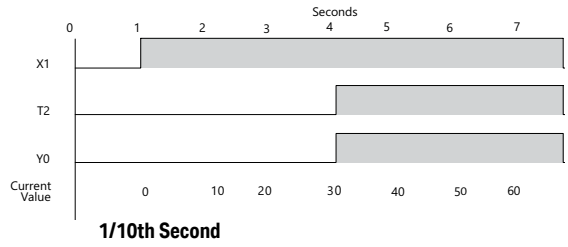
DirectSOFT



Handheld Programmer Keystrokes



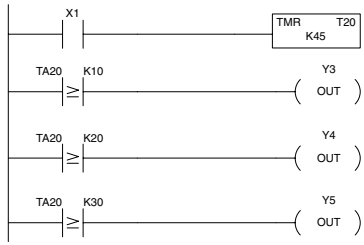
Timing Diagram



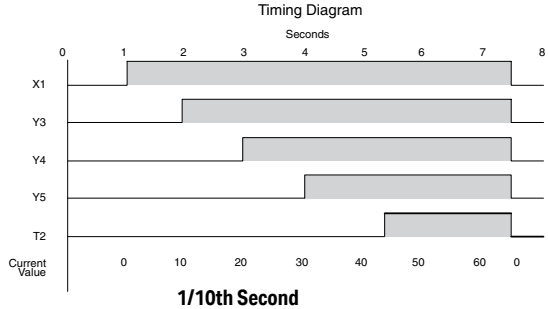
Timer Example Using Comparative Contacts

In the following example, a single input timer is used with a preset of 4.5 seconds. Comparative contacts are used to energize Y3, Y4, and Y5 at one second intervals respectively. When X1 is turned off, the timer will be reset to 0 and the comparative contacts will turn off Y3, Y4, and Y5.

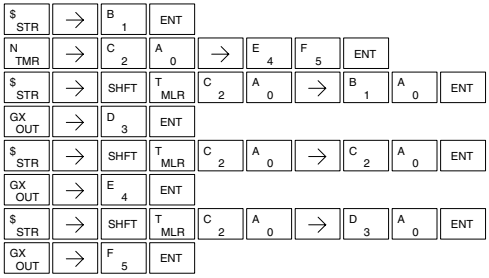
DirectSOFT



Timing Diagram



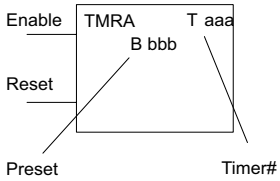
Handheld Programmer Keystrokes



Accumulating Timer (TMRA)

DS	Used
HPP	Used

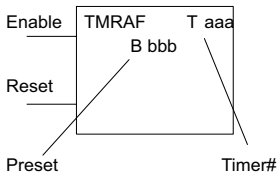
The Accumulating Timer is a 0.1 second two input timer that will time to a maximum of 9999999.9. *The TMRA uses two timer registers in V-memory.*



Accumulating Fast Timer (TMRAF)

DS	Used
HPP	Used

The Accumulating Fast Timer is a 0.01 second two-input timer that will time to a maximum of 99999.99. *The TMRA uses two timer registers in V-memory.*



Each timer uses two timer registers in V-memory. The preset and current values are in double word BCD format, and the decimal point is implied. These timers have two inputs, an enable and a reset. The timer starts timing when the enable is on and stops when the enable is off (without resetting the count). The reset will reset the timer when on and allow the timer to time when off.

Timer Reference (Taaa): Specifies the timer number.

Preset Value (Bbbb): Constant value (K) or V-memory.

Current Value: Timer current values are accessed by referencing the associated V or T memory location*. For example, the timer current value for T3 resides in V-memory, V3.

Discrete Status Bit: The discrete status bit is accessed by referencing the associated T memory location. Operating as a "timer done bit," it will be on if the current value is equal to or greater than the preset value. For example, the discrete status bit for timer 2 would be T2.

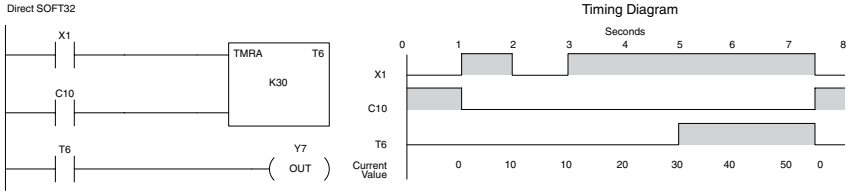
NOTE: *The accumulating timer uses two consecutive V-memory locations for the 8-digit value, therefore two consecutive timer locations. For example, if TMRA T1 is used, the next available timer number is T3.*
NOTE: *A V-memory preset is required if the ladder program or an OIP must be used to change the preset.*

Operand Data Type		DL06 Range	
	A/B	aaa	bbb
Timers	T	0–777	—
V-memory for preset values	V	—	400-677 1200–7377 7400–7577 10000-17777
Pointers (preset only)	P	—	400-677 1200–7377 7400–7577* 10000-17777
Constants (preset only)	K	—	0–99999999
Timer discrete status bits	T/V	0–377 or V41100–41117	
Timer current values	V /T**	0–377	

NOTE: **May be non-volatile if MOV instruction is used.** With the HPP, both the Timer discrete status bits and current value are accessed with the same data reference. DirectSOFT uses separate references, such as "T2" for discrete status bit for Timer T2, and "TA2" for the current value of Timer T2.*

Accumulating Timer Example using Discrete Status Bits

In the following example, a two input timer (accumulating timer) is used with a preset of 3 seconds. The timer discrete status bit (T6) will turn on when the timer has timed for 3 seconds. Notice, in this example, that the timer times for 1 second, stops for one second, then resumes timing. The timer will reset when C10 turns on, turning the discrete status bit off and resetting the timer current value to 0.



Handheld Programmer Keystrokes

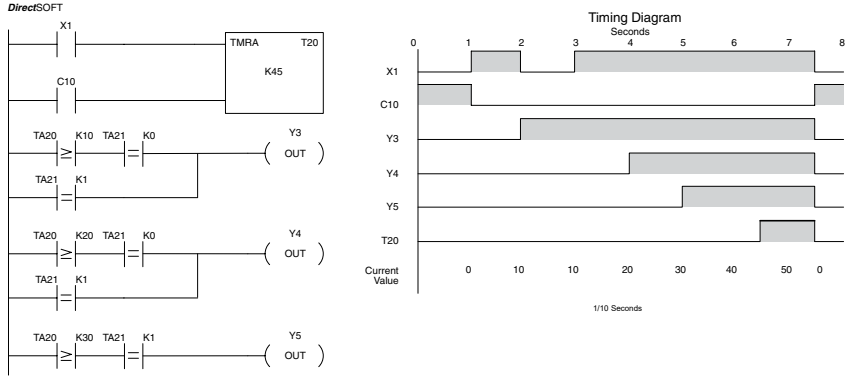
$ STR	→	B 1	ENT								
$ STR	→	SHFT	C 2	B 1	A 0	ENT					
N TMR	SHFT	A 0	→	G 6	→						

Handheld Programmer Keystrokes (cont)

D 3	A 0	ENT									
$ STR	→	SHFT	T MLR	G 6	ENT						
GX OUT	→	B 1	A 0	ENT							

Accumulator Timer Example Using Comparative Contacts

In the following example, a single input timer is used with a preset of 4.5 seconds. Comparative contacts are used to energized Y3, Y4, and Y5 at one second intervals respectively. The comparative contacts will turn off when the timer is reset.



Handheld Programmer Keystrokes

$ STR	→	B 1	ENT								
$ STR	→	SHFT	C 2	B 1	A 0	ENT					
N TMR	SHFT	A 0	→	C 2	A 0	→	E 4	F 5	ENT		
$ STR	→	SHFT	T MLR	C 2	A 0	→	B 1	A 0	ENT		
V AND	SHFT	E 4	→	SHFT	T MLR	C 2	B 1	→	A 0	ENT	
Q OR	SHFT	E 4	→	SHFT	T MLR	C 2	B 1	→	B 1	ENT	
GX OUT	→	D 3	ENT								

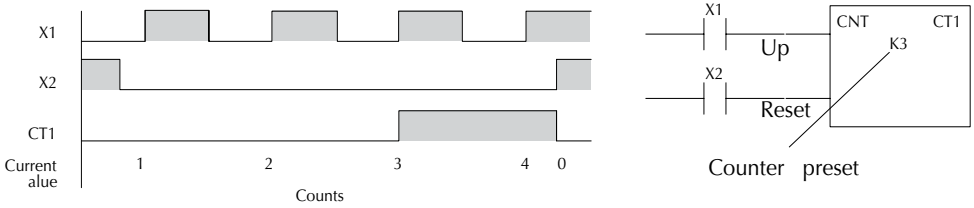
Handheld Programmer Keystrokes (cont'd)

$ STR	→	SHFT	T MLR	C 2	A 0	→	C 2	A 0	ENT		
V AND	SHFT	E 4	→	SHFT	T MLR	C 2	B 1	→	A 0	ENT	
Q OR	SHFT	E 4	→	SHFT	T MLR	C 2	B 1	→	B 1	ENT	
GX OUT	→	E 4	ENT								
$ STR	→	SHFT	T MLR	C 2	A 0	→	D 3	A 0	ENT		
V AND	SHFT	E 4	→	SHFT	T MLR	C 2	B 1	→	B 1	ENT	
GX OUT	→	F 5	ENT								

Using Counters

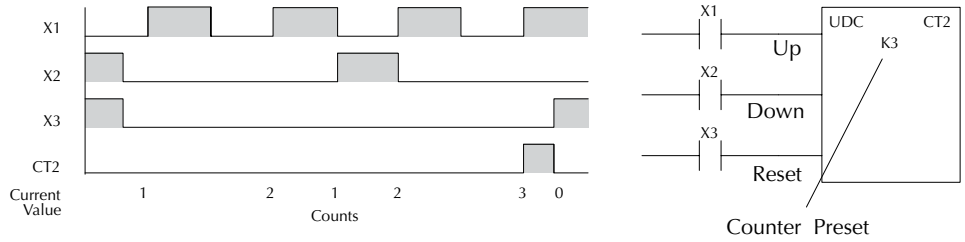
Counters are used to count events . The counters available are up counters, up/down counters, and stage counters (used with RLL*PLUS* programming).

The up counter (CNT) has two inputs, a count input and a reset input. The maximum count value is 9999. The timing diagram below shows the relationship between the counter input, counter reset, associated discrete bit, current value, and counter preset. The CNT counter preset and current value are bothe single word BCD values.

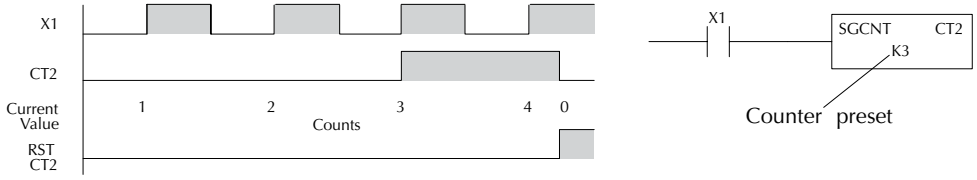


The up down counter (UDC) has three inputs, a count up input, count down input and reset input. The maximum count value is 99999999. The timing diagram below shows the relationship between the counter up and down inputs, counter reset, associated discrete bit, current value, and counter preset. The UDC counter preset and current value are both double word BCD values.

NOTE: *The UDC uses two consecutive V-memory locations for the 8-digit value, therefore, two consecutive timer locations. For example, if UDC CT1 is used, the next available counter number is CT3.*



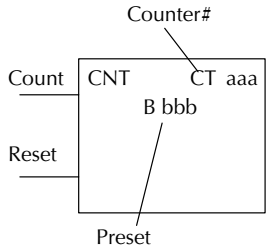
The stage counter (SGCNT) has a count input and is reset by the RST instruction. This instruction is useful when programming using the RLL*PLUS* structured programming. The maximum count value is 9999. The timing diagram below shows the relationship between the counter input, associated discrete bit, current value, counter preset and reset instruction.



Counter (CNT)

DS	Used
HPP	Used

The Counter is a two-input counter that increments when the count input logic transitions from Off to On. When the counter reset input is On, the counter resets to 0. When the current value equals the preset value, the counter status bit comes On and the counter continues to count up to a maximum count of 9999. The maximum value will be held until the counter is reset.



Instruction Specifications

Counter Reference (CTaaa): Specifies the counter number.

Preset Value (Bbbb): Constant value (K) or a V-memory location.

Current Values: Counter current values are accessed by referencing the associated V or CT memory locations.* The V-memory location is the counter location + 1000. For example, the counter current value for CT3 resides in V-memory location V1003.

Discrete Status Bit: The discrete status bit is accessed by referencing the associated CT memory location. It will be On if the value is equal to or greater than the preset value. For example the discrete status bit for counter 2 would be CT2.

***NOTE**: A V-memory preset is required if the ladder program or OIP must change the preset.*

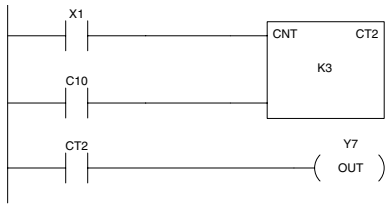
Operand Data Type		DL06 Range	
	A/B	aaa	bbb
Counters	CT	0–177	—
V-memory (preset only)	V	—	400-677 1200–7377 7400–7577 10000-17777
Pointers (preset only)	P	—	400-677 1200–7377 7400–7577* 10000-17777
Constants (preset only)	K	—	0–9999
Counter discrete status bits	CT/V	0–177 or V41140–41147	
Counter current values	V /CT**	1000-1177	

***NOTE:** *May be non-volatile if MOV instruction is used.*
*** With the HPP, both the Counter discrete status bits and current value are accessed with the same data reference. DirectSOFT uses separate references, such as "CT2" for discrete status bit for Counter CT2, and "CTA2" for the current value of Counter CT2.*

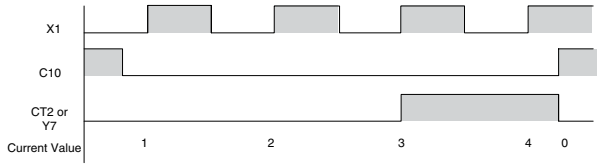
Counter Example Using Discrete Status Bits

In the following example, when X1 makes an Off-to-On transition, counter CT2 will increment by one. When the current value reaches the preset value of 3, the counter status bit CT2 will turn on and energize Y7. When the reset C10 turns on, the counter status bit will turn off and the current value will be 0. The current value for counter CT2 will be held in V-memory location V1002.

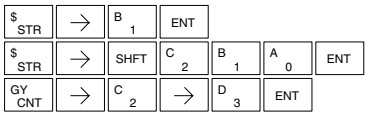
*Direct*SOFT



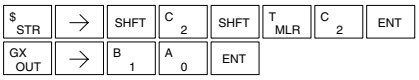
Counting diagram



Handheld Programmer Keystrokes



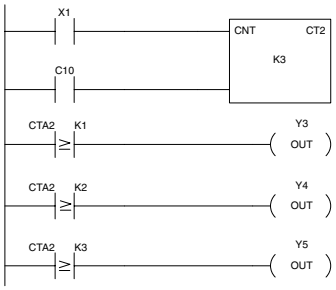
Handheld Programmer Keystrokes (cont)



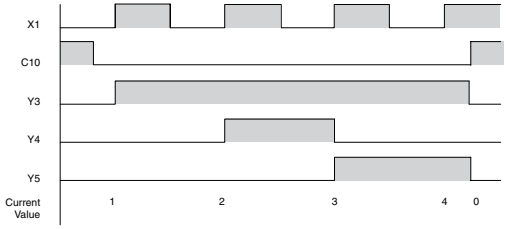
Counter Example Using Comparative Contacts

In the following example, when X1 makes an Off-to-On transition, counter CT2 will increment by one. Comparative contacts are used to energize Y3, Y4, and Y5 at different counts. When the reset C10 turns on, the counter status bit will turn off and the counter current value will be 0, and the comparative contacts will turn off.

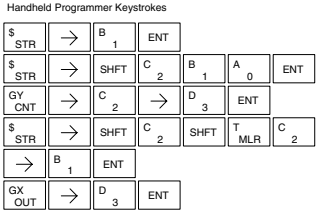
*Direct*SOFT



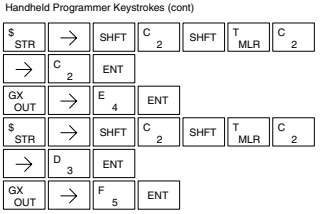
Counting diagram



Handheld Programmer Keystrokes



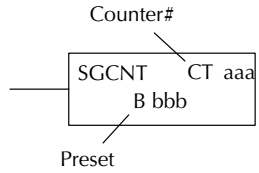
Handheld Programmer Keystrokes (cont)



Stage Counter (SGCNT)

DS	Used
HPP	Used

The Stage Counter is a single input counter that increments when the input logic transitions from off to on. This counter differs from other counters since it will hold its current value until reset using the RST instruction. The Stage Counter is designed for use in RLL*PLUS* programs but can be used in relay ladder logic programs. When the current value equals the preset value, the counter status bit turns on and the counter continues to count up to a maximum count of 9999. The maximum value will be held until the counter is reset.



Instruction Specifications

Counter Reference (CTaaa): Specifies the counter number.

Preset Value (Bbbb): Constant value (K) or a V-memory location.

Current Values: Counter current values are accessed by referencing the associated V or CT memory locations*. The V-memory location is the counter location + 1000. For example, the counter current value for CT3 resides in V-memory location V1003.

Discrete Status Bit: The discrete status bit is accessed by referencing the associated CT memory location. It will be on if the value is equal to or greater than the preset value. For example, the discrete status bit for counter 2 would be CT2.

NOTE: *In using a counter inside a stage, the stage must be active for one scan before the input to the counter makes a 0-1 transition. Otherwise, there is no real transition and the counter will not count.*

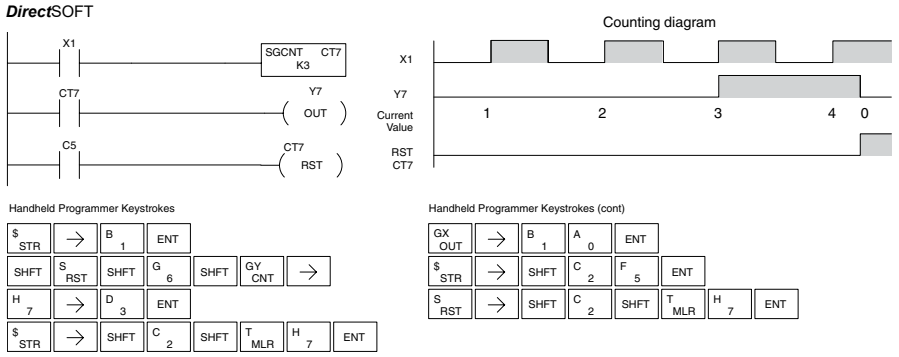
NOTE*: A V-memory preset is required only if the ladder program or an Operator Interface unit must change the preset.*

Operand Data Type		DL06 Range	
	A/B	aaa	bbb
Counters	CT	0–177	—
V-memory (preset only)	V	—	400-677 1200–7377 7400–7577 10000-17777
Pointers (preset only)	P	—	400-677 1200–7377 7400–7577* 10000-17777
Constants (preset only)	K	—	0–9999
Counter discrete status bits	CT/V	0–177 or V41140–41147	
Counter current values	V /CT**	1000-1177	

NOTE: **May be non-volatile if MOV instruction is used.*
*** With the HPP, both the Counter discrete status bits and current value are accessed with the same data reference. DirectSOFT uses separate references, such as "CT2" for discrete status bit for Counter CT2, and "CTA2" for the current value of Counter CT2.*

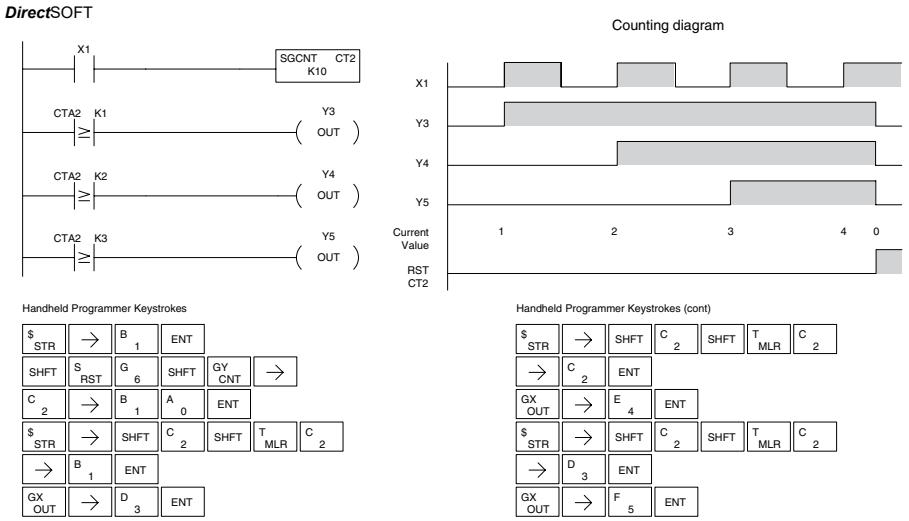
Stage Counter Example Using Discrete Status Bits

In the following example, when X1 makes an off-to-on transition, stage counter CT7 will increment by one. When the current value reaches 3, the counter status bit CT7 will turn on and energize Y7. The counter status bit CT7 will remain on until the counter is reset using the RST instruction. When the counter is reset, the counter status bit will turn off and the counter current value will be 0. The current value for counter CT7 will be held in V-memory location V1007.



Stage Counter Example Using Comparative Contacts

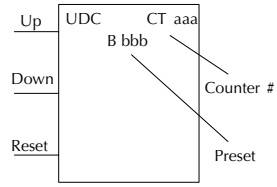
In the following example, when X1 makes an off-to-on transition, counter CT2 will increment by one. Comparative contacts are used to energize Y3, Y4, and Y5 at different counts. Although this is not shown in the example, when the counter is reset using the Reset instruction, the counter status bit will turn off and the current value will be 0. The current value for counter CT2 will be held in V-memory location V1002 (CTA2).



Up Down Counter (UDC)

DS	Used
HPP	Used

This Up/Down Counter counts up on each off to on transition of the Up input and counts down on each off-to-on transition of the Down input. The counter is reset to 0 when the Reset input is on. The count range is 0–99999999. The count input not being used must be off in order for the active count input to function.



Caution: The UDC uses two V-memory locations for the 8 digit current value. This means that the UDC uses two consecutive counter locations. If UDC CT1 is used in the program, the next available counter is CT3.

Instruction Specification

Counter Reference (CTaaa): Specifies the counte number.

Preset Value (Bbbb): Constant value (K) or tw consecutive V-memory locations, in BCD.

Current Values: Current count is a double word value accessed by referencing the associated V or CT memory locations* in BCD. The V-memory location is the counter location + 1000. For example, the counter current value for CT5 resides in V-memory location V1005 and V1006.

Discrete Status Bit: The discrete status bit is accessed by referencing the associated CT memory location. Operating as a "counter done bit" it will be on if the value is equal to or greater than the preset value. For example the discrete status bit for counter 2 would be CT2.

NOTE: *The UDC uses two consecutive V-memory locations for the 8-digit value, therefore two consecutive timer locations. For example, if UDC CT1 is used, the next available counter number is CT3.*

NOTE: *A V-memory preset is required only if the ladder program or an Operator Interface unit must change the preset.*

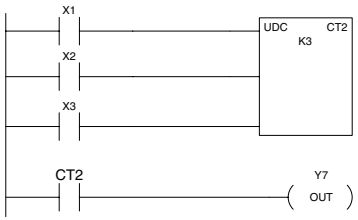
Operand Data Type		DL06 Range	
	A/B	aaa	bbb
Counters	CT	0–177	—
V-memory (preset only)	V	—	400-677 1200–7377 7400–7577 10000-17777
Pointers (preset only)	P	—	400-677 1200–7377* 7400–7577 10000-17777
Constants (preset only)	K	—	0–99999999
Counter discrete status bits	CT/V	0–177 or V41140–41147	
Counter current values	V /CT**	1000-1177	

NOTE: **May be non-volatile if MOV instruction is used.*
*** With the HPP, both the Counter discrete status bits and current value are accessed with the same data reference. DirectSOFT uses separate references, such as "CT2" for discrete status bit for Counter CT2, and "CTA2" for the current value of Counter CT2.*

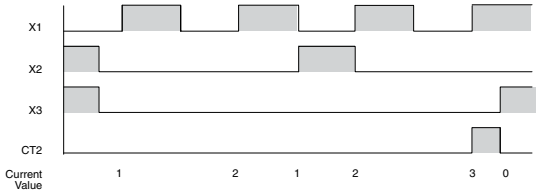
Up / Down Counter Example Using Discrete Status Bits

In the following example, if X2 and X3 are off, the counter will increment by one when X1 toggles from Off to On . If X1 and X3 are off, the counter will decrement by one when X2 toggles from Off to On. When the count value reaches the preset value of 3, the counter status bit will turn on. When the reset X3 turns on, the counter status bit will turn off and the current value will be 0.

DirectSOFT



Counting Diagram



Handheld Programmer Keystrokes

$ STR	→	B 1	ENT				
$ STR	→	C 2	ENT				
$ STR	→	D 3	ENT				
SHFT	U ISG	D 3	C 2	→	C 2		

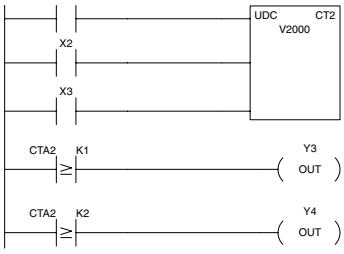
Handheld Programmer Keystrokes (cont)

→	D 3	ENT					
$ STR	→	SHFT	C 2	SHFT	T MLR	C 2	ENT
GX OUT	→	B 1	A 0	ENT			

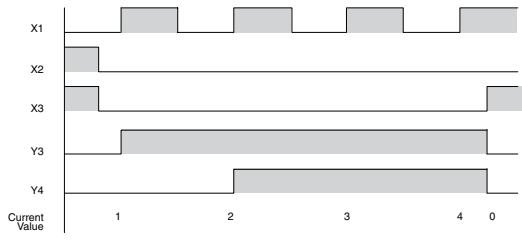
Up / Down Counter Example Using Comparative Contacts

In the following example, when X1 makes an off-to-on transition, counter CT2 will increment by one. Comparative contacts are used to energize Y3 and Y4 at different counts. When the reset (X3) turns on, the counter status bit will turn off, the current value will be 0, and the comparative contacts will turn off.

DirectSOFT



Counting Diagram



Handheld Programmer Keystrokes

$ STR	→	B 1	ENT				
$ STR	→	C 2	ENT				
$ STR	→	D 3	ENT				
SHFT	U ISG	D 3	C 2	→	C 2	→	
SHFT	V AND	C 2	A 0	A 0	A 0	ENT	
$ STR	→	SHFT	C 2	SHFT	T MLR	C 2	

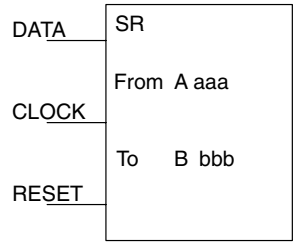
Handheld Programmer Keystrokes (cont)

→	B 1	ENT					
GX OUT	→	D 3	ENT				
$ STR	→	SHFT	C 2	SHFT	T MLR	C 2	
→	C 2	ENT					
GX OUT	→	E 4	ENT				

Shift Register (SR)

DS	Used
HPP	Used

The Shift Register instruction shifts data through a predefined number of control relays. The control ranges in the shift register block must start at the beginning of an 8 bit boundary and must use 8-bit blocks.

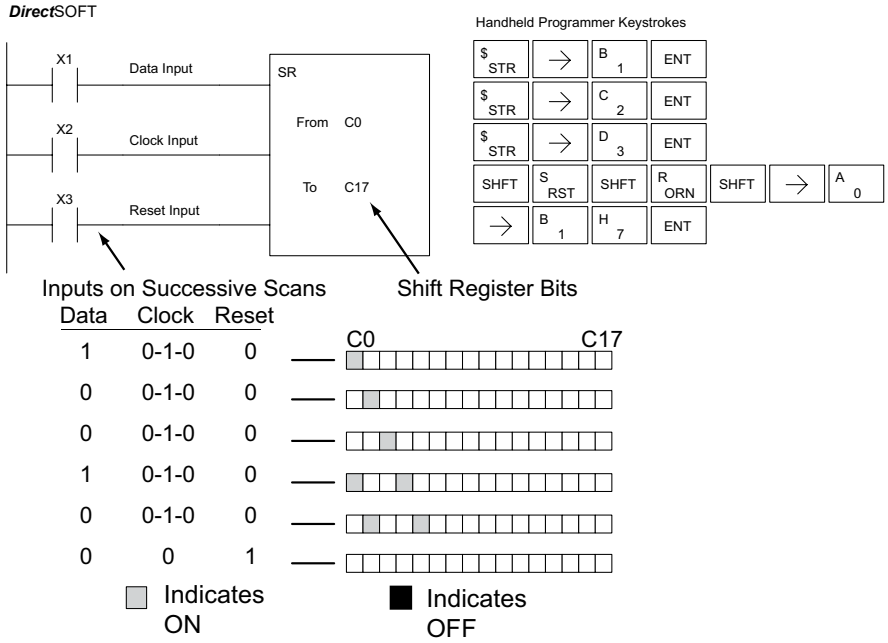


The Shift Register has three contacts.

- Data — determines the value (1 or 0) that will enter the register
- Clock — shifts the bits one position on each low to high transition
- Reset —resets the Shift Register to all zeros.

With each off-to-on transition of the clock input, the bits which make up the shift register block are shifted by one bit position and the status of the data input is placed into the starting bit position in the shift register. The direction of the shift depends on the entry in the From and To fields. From C0 to C17 would define a block of sixteen bits to be shifted from left to right. From C17 to C0 would define a block of sixteen bits to be shifted from right to left. The maximum size of the shift register block depends on the number of available control relays. The minimum block size is 8 control relays.

Operand Data Type	DL06 Range	
	A/B	aaa	bbb
Control Relay	C	0–1777	0–1777



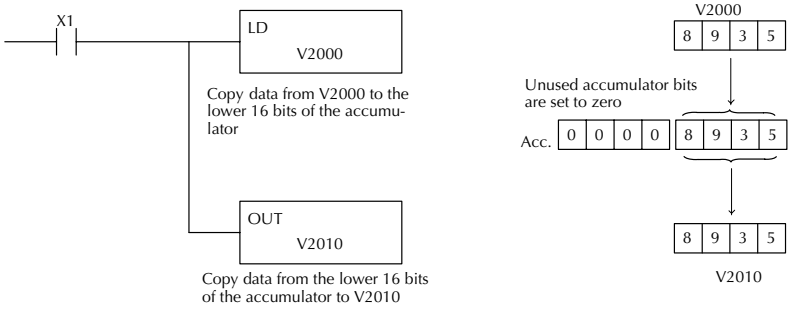
Accumulator/Stack Load and Output Data Instructions

Using the Accumulator

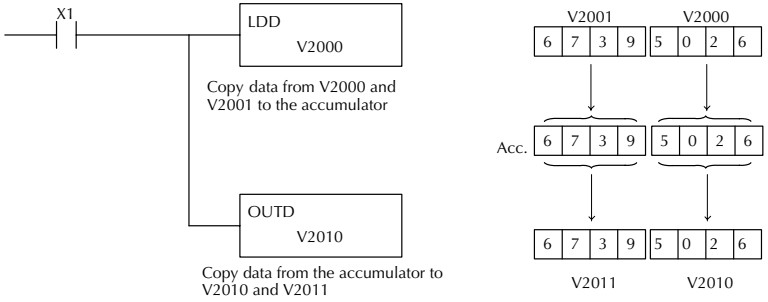
The accumulator in the DL06 internal CPUs is a 32-bit register which is used as a temporary storage location for data that is being copied or manipulated in some manner. For example, you have to use the accumulator to perform math operations such as add, subtract, multiply, etc. Since there are 32 bits, you can use up to an 8-digit BCD number. The accumulator is reset to 0 at the end of every CPU scan.

Copying Data to the Accumulator

The Load and Out instructions and their variations are used to copy data from a V-memory location to the accumulator, or to copy data from the accumulator to V-memory. The following example copies data from V-memory location V2000 to V-memory location V2010.

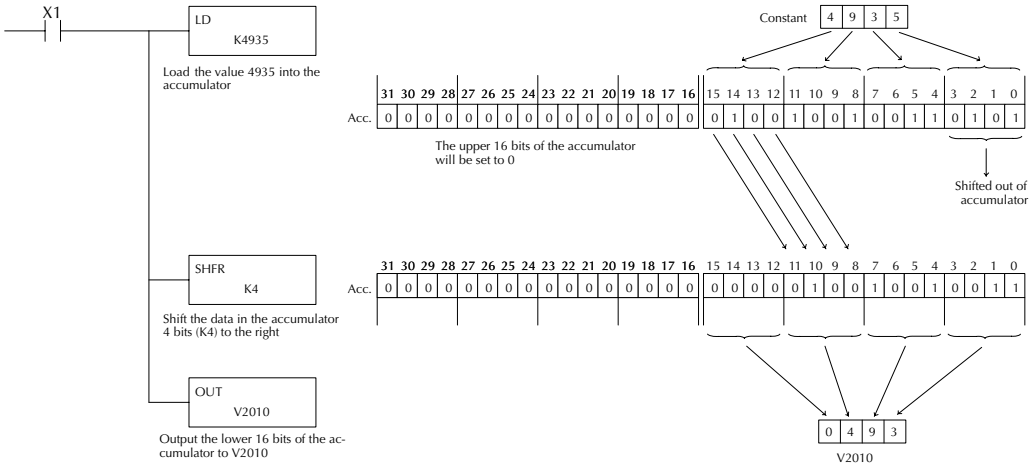


Since the accumulator is 32 bits and V-memory locations are 16 bits, the Load Double and Out Double (or variations thereof) use two consecutive V-memory locations or 8 digit BCD constants to copy data either to the accumulator from a V-memory address or from a V-memory address to the accumulator. For example, if you wanted to copy data from V2000 and V2001 to V2010 and V2011 the most efficient way to perform this function would be as follows:



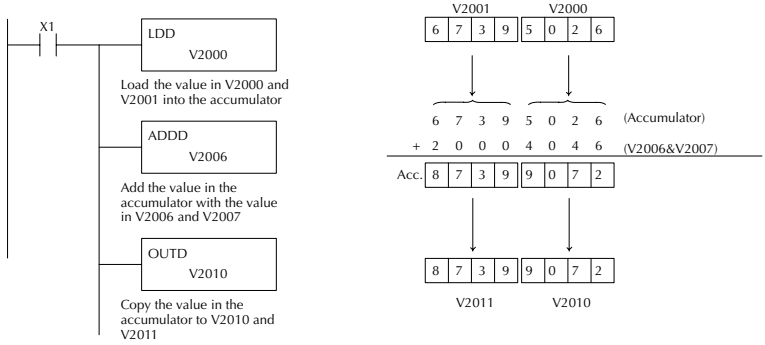
Changing the Accumulator Data

Instructions that manipulate data also use the accumulator. The result of the manipulated data resides in the accumulator. The data that was being manipulated is cleared from the accumulator. The following example loads the constant value 4935 into the accumulator, shifts the data right 4 bits, and outputs the result to V2010.



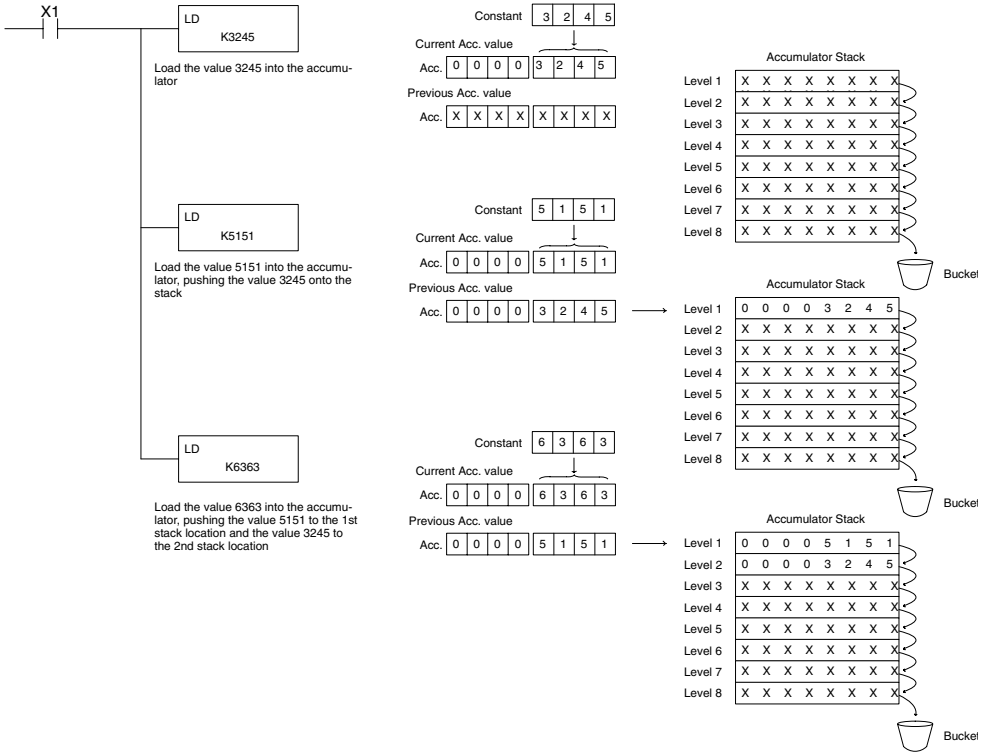
Some of the data manipulation instructions use 32 bits. They use two consecutive V-memory locations or an 8 digit BCD constant to manipulate data in the accumulator.

In the following example, when X1 is on, the value in V2000 and V2001 will be loaded into the accumulator using the Load Double instruction. The value in the accumulator is added with the value in V2006 and V2007 using the Add Double instruction. The value in the accumulator is copied to V2010 and V2011 using the Out Double instruction.

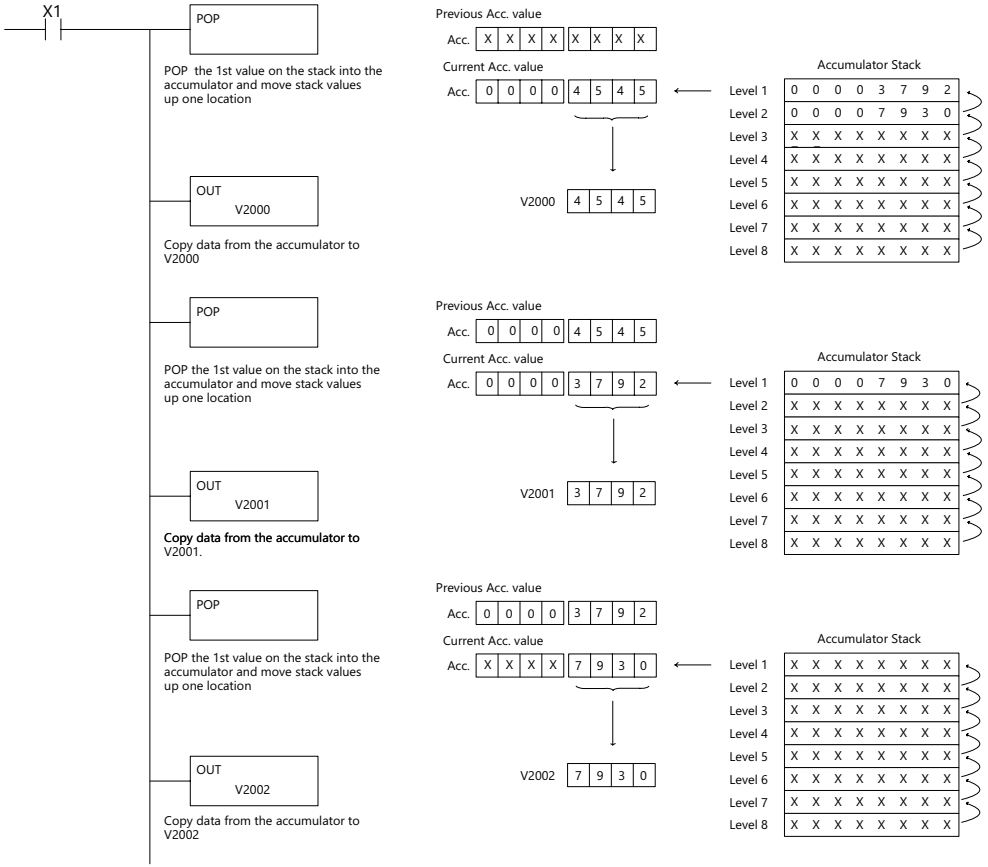


Using the Accumulator Stack

The accumulator stack is used for instructions that require more than one parameter to execute a function or for user-defined functionality. The accumulator stack is used when more than one Load instruction is executed without the use of an Out instruction. The first load instruction in the scan places a value into the accumulator. Every Load instruction thereafter without the use of an Out instruction places a value into the accumulator and the value that was in the accumulator is placed onto the accumulator stack. The Out instruction nullifies the previous load instruction and does not place the value that was in the accumulator onto the accumulator stack when the next load instruction is executed. Every time a value is placed onto the accumulator stack the other values in the stack are pushed down one location. The accumulator is eight levels deep (eight 32-bit registers). If there is a value in the eighth location when a new value is placed onto the stack, the value in the eighth location is pushed off the stack and cannot be recovered.



The POP instruction rotates values upward through the stack into the accumulator. When a POP is executed, the value which was in the accumulator is cleared and the value that was on top of the stack is in the accumulator. The values in the stack are shifted up one position in the stack.

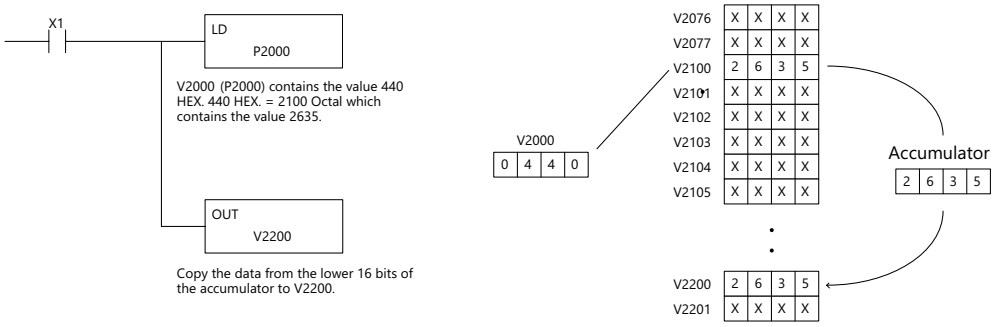


Using Pointers

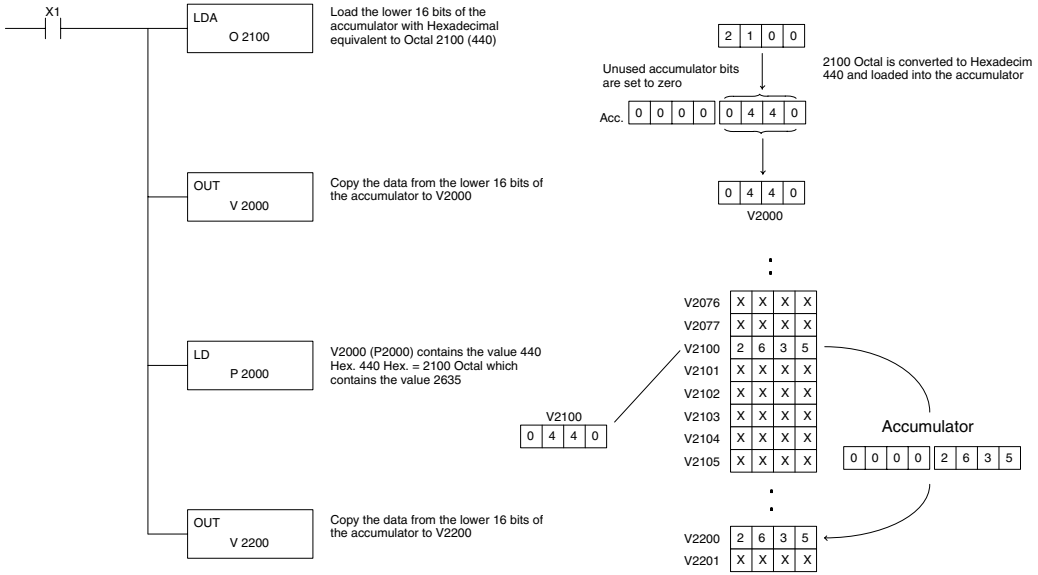
Many of the DL06 series instructions will allow V-memory pointers as operands (commonly known as indirect addressing). Pointers allow instructions to obtain data from V-memory locations referenced by the pointer value.

***NOTE**: DL06 V-memory addressing is in octal. However, the pointers reference a V-memory location with values viewed as HEX. Use the Load Address (LDA) instruction to move an address into the pointer location. This instruction performs the Octal to Hexadecimal conversion automatically.*

In the following example we are using a pointer operand in a Load instruction. V-memory location 2000 is being used as the pointer location. V2000 contains the value 440 which the CPU views as the Hex equivalent of the Octal address V-memory location V2100. The CPU will copy the data from V2100, which (in this example) contains the value 2635, into the lower word of the accumulator.



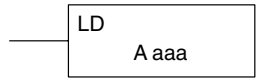
The following example is identical to the one above, with one exception. The LDA (Load Address) instruction automatically converts the Octal address to Hex.



Load (LD)

DS	Used
HPP	Used

The Load instruction is a 16 bit instruction that loads the value (Aaaa), which is either a V-memory location or a 4 digit constant, into the lower 16 bits of the accumulator. The upper 16 bits of the accumulator are set to 0.

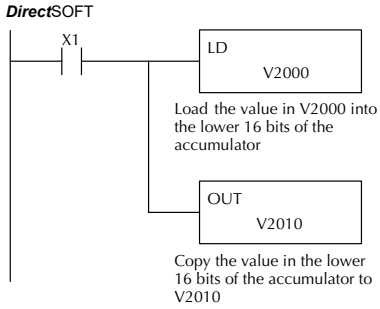


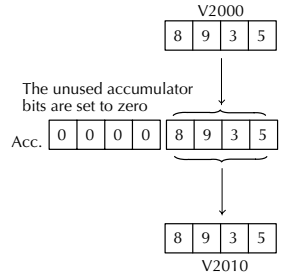
Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	K	0–FFFF

Discrete Bit Flags	Description
SP53	On when the pointer is outside of the available range.
SP70	On anytime the value in the accumulator is negative.
SP76	On when any instruction loads a value of zero into the accumulator.

***NOTE**: Two consecutive Load instructions will place the value of the first load instruction onto the accumulator stack.*

In the following example, when X1 is on, the value in V2000 will be loaded into the accumulator and output to V2010.





Handheld Programmer Keystrokes

$ STR	→	B 1	X SET					
SHFT	L ANDST	D 3	→					
C 2	A 0	A 0	A 0	ENT				
GX OUT	→	SHFT	V AND	C 2	A 0	B 1	A 0	ENT

Load Double (LDD)

DS	Used
HPP	Used

The Load Double instruction is a 32-bit instruction that loads tl value (Aaaa), which is either two consecutive V-memory locations an 8 digit constant value, into the accumulator.

LDD
A aaa

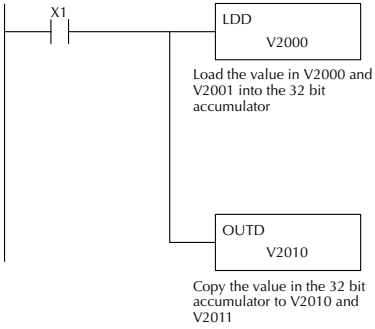
Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	K	0–FFFFFFFF

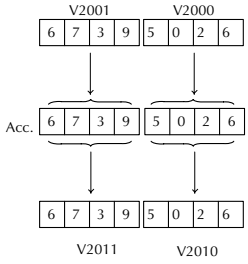
Discrete Bit Flags	Description
SP53	On when the pointer is outside of the available range.
SP70	On anytime the value in the accumulator is negative.
SP76	On when any instruction loads a value of zero into the accumulator.

***NOTE**: Two consecutive Load instructions will place the value of the first load instruction onto the accumulator stack.*

In the following example, when X1 is on, the 32-bit value in V2000 and V2001 will be loaded into the accumulator and output to V2010 and V2011.

***Direct*SOFT**





Handheld Programmer Keystrokes

$ STR	→	B 1	ENT	
SHFT	L ANDST	D 3	D 3	→
C 2	A 0	A 0	A 0	ENT
GX OUT	SHFT	D 3	→	
C 2	A 0	B 1	A 0	ENT

Load Formatted (LDF)

DS	Used
HPP	Used

The Load Formatted instruction loads 1–32 consecutive bits from discrete memory locations into the accumulator. The instruction requires a starting location (Aaaa) and the number of bits (Kbbb) to be loaded. Unused accumulator bit locations are set to zero.

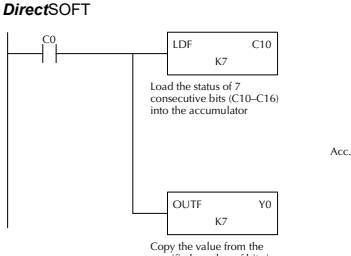
Operand Data Type		DL06 Range	
	A	aaa	bbb
Inputs	X	0–777	—
Outputs	Y	0–777	—
Control Relays	C	0–1777	—
Stage Bits	S	0–1777	—
Timer Bits	T	0–377	—
Counter Bits	CT	0–177	—
Special Relays	SP	0–777	—
Constant	K	—	1–32

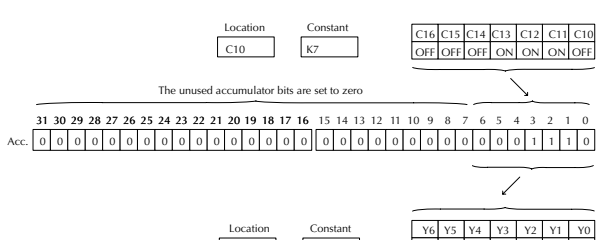
Discrete Bit Flags	Description
SP70	On anytime the value in the accumulator is negative.
SP76	On when any instruction loads a value of zero into the accumulator.

***NOTE**: Two consecutive Load instructions will place the value of the first load instruction onto the accumulator stack.*

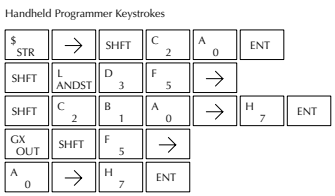
In the following example, when C0 is on, the binary pattern of C10–C16 (7 bits) will be loaded into the accumulator using the Load Formatted instruction. The lower 7 bits of the accumulator are output to Y0–Y6 using the Out Formatted instruction.

***Direct*SOFT**





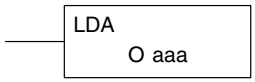
Handheld Programmer Keystrokes



Load Address (LDA)

DS	Used
HPP	Used

The Load Address instruction is a 16-bit instruction. It converts any octal value or address to the HEX equivalent value and loads the HEX value into the accumulator. This instruction is useful when an address parameter is required, since all addresses for the DL06 system are in octal.



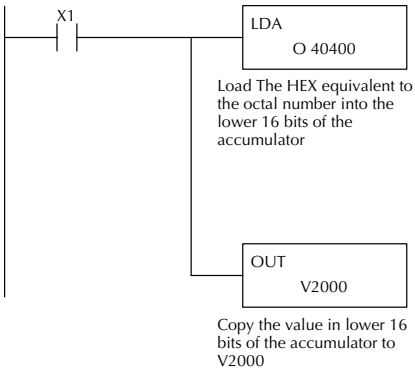
Operand Data Type		DL06 Range
		aaa
Octal Address	O	See memory map

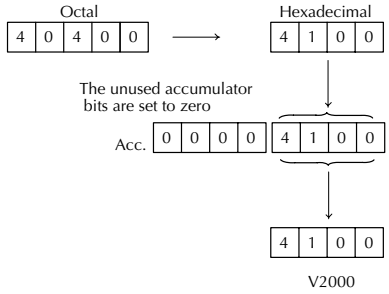
Discrete Bit Flags	Description
SP70	On anytime the value in the accumulator is negative.
SP76	On when any instruction loads a value of zero into the accumulator.

***NOTE**: Two consecutive Load instructions will place the value of the first load instruction onto the accumulator stack.*

In the following example, when X1 is on, the octal number 40400 will be converted to a HEX 4100 and loaded into the accumulator using the Load Address instruction. The value in the lower 16 bits of the accumulator is copied to V2000 using the Out instruction.

DirectSOFT





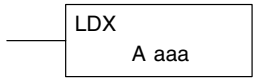
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT					
SHFT	L ANDST	D 3	A 0	→				
E 4	A 0	E 4	A 0	A 0	ENT			
GX OUT	→	SHFT	V AND	C 2	A 0	A 0	A 0	ENT

Load Accumulator Indexed (LDX)

DS	Used
HPP	Used

Load Accumulator Indexed is a 16-bit instruction that specifies a source address (V-memory) which will be offset by the value in the first stack location. This instruction interprets the value in the first stack location as HEX. The value in the offset address (source address + offset) is loaded into the lower 16 bits of the accumulator. The upper 16 bits of the accumulator are set to 0.



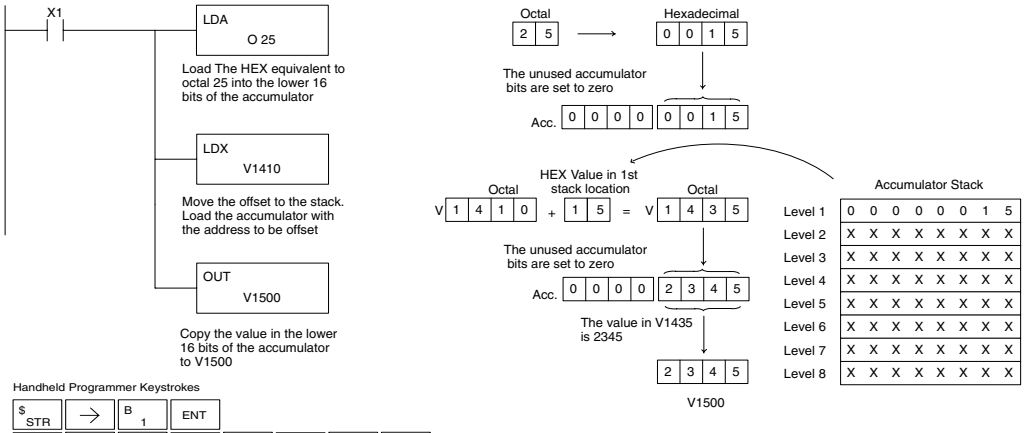
Helpful Hint: — The Load Address instruction can be used to convert an octal address to a HEX address and load the value into the accumulator.

Operand Data Type		DL06 Range	
A		aaa	aaa
V-memory	V	See memory map	See memory map
Pointer	P	See memory map	See memory map

Discrete Bit Flags	Description
SP53	On when the pointer is outside of the available range.
SP70	On anytime the value in the accumulator is negative.
SP76	On when any instruction loads a value of zero into the accumulator.

***NOTE**: Two consecutive Load instructions will place the value of the first load instruction onto the accumulator stack.*

In the following example, when X1 is on, the HEX equivalent for octal 25 will be loaded into the accumulator (this value will be placed on the stack when the Load Accumulator Indexed instruction is executed). V-memory location V1410 will be added to the value in the first level of the stack and the value in this location (V1435 = 2345) is loaded into the lower 16 bits of the accumulator using the Load Accumulator Indexed instruction. The value in the lower 16 bits of the accumulator is output to V1500 using the Out instruction.



Handheld Programmer Keystrokes

$ STR	→	B 1	ENT					
SHFT	L ANDST	D 3	A 0	→	C 2	F 5	ENT	
SHFT	L ANDST	D 3	X SET	→	B 1	E 4	B 1	A 0	ENT
GX OUT	→	PREV	PREV	PREV	B 1	F 5	A 0	A 0	ENT

Load Accumulator Indexed from Data Constants (LDSX)

DS	Used
HPP	Used

The Load Accumulator Indexed from Data Constants is a 16-bit instruction. The instruction specifies a Data Label Area (DLBL) where numerical or ASCII constants are stored. This value will be loaded into the lower 16 bits.

LDSX
K aaa

The LDSX instruction uses the value in the first level of the accumulator stack as an offset to determine which numerical or ASCII constant within the Data Label Area will be loaded into the accumulator. The LDSX instruction interprets the value in the first level of the accumulator stack as a HEX value.

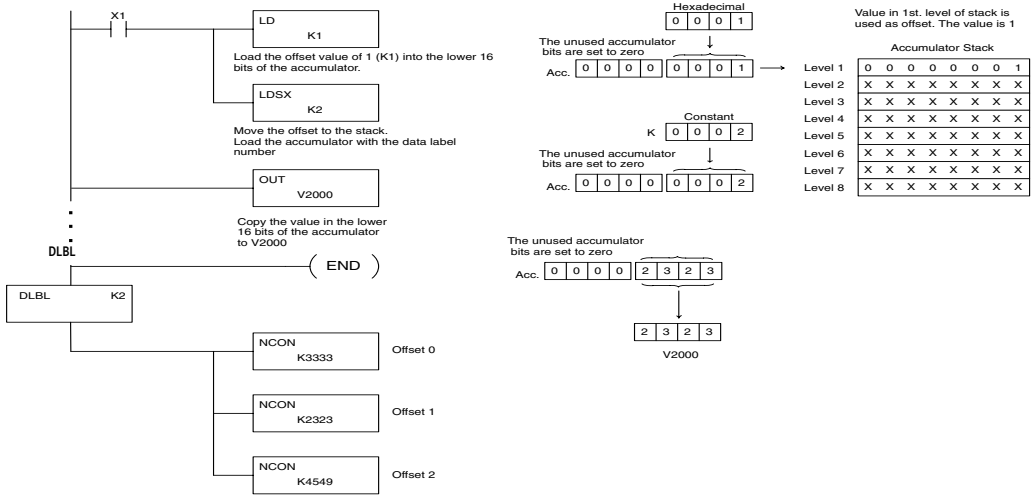
Helpful Hint: — The Load Address instruction can be used to convert octal to HEX and load the value into the accumulator.

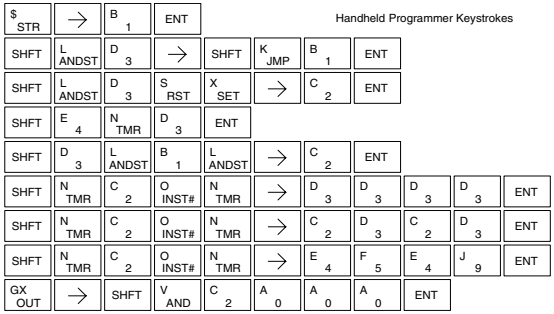
Operand Data Type		DL06 Range
		aaa
Constant	K	1-FFFF

Discrete Bit Flags	Description
SP53	On when the pointer is outside of the available range.
SP70	On anytime the value in the accumulator is negative.
SP76	On when any instruction loads a value of zero into the accumulator.

***NOTE**: Two consecutive Load instructions will place the value of the first load instruction onto the accumulator stack.*

In the following example when X1 is on, the offset of 1 is loaded into the accumulator. This value will be placed into the first level of the accumulator stack when the LDSX instruction is executed. The LDSX instruction specifies the Data Label (DLBL K2) where the numerical constant(s) are located in the program and loads the constant value, indicated by the offset in the stack, into the lower 16 bits of the accumulator.





Load Real Number (LDR)

DS	Used
HPP	N/A

The Load Real Number instruction loads a real number containe in two consecutive V-memory locations, or an 8-digit consta into the accumulator.

LDR
A aaa

Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Real Constant	R	$-3.402823E^{+38}$ to + $-3.402823E^{+38}$

Discrete Bit Flags	Description
SP70	On anytime the value in the accumulator is negative.
SP76	On when any instruction loads a value of zero into the accumulator.

*Direct*SOFT allows you to enter real numbers directly, by using the leading "R" to indicate a real number entry. You can enter a constant such as Pi, shown in the example to the right. To enter negative numbers, use a minus (–) after the "R".

LDR
R3.14159

For very large numbers or very small numbers, you can use exponential notation. The number to the right is 5.3 million. The OUTD instruction stores it in V1400 and V1401.

LDR
R5.3E6
OUTD
V1400

These real numbers are in the IEEE 32-bit floating point format, so they occupy two V-memory locations, regardless of how big or small the number may be! If you view a stored real number in hex, binary, or even BCD, the number shown will be very difficult to decipher. Just like all other number types, you must keep track of real number locations in memory, so they can be read with the proper instructions later.

The previous example above stored a real number in V1400 and V1401. Suppose that now we want to retrieve that number. Just use the Load Real with the V data type, as shown to the right. Next we could perform real math on it, or convert it to a binary number.

LDR
V1400

Out (OUT)

DS	Used
HPP	Used

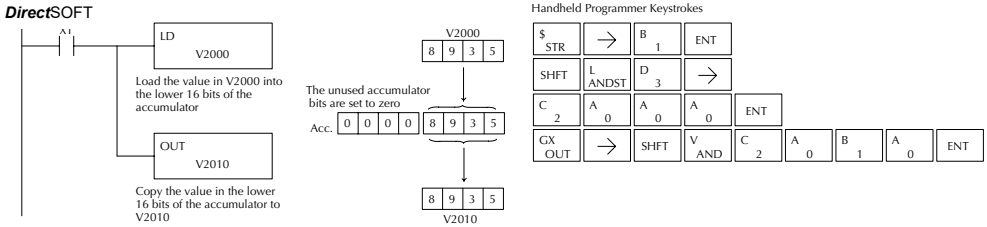
The Out instruction is a 16-bit instruction that copies the value in the lower 16 bits of the accumulator to a specified V-memory location (Aaaa).

OUT
A aaa

Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map

Discrete Bit Flags	Description
SP53	On if CPU cannot solve the logic.

In the following example, when X1 is on, the value in V2000 will be loaded into the lower 16 bits of the accumulator using the Load instruction. The value in the lower 16 bits of the accumulator are copied to V2010 using the OUT instruction.



Out Double (OUTD)

DS	Used
HPP	Used

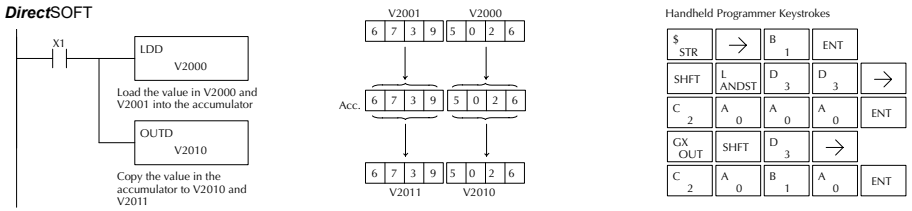
The Out Double instruction is a 32 bit instruction that copies th value in the accumulator to two consecutive V-memory locations at specified starting location (Aaaa).

OUTD
A aaa

Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map

Discrete Bit Flags	Description
SP53	On if CPU cannot solve the logic.

In the following example, when X1 is on, the 32-bit value in V2000 and V2001 will be loaded into the accumulator using the Load Double instruction. The value in the accumulator is output to V2010 and V2011 using the OUTD instruction.



Out Formatted (OUTF)

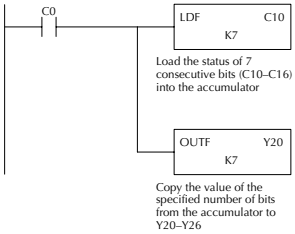
DS	Used
HPP	Used

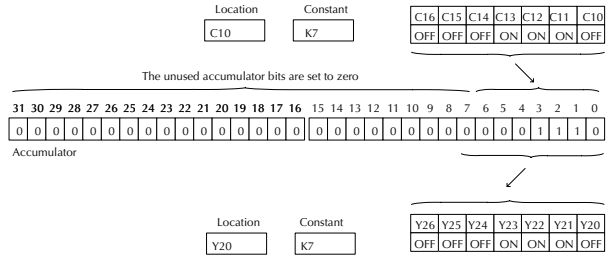
The Out Formatted instruction outputs 1–32 bits from the accumulator to the specified discrete memory locations. The instruction requires a starting location (Aaaa) for the destination and the number of bits (Kbbb) to be output.

Operand Data Type	DL06 Range		
	A	aaa	bbb
Inputs	X	0–777	—
Outputs	Y	0–777	—
Control Relays	C	0–1777	—
Constant	K	—	1–32

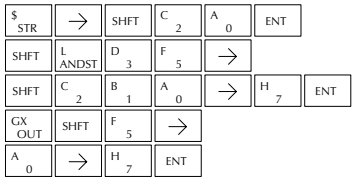
In the following example, when C0 is on, the binary pattern of C10–C16 (7 bits) will be loaded into the accumulator using the Load Formatted instruction. The lower 7 bits of the accumulator are output to Y0–Y6 using the OUTF instruction.

DirectSOFT





Handheld Programmer Keystrokes



Pop (POP)

DS	Used
HPP	Used

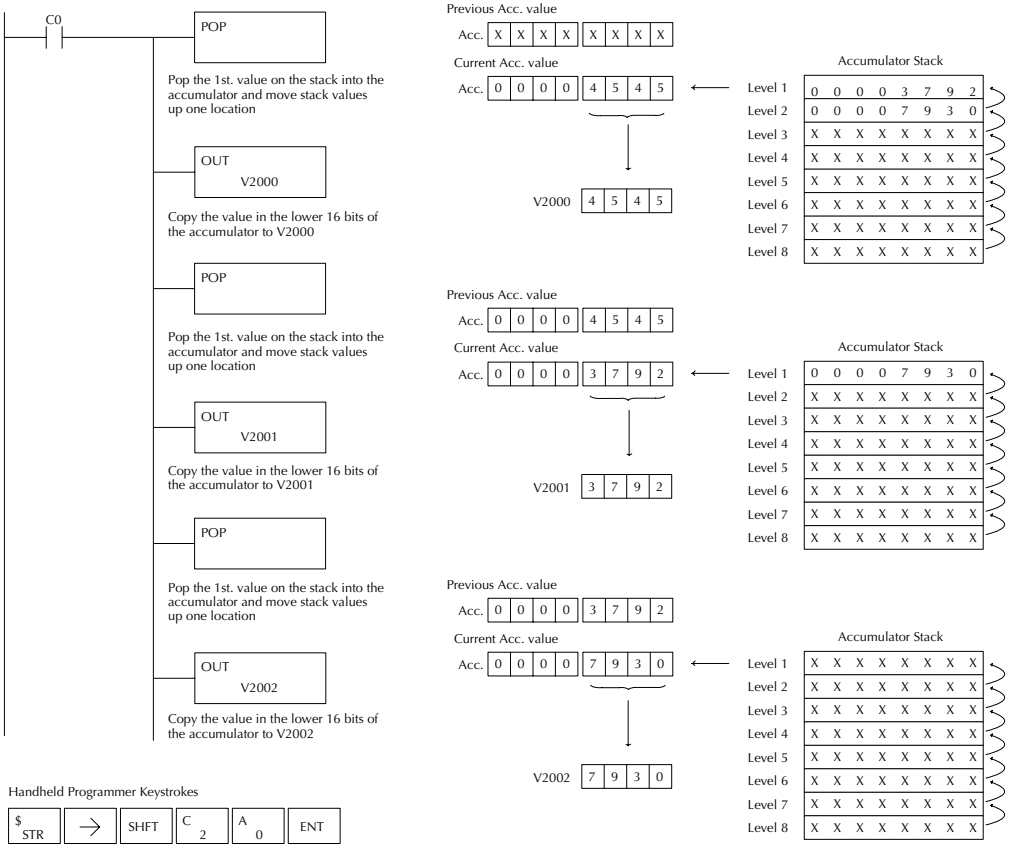
The Pop instruction moves the value from the first level of the accumulator stack (32 bits) to the accumulator and shifts each value in the stack up one level.

Discrete Bit Flags	Description
SP63	ON when the result of the instruction causes the value in the accumulator to be zero.

Pop Instruction (cont'd)

In the example below, when C0 is on, the value 4545 that was on top of the stack is moved into the accumulator using the Pop instruction The value is output to V2000 using the OUT instruction. The next Pop moves the value 3792 into the accumulator and outputs the value to V2001. The last Pop moves the value 7930 into the accumulator and outputs the value to V2002. Please note if the value in the stack were greater than 16 bits (4 digits) the OUTD instruction would be used and 2 V-memory locations for each OUTD must be allocated.

DirectSOFT



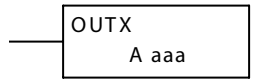
Handheld Programmer Keystrokes

$ STR	→	SHFT	C 2	A 0	ENT			
SHFT	P CV	SHFT	O INST#	P CV	ENT			
GX OUT	→	SHFT	V AND	C 2	A 0	A 0	A 0	ENT
SHFT	P CV	SHFT	O INST#	P CV	ENT			
GX OUT	→	SHFT	V AND	C 2	A 0	A 0	B 1	ENT
SHFT	P CV	SHFT	O INST#	P CV	ENT			
GX OUT	→	SHFT	V AND	C 2	A 0	A 0	C 2	ENT

Out Indexed (OUTX)

DS	Used
HPP	Used

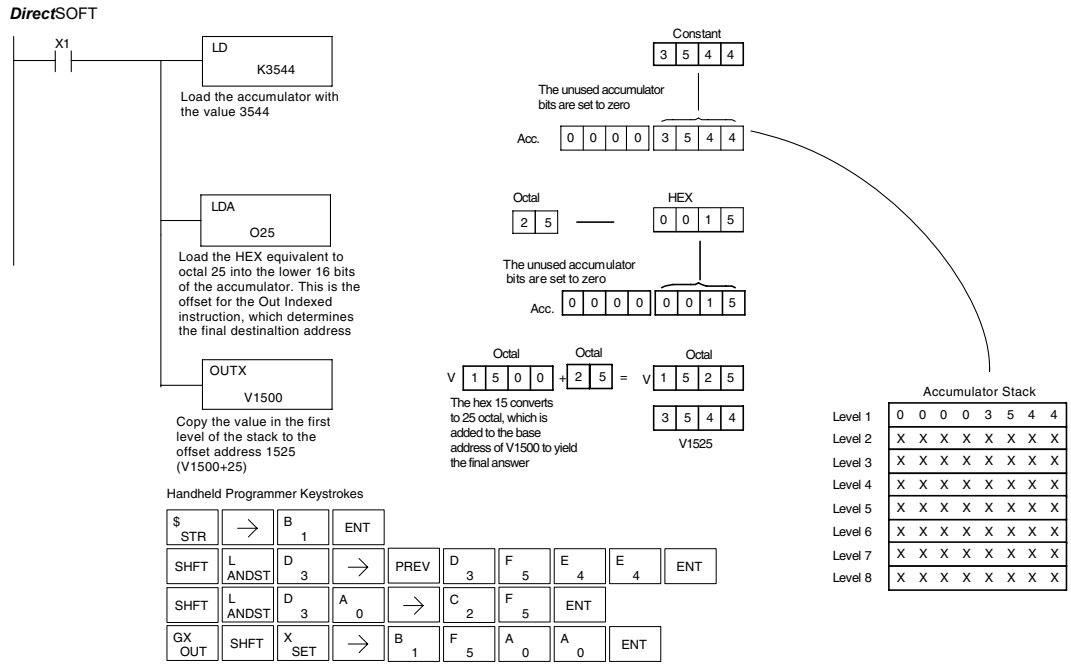
The OUTX instruction is a 16 bit instruction. It copies a 16 bit or 4 digit value from the first level of the accumulator stack to a source address offset by the value in the accumulator(V-memory + offset).This instruction interprets the offset value as a HEX number. The upper 16 bits of the accumulator are set to zero.



Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map

Discrete Bit Flags	Description
SP53	On if CPU cannot solve the logic.

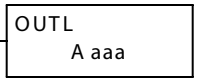
In the following example, when X1 is on, the constant value 3544 is loaded into the accumulator. This is the value that will be output to the specified offset V-memory location (V1525). The value 3544 will be placed onto the stack when the LDA instruction is executed. Remember, two consecutive LD instructions places the value of the first load instruction onto the stack. The LDA instruction converts octal 25 to HEX 15 and places the value in the accumulator. The OUTX instruction outputs the value 3544 which resides in the first level of the accumulator stack to V1525.



Out Least (OUTL)

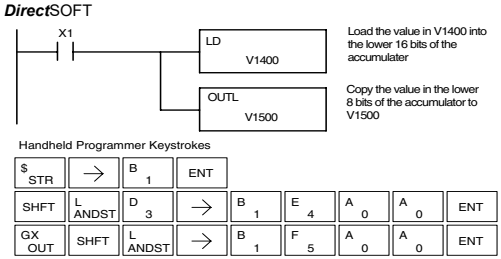
DS	Used
HPP	Used

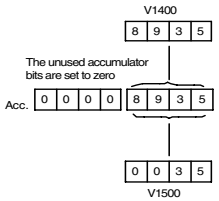
The OUTL instruction copies the value in the lower eight bits of the accumulator to the lower eight bits of the specified V-memory location (i.e., it copies the low byte of the low word of the accumulator).



Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map

In the following example, when X1 is on, the value in V1400 will be loaded into the lower 16 bits of the accumulator using the LD instruction. The value in the lower 8 bits of the accumulator is copied to V1500 using the OUTL instruction.

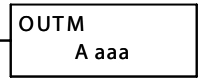




Out Most (OUTM)

DS	Used
HPP	Used

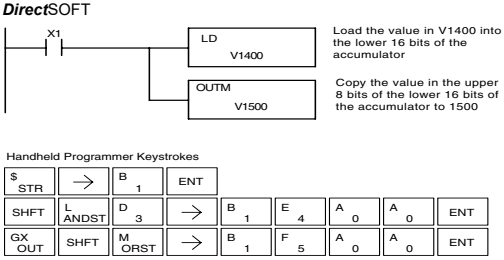
The OUTM instruction copies the value in the upper eight bits of the lower sixteen bits of the accumulator to the upper eight bits of the specified V-memory location (i.e., it copies the high byte of the low word of the accumulator).

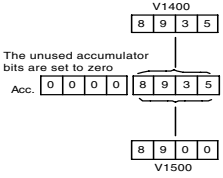


In the following example, when X1 is on, the value in V1400 will be loaded into the lower 16

Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map

bits of the accumulator using the LD instruction. The value in the upper 8 bits of the lower 16 bits of the accumulator is copied to V1500 using the OUTM instruction.



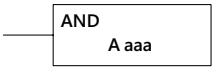


Logical Instructions (Accumulator)

And (AND logical)

DS	Used
HPP	Used

The AND instruction is a 16-bit instruction that logically ANDs the value in the lower 16 bits of the accumulator with a specified V-memory location (Aaaa). The result resides in the accumulator. The discrete status flag indicates if the result of the AND is zero.



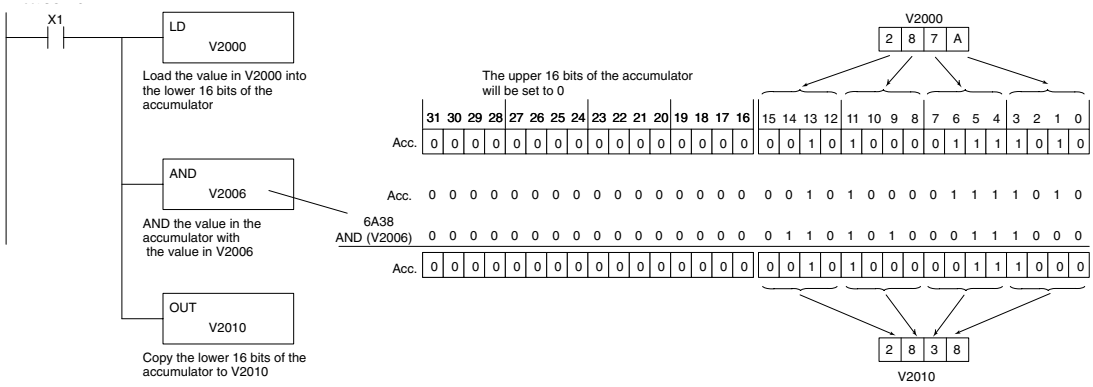
Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map

Discrete Bit Flags	Description
SP63	ON if the result in the accumulator is zero.
SP70	ON when the value loaded into the accumulator is zero.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V2000 will be loaded into the accumulator using the LD instruction. The value in the accumulator is ANDed with the value in V2006 using the AND instruction. The value in the lower 16 bits of the accumulator is output to V2010 using the OUT instruction.

DirectSOFT



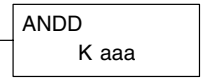
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT					
SHFT	L ANDST	D 3	→	C 2	A 0	A 0	A 0	ENT
V AND	→	SHFT	V AND	C 2	A 0	A 0	G 6	ENT
GX OUT	→	SHFT	V AND	C 2	A 0	B 1	A 0	ENT

And Double (ANDD)

DS	Used
HPP	Used

ANDD is a 32-bit instruction that logically ANDs the value in the accumulator with two consecutive V-memory locations or an 8 digit (max.) constant value (Aaaa). The result resides in the accumulator. Discrete status flags indicate if the result of the ANDD is zero or a negative number (the most significant bit is on).

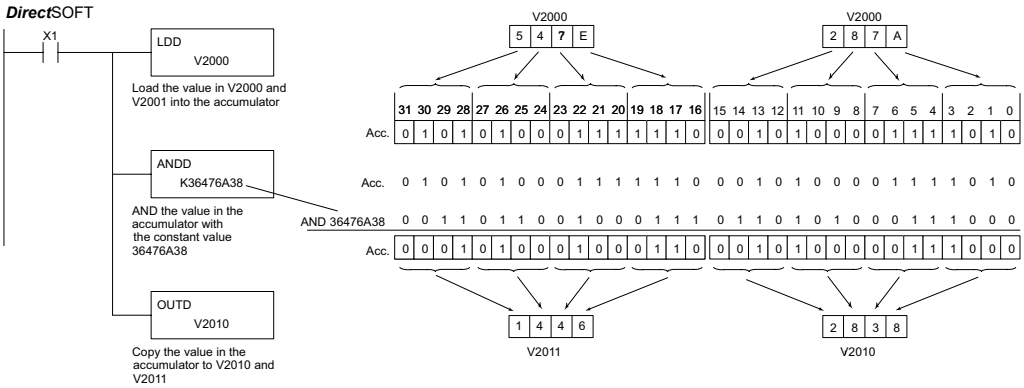


Operand Data Type		DL06 Range
		aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	K	0–FFFFFFFF

Discrete Bit Flags	Description
SP63	ON if the result in the accumulator is zero.
SP70	ON if the result in the accumulator is negative

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V2000 and V2001 will be loaded into the accumulator using the LDD instruction. The value in the accumulator is ANDed with 36476A38 using the ANDD instruction. The value in the accumulator is output to V2010 and V2011 using the OUTD instruction.



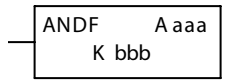
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT																
SHFT	L ANDST	D 3	D 3	→	C 2	A 0	A 0	A 0	ENT										
V AND	SHFT	D 3	→	SHFT	K JMP	D 3	G 6	E 4	H 7	G 6	SHFT	A 0	SHFT	D 3	I 8	ENT			
GX OUT	SHFT	D 3	→	C 2	A 0	B 1	A 0	ENT											

And Formatted (ANDF)

DS	Used
HPP	Used

The ANDF instruction logically ANDs the binary value in the accumulator with a specified range of discrete memory bits (1–32). The instruction requires a starting location (Aaaa) and number of bits (Kbbb) to be ANDed. Discrete status flags indicate if the result is zero or a negative number (the most significant bit =1).

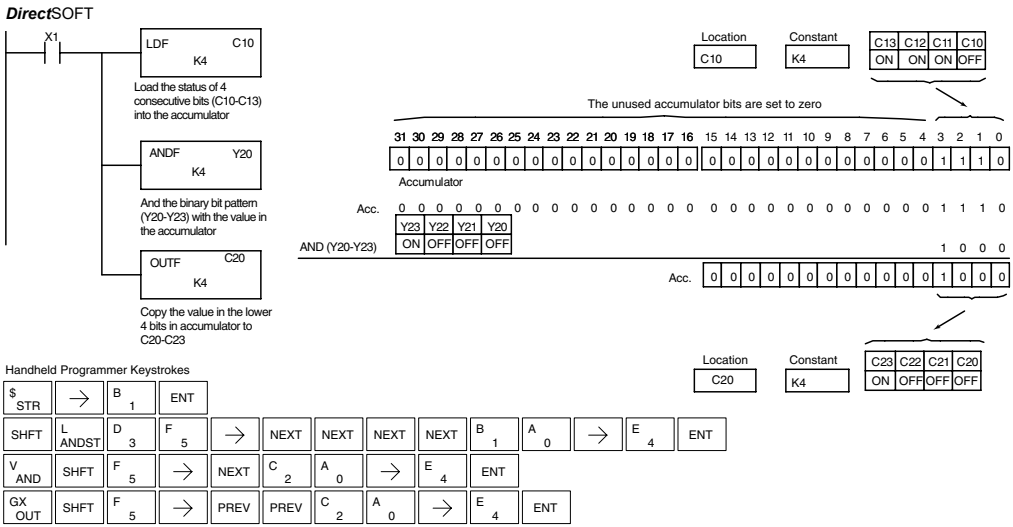


Operand Data Type	DL06 Range		
	B	aaa	bbb
Inputs	X	0-777	-
Outputs	Y	0-777	-
Control Relays	C	0-1777	-
Stage Bits	S	0-1777	-
Timer Bits	T	0-377	-
Counter Bits	CT	177	-
Special Relays	SP	0-777	-
Constant	K	-	1-32

Discrete Bit Flags	Description
SP63	ON if the result in the accumulator is zero.
SP70	ON if the result in the accumulator is negative

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

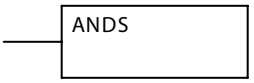
In the following example, when X1 is on, the LDF instruction loads C10–C13 (4 binary bits) into the accumulator. The accumulator contents is logically ANDed with the bit pattern from Y20–Y23 using the ANDF instruction. The OUTF instruction outputs the accumulator's lower four bits to C20–C23.



And with Stack (ANDS)

DS	Used
HPP	Used

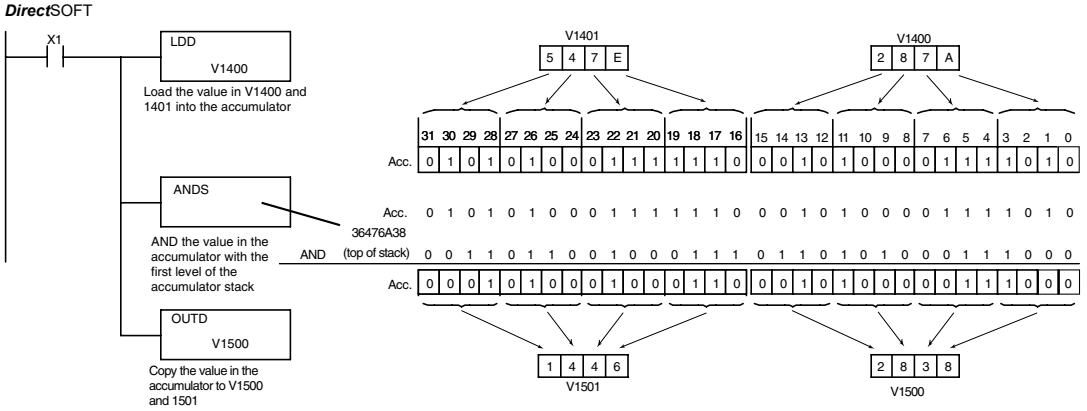
The ANDS instruction is a 32-bit instruction that logically ANDs the value in the accumulator with the first level of the accumulator stack. The result resides in the accumulator. The value in the first level of the accumulator stack is removed from the stack and all values are moved up one level. Discrete status flags indicate if the result of the ANDS is zero or a negative number (the most significant bit is on).



Discrete Bit Flags	Description
SP63	ON if the result in the accumulator is zero.
SP70	ON if the result in the accumulator is negative

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the binary value in the accumulator will be ANDed with the binary value in the first level or the accumulator stack. The result resides in the accumulator. The 32-bit value is then output to V1500 and V1501.



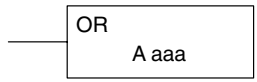
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT					
SHFT	L ANDST	D 3	D 3	→	B 1	E 4	A 0	A 0	ENT
V AND	SHFT	S RST	ENT					
GX OUT	SHFT	D 3	→	B 1	F 5	A 0	A 0	ENT

Or (OR)

DS	Used
HPP	Used

The Or instruction is a 16-bit instruction that logically ORs the value in the lower 16 bits of the accumulator with a specified V-memory location (Aaaa). The result resides in the accumulator. The discrete status flag indicates if the result of the OR is zero.

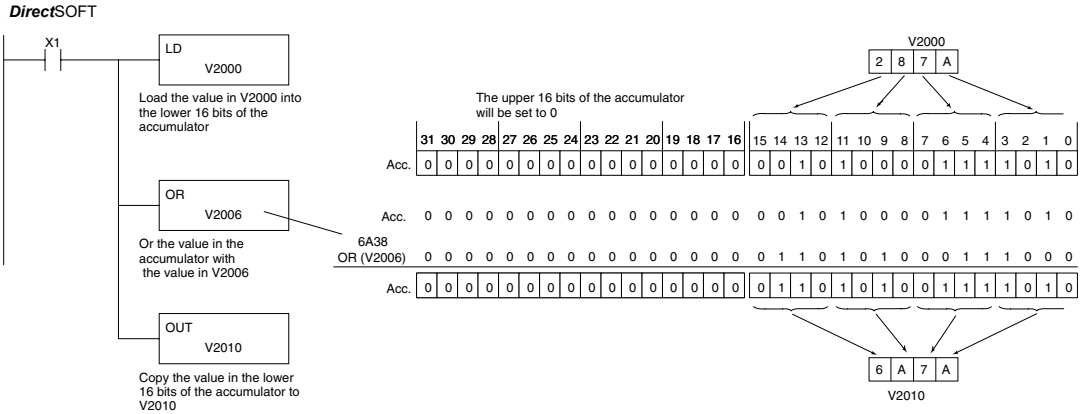


Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map

Discrete Bit Flags	Description
SP63	ON if the result in the accumulator is zero.
SP70	ON if the result in the accumulator is negative

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V2000 will be loaded into the accumulator using the Load instruction. The value in the accumulator is OR'd with V2006 using the OR instruction. The value in the lower 16 bits of the accumulator is output to V2010 using the Out instruction.



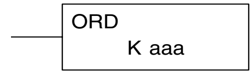
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT					
SHFT	L ANDST	D 3	→	C 2	A 0	A 0	A 0	ENT
Q OR	→	SHFT	V AND	C 2	A 0	A 0	G 6	ENT
GX OUT	→	SHFT	V AND	C 2	A 0	B 1	A 0	ENT

Or Double (ORD)

DS	Used
HPP	Used

ORD is a 32-bit instruction that logically ORs the value in the accumulator with the value (Aaaa), which is either two consecutive V-memory locations or an 8-digit (max.) constant value. The result resides in the accumulator. Discrete status flags indicate if the result of the ORD is zero or a negative number (the most significant bit is on).

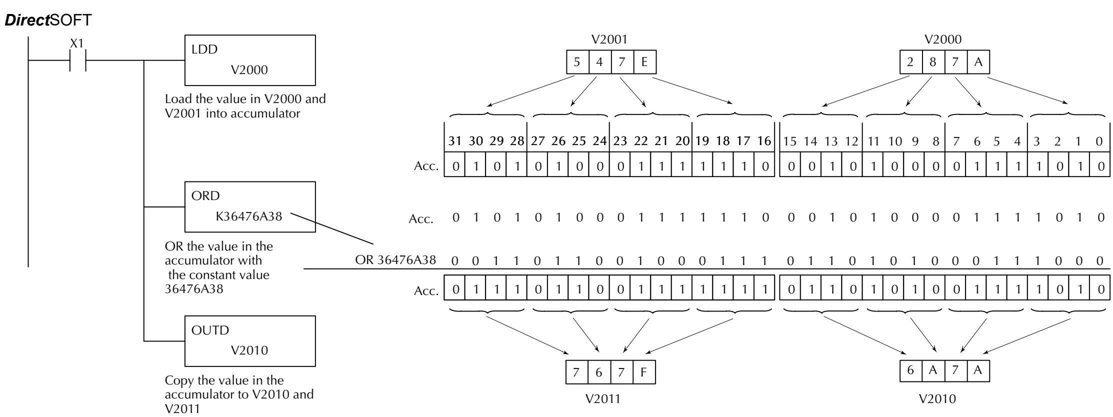


Operand Data Type		DL06 Range
		aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	K	0–FFFFFFFF

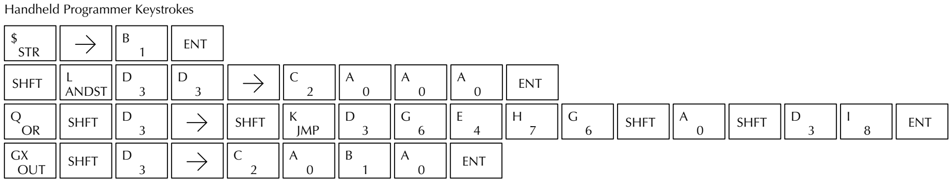
Discrete Bit Flags	Description
SP63	ON if the result in the accumulator is zero.
SP70	ON if the result in the accumulator is negative

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V2000 and V2001 will be loaded into the accumulator using the LDD instruction. The value in the accumulator is OR'd with 36476A38 using the ORD instruction. The value in the accumulator is output to V2010 and V2011 using the OUTD instruction.



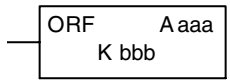
Handheld Programmer Keystrokes



Or Formatted (ORF)

DS	Used
HPP	Used

The ORF instruction logically ORs the binary value in the accumulator and a specified range of discrete bits (1–32). The instruction requires a starting location (Aaaa) and the number of bits (Kbbb) to be OR'd. Discrete status flags indicate if the result is zero or negative (the most significant bit =1).

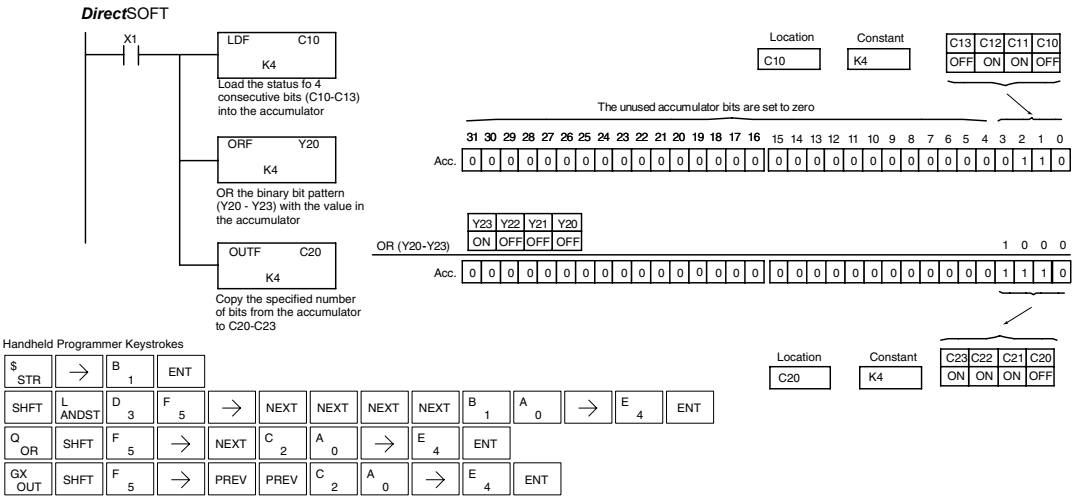


Operand Data Type		DL06 Range	
	A/B	aaa	bbb
Inputs	X	0-777	--
Outputs	Y	0-777	--
Control Relays	C	0-1777	--
Stage Bits	S	0-1777	--
Timer Bits	T	0-377	--
Counter Bits	CT	0-177	--
Special Relays	SP	0-777	--
Constant	K	-	1-32

Discrete Bit Flags	Description
SP63	ON if the result in the accumulator is zero.
SP70	ON if the result in the accumulator is negative

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the LDF instruction loads C10–C13 (4 binary bits) into the accumulator. The ORF instruction logically ORs the accumulator contents with Y20–Y23 bit pattern. The ORF instruction outputs the accumulator's lower four bits to C20–C23.



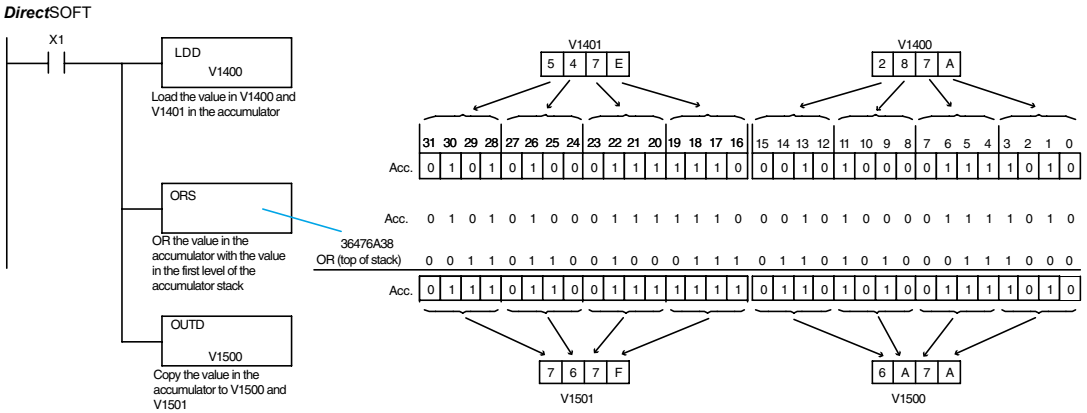
Or with Stack (ORS)

DS	Used
HPP	Used

The ORS instruction is a 32-bit instruction that logically ORs the value in the accumulator with the first level of the accumulator stack. The result resides in the accumulator. The value in the first level of the accumulator stack is removed from the stack and all values are moved up one level. Discrete status flags indicate if the result of the ORS is zero or a negative number (the most significant bit is on).

Discrete Bit Flags	Description
SP63	ON if the result in the accumulator is zero.
SP70	ON if the result in the accumulator is negative.

In the following example when X1 is on, the binary value in the accumulator will be OR'd with the binary value in the first level of the stack. The result resides in the accumulator.



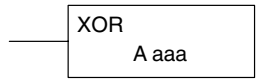
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	D 3	→	B 1	E 4	A 0	A 0	ENT
Q OR	SHFT	S RST	ENT						
GX OUT	SHFT	D 3	→	B 1	F 5	A 0	A 0	ENT	

Exclusive Or (XOR)

DS	Used
HPP	Used

The XOR instruction is a 16-bit instruction that performs an exclusive OR of the value in the lower 16 bits of the accumulator and a specified V-memory location (Aaaa). The result resides in the in the accumulator. The discrete status flag indicates if the result of the XOR is zero.



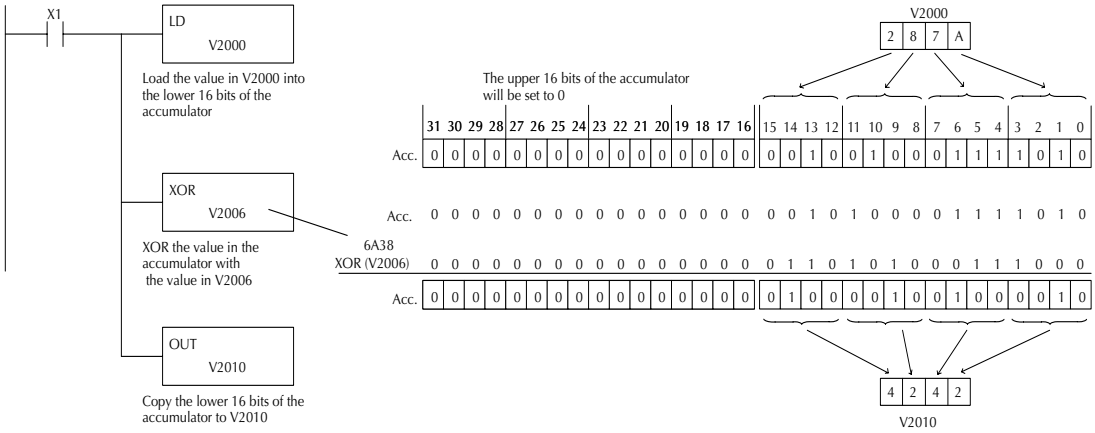
Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map

Discrete Bit Flags	Description
SP63	ON if the result in the accumulator is zero.
SP70	ON if the result in the accumulator is negative.

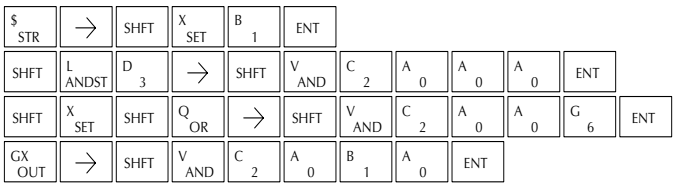
***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V2000 will be loaded into the accumulator using the LD instruction. The value in the accumulator is exclusive OR'd with V2006 using the XOR instruction. The value in the lower 16 bits of the accumulator is output to V2010 using the OUT instruction.

DirectSOFT



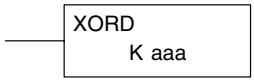
Handheld Programmer Keystrokes



Exclusive Or Double (XORD)

DS	Used
HPP	Used

The XORD is a 32-bit instruction that performs an exclusive OR of the value in the accumulator and the value (Aaaa), which is either two consecutive V-memory locations or an 8 digit (max.) constant. The result resides in the accumulator. Discrete status flags indicate if the result of the XORD is zero or a negative number (the most significant bit is on).

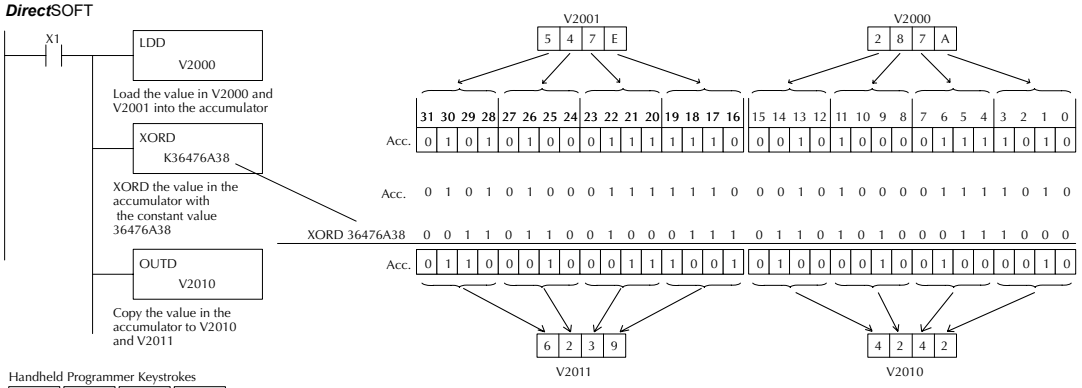


Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	K	0–FFFFFFFF

Discrete Bit Flags	Description
SP63	ON if the result in the accumulator is zero.
SP70	ON if the result in the accumulator is negative

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V2000 and V2001 will be loaded into the accumulator using the LDD instruction. The value in the accumulator is exclusively OR'd with 36476A38 using the XORD instruction. The value in the accumulator is output to V2010 and V2011 using the OUTD instruction.



Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	D 3	→	C 2	A 0	A 0	A 0	ENT
SHFT	X SET	Q OR	SHFT	D 3	→	SHFT	K JMP		
D 3	G 6	E 4	H 7	G 6	SHFT	A 0	SHFT	D 3	I 8	ENT
GX OUT	SHFT	D 3	→	C 2	A 0	B 1	A 0	ENT		

Exclusive Or Formatted (XORF)

DS	Used
HPP	Used

The XORF instruction performs an exclusive OR of the binary value in the accumulator and a specified range of discrete memory bits (1–32).

```
XORF    Aaaa
K bbb
```

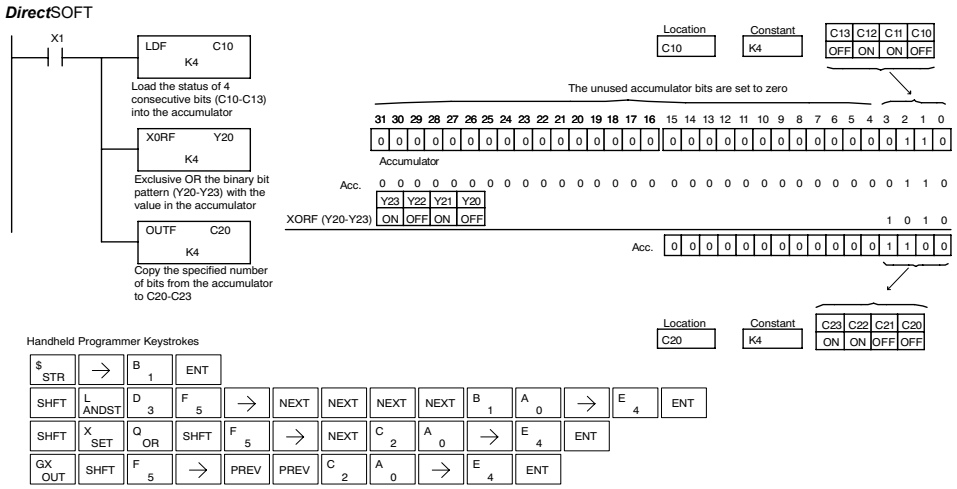
The instruction requires a starting location (Aaaa) and the number of bits (Bbbb) to be exclusive OR'd. Discrete status flags indicate if the result of the XORF is zero or negative (the most significant bit =1).

Operand Data Type		DL06 Range	
	A/B	aaa	bbb
Inputs	X	0-777	-
Outputs	Y	0-777	-
Control Relays	C	0-1777	-
Stage Bits	S	0-1777	-
Timer Bits	T	0-377	-
Counter Bits	CT	177	-
Special Relays	SP	0-777	-
Constant	K	-	1-32

Discrete Bit Flags	Description
SP63	ON if the result in the accumulator is zero.
SP70	ON if the result in the accumulator is negative

NOTE: *Status flags are valid only until another instruction uses the same flag.*

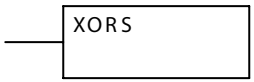
In the following example, when X1 is on, the binary pattern of C10–C13 (4 bits) will be loaded into the accumulator using the LDF instruction. The value in the accumulator will be logically exclusive OR'd with the bit pattern from Y20–Y23 using the XORF instruction. The value in the lower 4 bits of the accumulator is output to C20–C23 using the OUTF instruction.



Exclusive Or with Stack (XORS)

DS	Used
HPP	Used

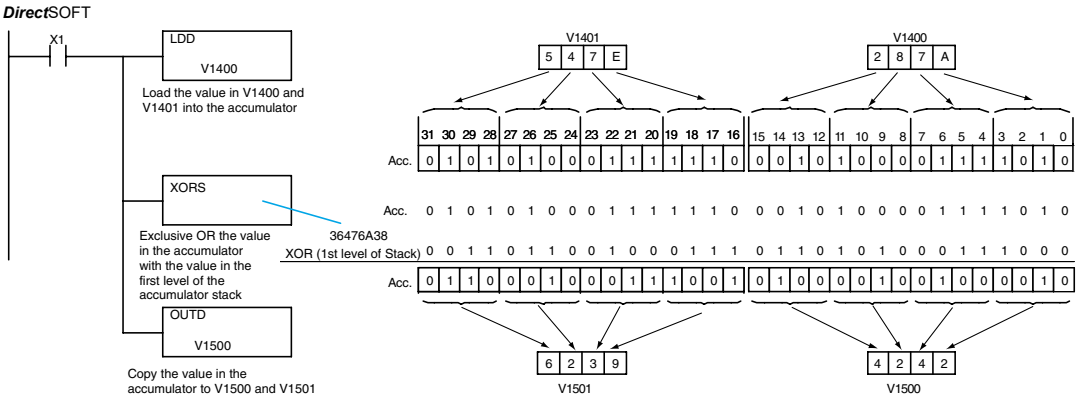
The XORS instruction is a 32-bit instruction that performs an Exclusive Or of the value in the accumulator with the first level of the accumulator stack. The result resides in the accumulator. The value in the first level of the accumulator stack is removed from the stack and all values are moved up one level. Discrete status flags indicate if the result of the XORS is zero or a negative number (the most significant bit is on).



Discrete Bit Flags	Description
SP63	ON if the result in the accumulator is zero.
SP70	ON if the result in the accumulator is negative

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V1400 and V1401 will be loaded into the accumulator using the LDD instruction. The binary value in the accumulator will be exclusively OR'd with 36476A38 using the XORS instruction. The value in the accumulator is output to V1500 and V1501 using the OUTD instruction.



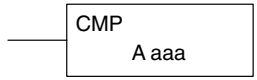
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT					
SHFT	L ANDST	D 3	D 3	→	B 1	E 4	A 0	A 0	ENT
SHFT	X SET	Q OR	SHFT	S RST	ENT			
GX OUT	SHFT	D 3	→	B 1	F 5	A 0	A 0	ENT

Compare (CMP)

DS	Used
HPP	Used

The CMP instruction is a 16-bit instruction that compares the value in the lower 16 bits of the accumulator with the value in a specified V-memory location (Aaaa). The corresponding status flag will be turned on indicating the result of the comparison. The data format for this instruction is BCD/Hex, Decimal and Binary.

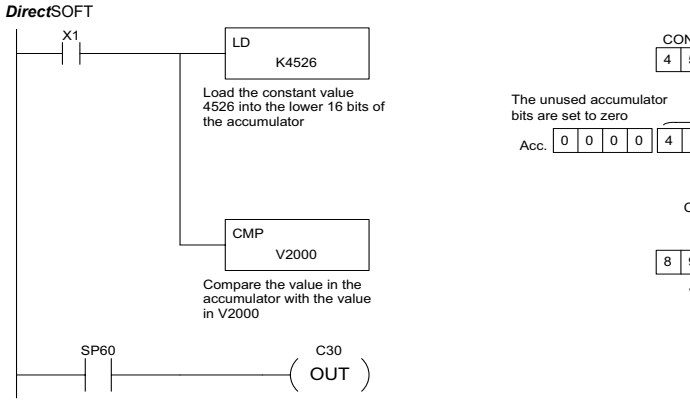


Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map

Discrete Bit Flags	Description
SP60	On when the value in the accumulator is less than the instruction value.
SP61	On when the value in the accumulator is equal to the instruction value.
SP62	On when the value in the accumulator is greater than the instruction value.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example when X1 is on, the constant 4526 will be loaded into the lower 16 bits of the accumulator using the LD instruction. The value in the accumulator is compared with the value in V2000 using the CMP instruction. The corresponding discrete status flag will be turned on indicating the result of the comparison. In this example, if the value in the accumulator is less than the value specified in the CMP instruction, SP60 will turn on, energizing C30.



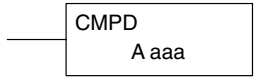
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT							
SHFT	L ANDST	D 3	→	SHFT	K JMP	E 4	F 5	C 2	G 6	ENT
SHFT	C 2	SHFT	M ORST	P CV	→	C 2	A 0	A 0	A 0	ENT
$ STR	→	SHFT	SP STRN	G 6	A 0	ENT				
GX OUT	→	SHFT	C 2	D 3	A 0	ENT				

Compare Double (CMPD)

DS	Used
HPP	Used

The Compare Double instruction is a 32–bit instruction that compares the value in the accumulator with the value (Aaaa), which is either two consecutive V-memory locations or an 8–digit (max.) constant. The corresponding status flag will be turned on indicating the result of the comparison. The data format for this instruction is BCD/Hex, Decimal and Binary.

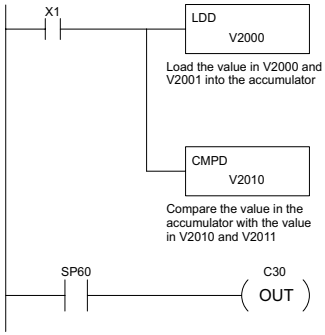


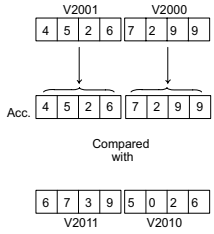
Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	K	0–FFFFFFFF

Discrete Bit Flags	Description
SP60	On when the value in the accumulator is less than the instruction value.
SP61	On when the value in the accumulator is equal to the instruction value.
SP62	On when the value in the accumulator is greater than the instruction value.

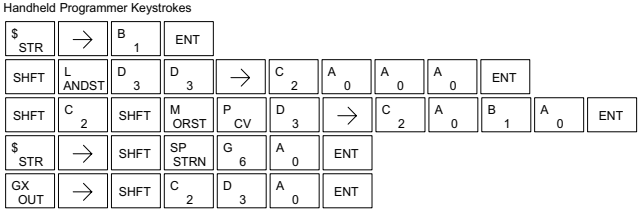
***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V2000 and V2001 will be loaded into the accumulator using the Load Double instruction. The value in the accumulator is compared with the value in V2010 and V2011 using the CMPD instruction. The corresponding discrete status flag will be turned on, indicating the result of the comparison. In this example, if the value in the accumulator is less than the value specified in the Compare instruction, SP60 will turn on energizing C30.





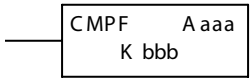
Handheld Programmer Keystrokes



Compare Formatted (CMPF)

DS	Used
HPP	Used

The Compare Formatted instruction compares the value in the accumulator with a specified number of discrete locations (1–32). The instruction requires a starting location (Aaaa) and the number of bits (Kbbb) to be compared. The corresponding status flag will be turned on, indicating the result of the comparison. The data format for this instruction is BCD/Hex, Decimal and Binary.



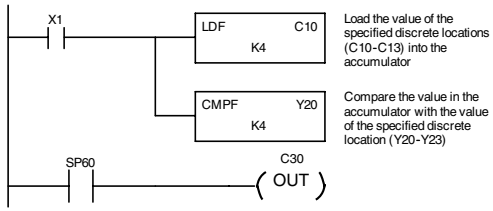
Operand Data Type		DL06 Range	
	A/B	aaa	bbb
Inputs	X	0-777	-
Outputs	Y	0-777	-
Control Relays	C	0-1777	-
Stage Bits	S	0-1777	-
Timer Bits	T	0-377	-
Counter Bits	CT	0-177	-
Special Relays	SP	0-777	-
Constant	K	-	1-32

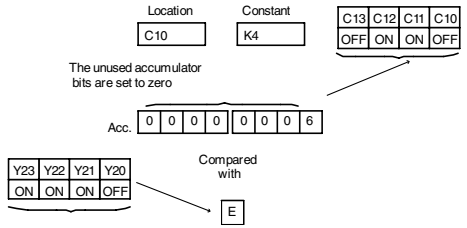
Discrete Bit Flags	Description
SP60	On when the value in the accumulator is less than the instruction value.
SP61	On when the value in the accumulator is equal to the instruction value.
SP62	On when the value in the accumulator is greater than the instruction value.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the Load Formatted instruction loads the binary value (6) from C10–C13 into the accumulator. The CMPF instruction compares the value in the accumulator to the value in Y20–Y23 (E hex). The corresponding discrete status flag will be turned on, indicating the result of the comparison. In this example, if the value in the accumulator is less than the value specified in the Compare instruction, SP60 will turn on energizing C30.

DirectSOFT

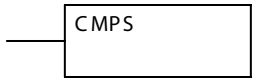




Compare with Stack (CMPS)

DS	Used
HPP	Used

The Compare with Stack instruction is a 32-bit instruction that compares the value in the accumulator with the value in the first level of the accumulator stack. The data format for this instruction is BCD/Hex, Decimal and Binary.

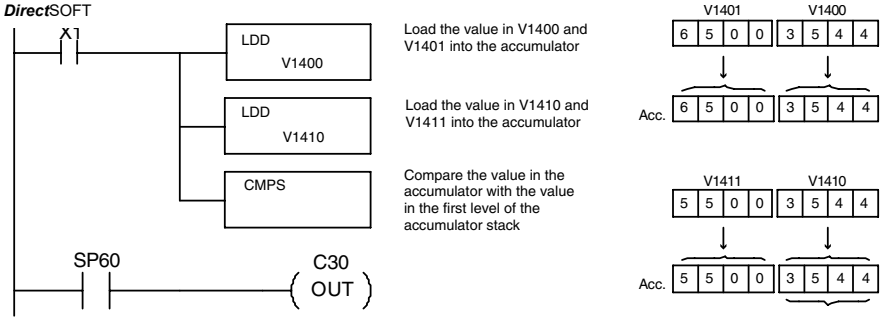


The corresponding status flag will be turned on, indicating the result of the comparison. This does not affect the value in the accumulator.

Discrete Bit Flags	Description
SP60	On when the value in the accumulator is less than the instruction value.
SP61	On when the value in the accumulator is equal to the instruction value.
SP62	On when the value in the accumulator is greater than the instruction value.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V1400 and V1401 is loaded into the accumulator using the Load Double instruction. The value in V1410 and V1411 is loaded into the accumulator using the Load Double instruction. The value that was loaded into the accumulator from V1400 and V1401 is placed on top of the stack when the second Load instruction is executed. The value in the accumulator is compared with the value in the first level of the accumulator stack using the CMPS instruction. The corresponding discrete status flag will be turned on indicating the result of the comparison. In this example, if the value in the accumulator is less than the value in the stack, SP60 will turn on, energizing C30.



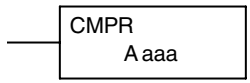
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	D 3	→	B 1	E 4	A 0	A 0	ENT
SHFT	L ANDST	D 3	D 3	→	B 1	E 4	B 1	A 0	ENT
SHFT	C 2	SHFT	M ORST	P CV	S RST	ENT			
$ STR	PREV	G 6	A 0	ENT					
GX OUT	→	NEXT	NEXT	NEXT	SHFT	C 2	D 3	A 0	ENT

Compare Real Number (CMPR)

DS	Used
HPP	Used

The Compare Real Number instruction compares a real number value in the accumulator with two consecutive V-memory locations containing a real number. The corresponding status flag will be turned on, indicating the result of the comparison. Both numbers being compared are 32 bits long.

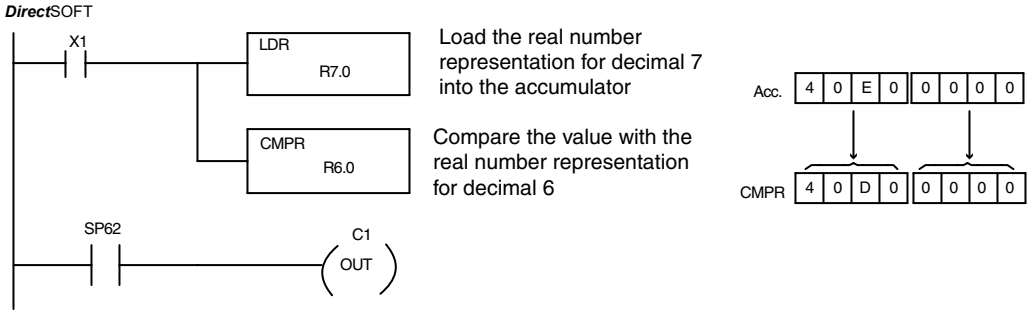


Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	R	-3.402823E+ 038 to + -3.402823E+ 038

Discrete Bit Flags	Description
SP60	On when the value in the accumulator is less than the instruction value.
SP61	On when the value in the accumulator is equal to the instruction value.
SP62	On when the value in the accumulator is greater than the instruction value.
SP71	On anytime the V-memory specified by a pointer (P) is not valid

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the LDR instruction loads the real number representation for 7 decimal into the accumulator. The CMPR instruction compares the accumulator contents with the real representation for decimal 6. Since 7 > 6, the corresponding discrete status flag is turned on (special relay SP62), turning on control relay C1.

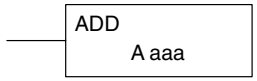


Math Instructions

Add (ADD)

DS	Used
HPP	Used

Add is a 16-bit instruction that adds a BCD value in the accumulator with a BCD value in a V-memory location (Aaaa). (You cannot use a constant as the parameter in the box.) The result resides in the accumulator.

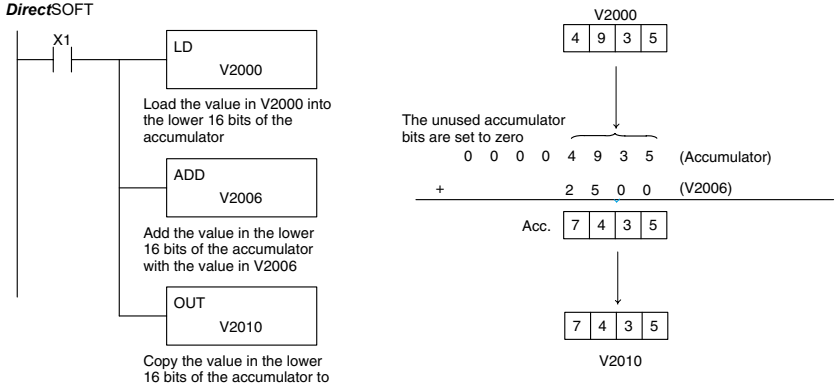


Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP66	On when the 16-bit addition instruction results in a carry.
SP67	On when the 32-bit addition instruction results in a carry.
SP70	On anytime the value in the accumulator is negative.
SP75	On when a BCD instruction is executed and a NON–BCD number was encountered.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V2000 will be loaded into the accumulator using the Load instruction. The value in the lower 16 bits of the accumulator is added to the value in V2006 using the Add instruction. The value in the accumulator is copied to V2010 using the Out instruction.



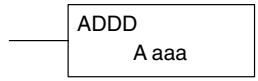
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT					
SHFT	L ANDST	D 3	→	C 2	A 0	A 0	A 0	ENT
SHFT	A 0	D 3	D 3	→	C 2	A 0	A 0	G 6	ENT
GX OUT	→	SHFT	V AND	C 2	A 0	B 1	A 0	ENT	

Add Double (ADDD)

DS	Used
HPP	Used

Add Double is a 32-bit instruction that adds the BCD value in the accumulator with a BCD value (Aaaa), which is either two consecutive V-memory locations or an 8–digit (max.) BCD constant. The result resides in the accumulator.

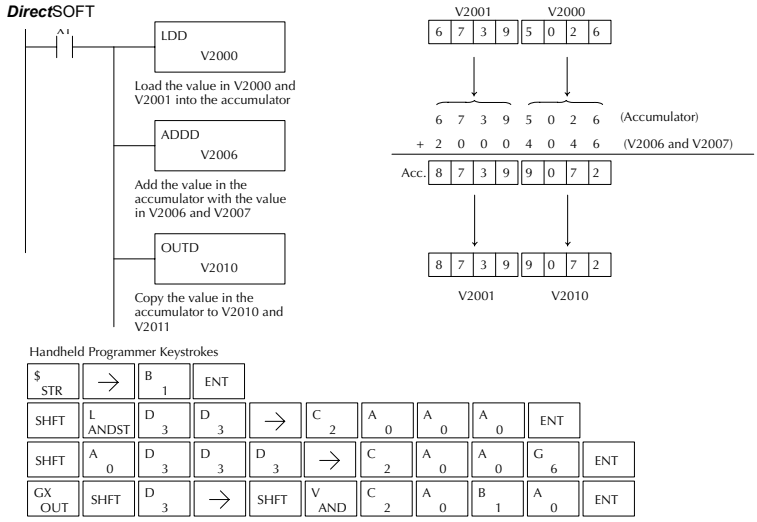


Operand Data Type		DL06Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	K	0–99999999

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP66	On when the 16-bit addition instruction results in a carry.
SP67	On when the 32-bit addition instruction results in a carry.
SP70	On anytime the value in the accumulator is negative.
SP75	On when a BCD instruction is executed and a NON–BCD number was encountered.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

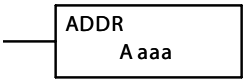
In the following example, when X1 is on, the value in V2000 and V2001 will be loaded into the accumulator using the Load Double instruction. The value in the accumulator is added with the value in V2006 and V2007 using the Add Double instruction. The value in the accumulator is copied to V2010 and V2011 using the Out Double instruction.



Add Real (ADDR)

DS	Used
HPP	Used

The Add Real instruction adds a real number in the accumulator with either a real constant or a real number occupying two consecutive V-memory locations. The result resides in the accumulator. Both numbers must be Real data type (IEEE floating point format).



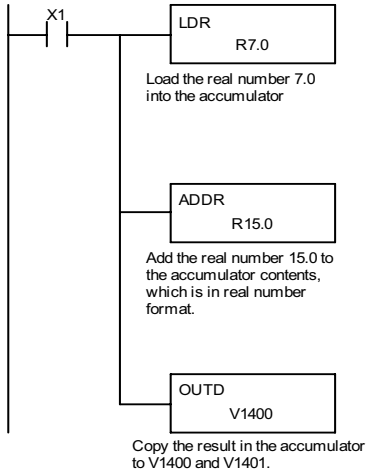
Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	R	-3.402823E+ 38 to + -3.402823E+ 38

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.
SP71	On anytime the V-memory specified by a pointer (P) is not valid.
SP72	On anytime the value in the accumulator is an invalid floating point number.
SP73	On when a signed addition or subtraction results in a incorrect sign bit.
SP74	On anytime a floating point math operation results in an underflow error.

NOTE: *Status flags are valid only until another instruction uses the same flag.*

NOTE*: The current HPP does not support real number entry with automatic conversion to the 32-bit IEEE format. You must use* ***Direct*****SOFT** *for this feature.*

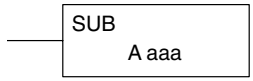
DirectSOFT



Subtract (SUB)

DS	Used
HPP	Used

Subtract is a 16-bit instruction that subtracts the BCD value (Aaaa) in a V-memory location from the BCD value in the lower 16 bits of the accumulator. The result resides in the accumulator.



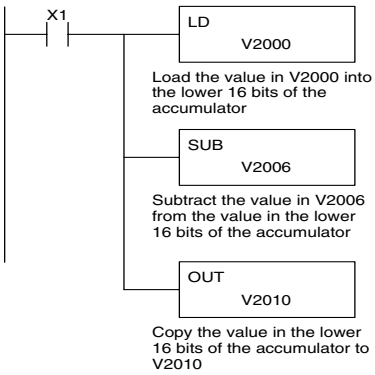
Operand Data Type		DL06Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map

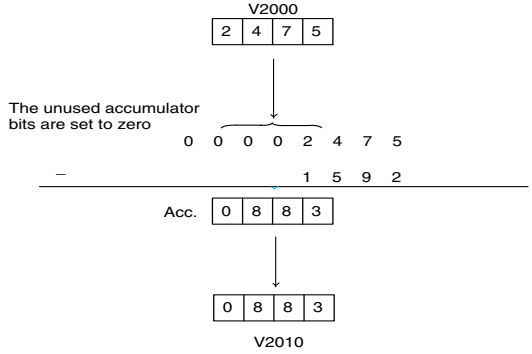
Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP64	On when the 16-bit subtraction instruction results in a borrow
SP65	On when the 32-bit subtraction instruction results in a borrow
SP70	On anytime the value in the accumulator is negative.
SP75	On when a BCD instruction is executed and a NON–BCD number was encountered.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V2000 will be loaded into the accumulator using the Load instruction. The value in V2006 is subtracted from the value in the accumulator using the Subtract instruction. The value in the accumulator is copied to V2010 using the Out instruction.

DirectSOFT





Handheld Programmer Keystrokes

$ STR	→	B 1	ENT							
SHFT	L ANDST	D 3	→	C 2	A 0	A 0	A 0	ENT		
SHFT	S RST	U ISG	B 1	→	SHFT	V AND	C 2	A 0	A 0	G 6	ENT
GX OUT	→	SHFT	V AND	C 2	A 0	B 1	A 0	ENT		

Subtract Double (SUBD)

DS	Used
HPP	Used

Subtract Double is a 32-bit instruction that subtracts the BCD value (Aaaa), which is either two consecutive V-memory locations or an 8-digit (max.) constant, from the BCD value in the accumulator.

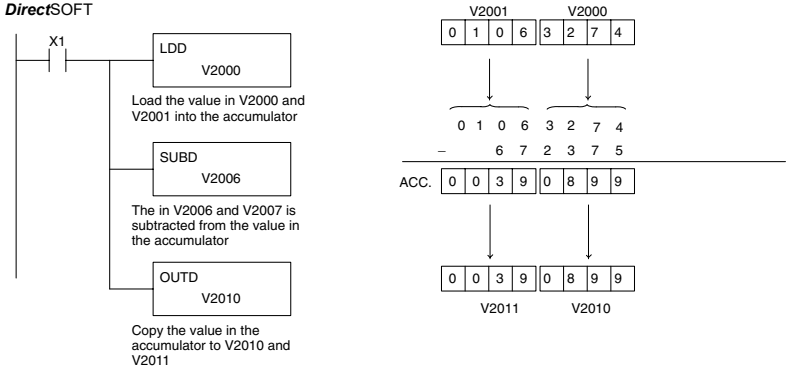
SUBD
A aaa

Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	K	0–99999999

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP64	On when the 16- bit subtraction instruction results in a borrow
SP65	On when the 32-bit subtraction instruction results in a borrow
SP70	On anytime the value in the accumulator is negative.
SP75	On when a BCD instruction is executed and a NON–BCD number was encountered.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V2000 and V2001 will be loaded into the accumulator using the Load Double instruction. The value in V2006 and V2007 is subtracted from the value in the accumulator. The value in the accumulator is copied to V2010 and V2011 using the Out Double instruction.



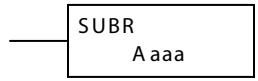
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT									
SHFT	L ANDST	D 3	D 3	→	C 2	A 0	A 0	A 0	ENT			
SHFT	S RST	SHFT	U ISG	B 1	D 3	→	C 2	A 0	A 0	G 6	ENT	
GX OUT	SHFT	D 3	→	C 2	A 0	B 1	A 0	ENT				

Subtract Real (SUBR)

DS	Used
HPP	N/A

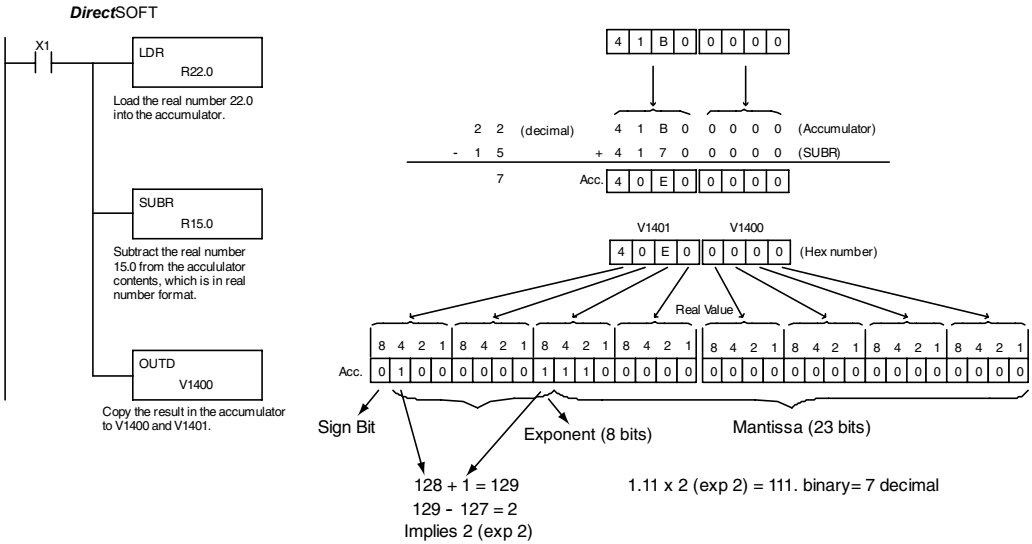
The Subtract Real is a 32-bit instruction that subtracts a real number, which is either two consecutive V-memory locations or a 32-bit constant, from a real number in the accumulator. The result is a 32-bit real number that resides in the accumulator. Both numbers must be Real data type (IEEE floating point format).



Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	R	-3.402823E + 38 to+-3.402823E + 38

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.
SP71	On anytime the V-memory specified by a pointer (P) is not valid.
SP72	On anytime the value in the accumulator is an invalid floating point number.
SP73	On when a signed addition or subtraction results in a incorrect sign bit.
SP74	On anytime a floating point math operation results in an underflow error.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

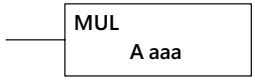


***NOTE**: The current HPP does not support real number entry with automatic conversion to the 32-bit IEEE format. You must use **DirectSOFT** for this feature*

Multiply (MUL)

DS	Used
HPP	Used

Multiply is a 16-bit instruction that multiplies the BCD value (Aaaa), which is either a V-memory location or a 4–digit (max.) constant, by the BCD value in the lower 16 bits of the accumulator The result can be up to 8 digits and resides in the accumulator.

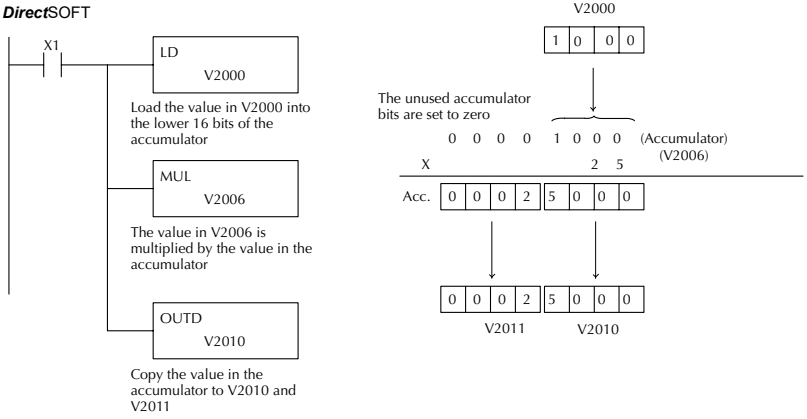


Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	K	0–9999

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.
SP75	On when a BCD instruction is executed and a NON–BCD number was encountered.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V2000 will be loaded into the accumulator using the Load instruction. The value in V2006 is multiplied by the value in the accumulator. The value in the accumulator is copied to V2010 and V2011 using the Out Double instruction.



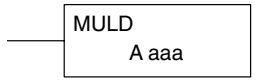
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	→	C 2	A 0	A 0	A 0	ENT	
SHFT	M ORST	U ISG	L ANDST	→	C 2	A 0	A 0	G 6	ENT
GX OUT	SHFT	D 3	→	C 2	A 0	B 1	A 0	ENT	

Multiply Double (MULD)

DS	Used
HPP	Used

Multiply Double is a 32-bit instruction that multiplies the 8-digit BCD value in the accumulator by the 8-digit BCD value in the two consecutive V-memory locations specified in the instruction. The lower 8 digits of the results reside in the accumulator. Upper digits of the result reside in the accumulator stack.

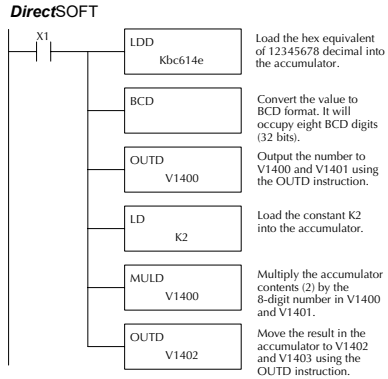


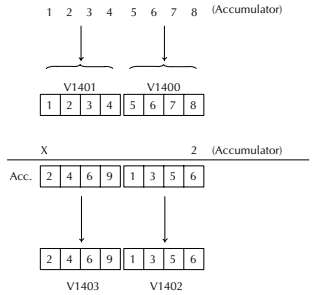
Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.
SP75	On when a BCD instruction is executed and a NON–BCD number was encountered.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the constant Kbc614e hex will be loaded into the accumulator. When converted to BCD the number is "12345678". That number is stored in V1400 and V1401. After loading the constant K2 into the accumulator, we multiply it times 12345678, which gives us 24691356.





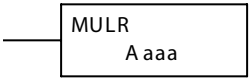
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT																				
SHFT	L ANDST	D 3	D 3	→	PREV	SHFT	B 1	C 2	SHFT	G 6	B 1	E 4	SHFT	E 4	ENT								
SHFT	B 1	C 2	D 3	ENT																			
GX OUT	SHFT	D 3	→	B 1	E 4	A 0	A 0	ENT															
SHFT	L ANDST	D 3	→	PREV	C 2	ENT																	
SHFT	M ORST	U ISG	L ANDST	D 3	→	B 1	E 4	A 0	A 0	ENT													
GX OUT	SHFT	D 3	→	B 1	E 4	A 0	C 2	ENT															

Multiply Real (MULR)

DS	Used
HPP	Used

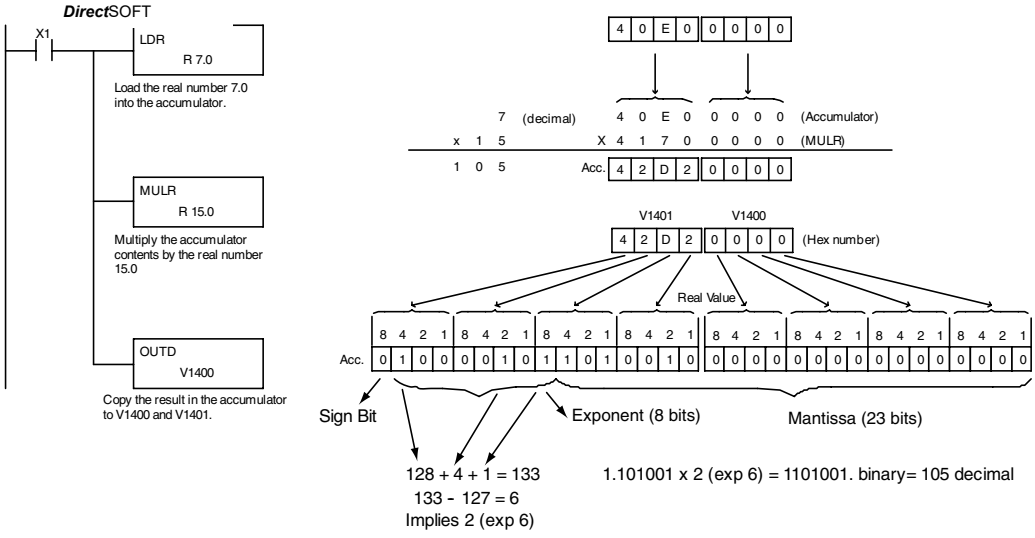
The Multiply Real instruction multiplies a real number in the accumulator with either a real constant or a real number occupying two consecutive V-memory locations. The result resides in the accumulator. Both numbers must be Real data type (IEEE floating point format).



Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Real Constant	R	-3.402823E +38 to + -3.402823E +38

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.
SP71	On anytime the V-memory specified by a pointer (P) is not valid.
SP72	On anytime the value in the accumulator is an invalid floating point number.
SP73	On when a signed addition or subtraction results in an incorrect sign bit.
SP74	On anytime a floating point math operation results in an underflow error.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

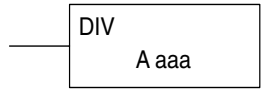


***NOTE**: The current HPP does not support real number entry with automatic conversion to the 32-bit IEEE format. You must use **DirectSOFT** for this feature.*

Divide (DIV)

DS	Used
HPP	Used

Divide is a 16-bit instruction that divides the BCD value in the accumulator by a BCD value (Aaaa), which is either a V-memory location or a 4-digit (max.) constant. The first part of the quotient resides in the accumulator and the remainder resides in the first stack location.

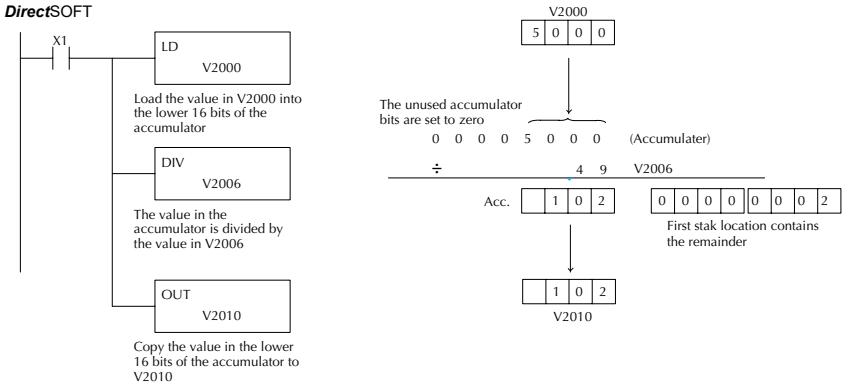


Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	K	0–9999

Discrete Bit Flags	Description
SP53	On when the value of the operand is larger than the accumulator can work with.
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.
SP75	On when a BCD instruction is executed and a NON–BCD number was encountered.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V2000 will be loaded into the accumulator using the Load instruction. The value in the accumulator will be divided by the value in V2006 using the Divide instruction. The value in the accumulator is copied to V2010 using the Out instruction.



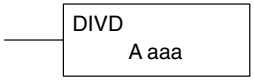
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT					
SHFT	L ANDST	D 3	→	C 2	A 0	A 0	A 0	ENT
SHFT	D 3	I 8	V AND	→	C 2	A 0	A 0	G 6	ENT
GX OUT	→	SHFT	V AND	C 2	A 0	B 1	A 0	ENT

Divide Double (DIVD)

DS	Used
HPP	Used

Divide Double is a 32-bit instruction that divides the BCD value in the accumulator by a BCD value (Aaaa), which must be obtained from two consecutive V-memory locations. (You cannot use a constant as the parameter in the box.) The first part of the quotient resides in the accumulator and the remainder resides in the first stack location.

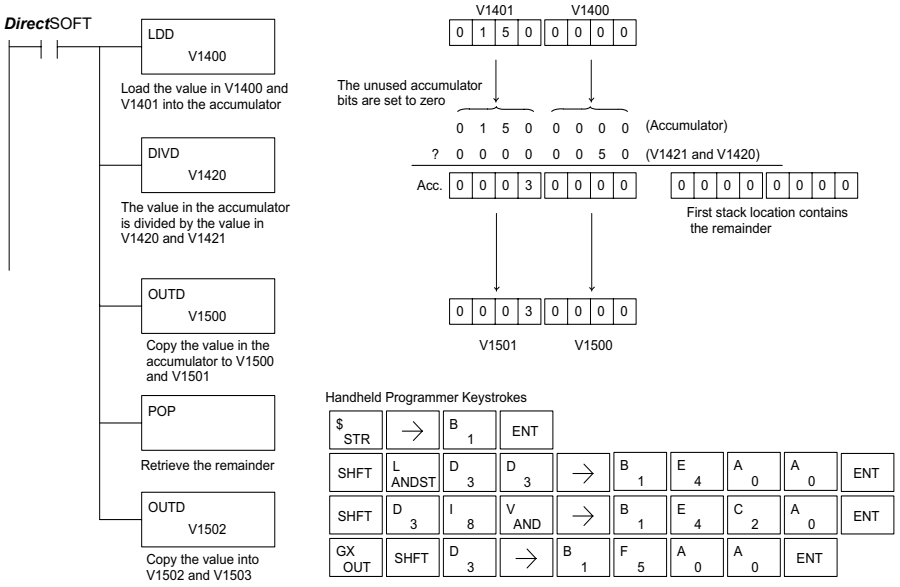


Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.
SP75	On when a BCD instruction is executed and a NON–BCD number was encountered.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

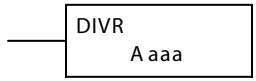
In the following example, when X1 is on, the value in V1400 and V1401 will be loaded into the accumulator using the Load Double instruction. The value in the accumulator is divided by the value in V1420 and V1421 using the Divide Double instruction. The first part of the quotient resides in the accumulator and the remainder resides in the first stack location. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.



Divide Real (DIVR)

DS	Used
HPP	N/A

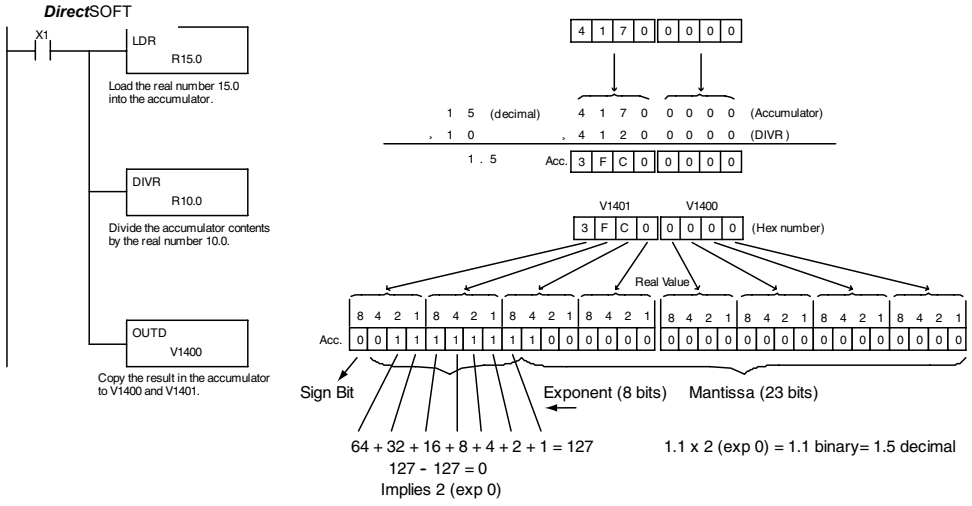
The Divide Real instruction divides a real number in the accumulator by either a real constant or a real number occupying two consecutive V-memory locations. The result resides in the accumulator. Both numbers must be Real data type (IEEE floating point format).



Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Real Constant	R	-3.402823E + 38 to + -3.402823E + 38

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.
SP71	On anytime the V-memory specified by a pointer (P) is not valid.
SP72	On anytime the value in the accumulator is an invalid floating point number.
SP74	On anytime a floating point math operation results in an underflow error.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

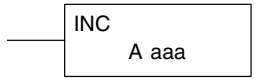


***NOTE**: The current HPP does not support real number entry with automatic conversion to the 32-bit IEEE format. You must use **Direct**SOFT for this feature.*

Increment (INC)

DS	Used
HPP	Used

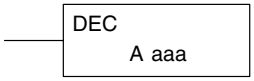
The Increment instruction increments a BCD value in a specified V-memory location by "1" each time the instruction is executed.



Decrement (DEC)

DS	Used
HPP	Used

The Decrement instruction decrements a BCD value in a specified V-memory location by "1" each time the instruction is executed.

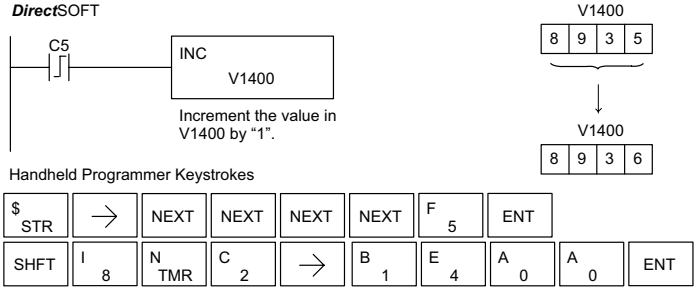


Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map

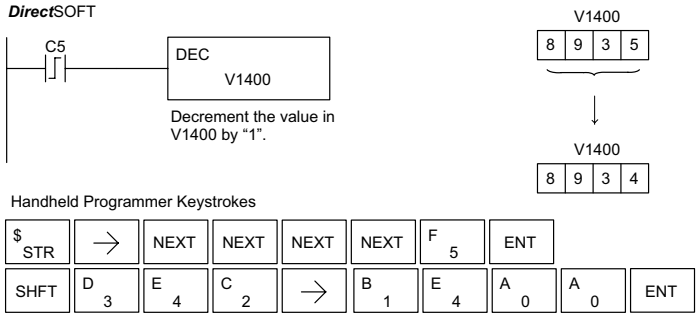
Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP75	On when a BCD instruction is executed and a NON–BCD number was encountered.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following increment example, when C5 makes an Off-to-On transition the value in V1400 increases by one.



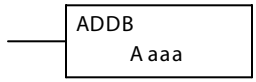
In the following decrement example, when C5 makes an Off-to-On transition the value in V1400 is decreased by one.



Add Binary (ADDB)

DS	Used
HPP	Used

Add Binary is a 16-bit instruction that adds the binary value in the lower 16 bits of the accumulator with a binary value (Aaaa), which is either a V-memory location or a 16-bit constant. The result can be up to 32 bits and resides in the accumulator.

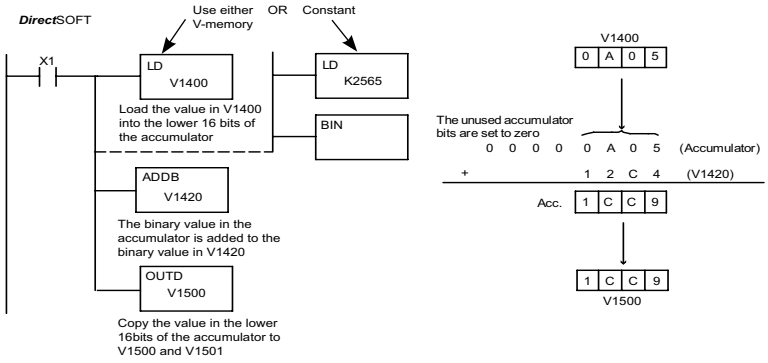


Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	K	0-FFFF, h=65636

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP66	On when the 16-bit addition instruction results in a carry.
SP67	On when the 32-bit addition instruction results in a carry.
SP70	On anytime the value in the accumulator is negative.
SP73	On when a signed addition or subtraction results in an incorrect sign bit.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V1400 will be loaded into the accumulator using the Load instruction. The binary value in the accumulator will be added to the binary value in V1420 using the Add Binary instruction. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.



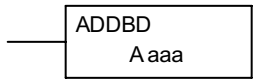
Handheld Programmer Keystrokes

STR	→	X(IN)	1	ENT							
SHFT	L	D	V	1	4	O	O	ENT			
SHFT	A	D	D	B	→	V	1	4	2	O	ENT
OUT	SHFT	D	→	V	1	5	O	O	ENT		

Add Binary Double (ADDBD)

DS	Used
HPP	Used

Add Binary Double is a 32-bit instruction that adds the binary value in the accumulator with the value (Aaaa), which is either two consecutive V-memory locations or an 8-digit (max.) binary constant. The result resides in the accumulator.

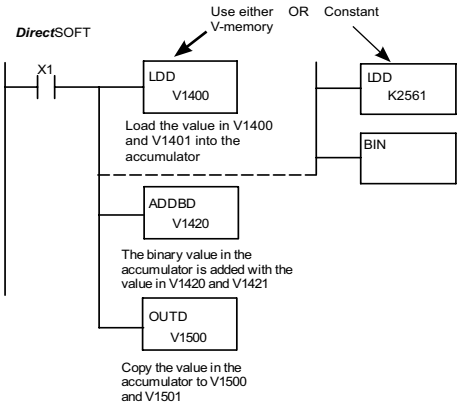


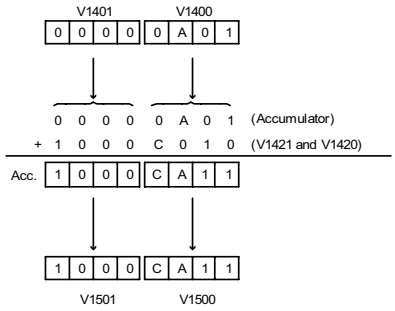
Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	K	0-FFFF FFFF

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP66	On when the 16-bit addition instruction results in a carry.
SP67	On when the 32-bit addition instruction results in a carry.
SP70	On anytime the value in the accumulator is negative.
SP73	On when a signed addition or subtraction results in an incorrect sign bit.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V1400 and V1401 will be loaded into the accumulator using the Load Double instruction. The binary value in the accumulator is added with the binary value in V1420 and V1421 using the Add Binary Double instruction. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.





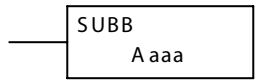
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT									
SHFT	L ANDST	D 3	D 3	→	B 1	E 4	A 0	A 0	ENT			
SHFT	A 0	D 3	D 3	B 1	D 3	→	B 1	E 4	C 2	A 0	ENT	
GX OUT	SHFT	D 3	→	B 1	F 5	A 0	A 0	ENT				

Subtract Binary (SUBB)

DS	Used
HPP	Used

Subtract Binary is a 16-bit instruction that subtracts the binary value (Aaaa), which is either a V-memory location or a 4-digit (max.) binary constant, from the binary value in the accumulator. The result resides in the accumulator.

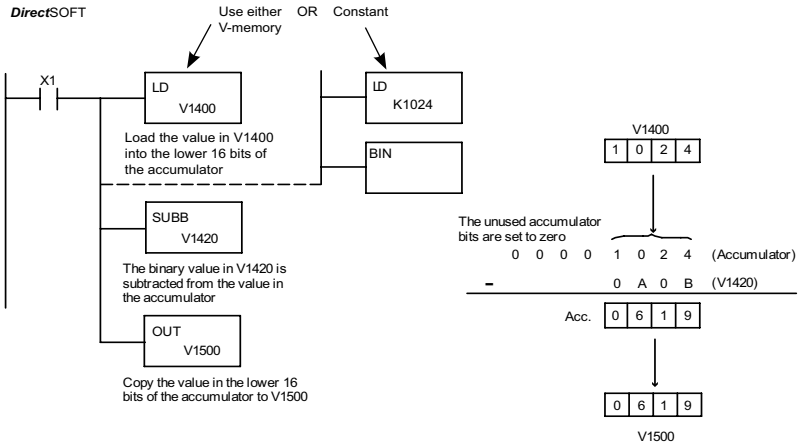


Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	K	0-FFFF, h=65636

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP64	On when the 16-bit subtraction instruction results in a borrow.
SP65	On when the 32-bit subtraction instruction results in a borrow.
SP70	On anytime the value in the accumulator is negative.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V1400 will be loaded into the accumulator using the Load instruction. The binary value in V1420 is subtracted from the binary value in the accumulator using the Subtract Binary instruction. The value in the accumulator is copied to V1500 using the Out instruction.



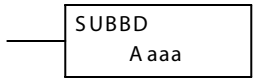
Handheld Programmer Keystrokes

STR	→	X(IN)	1	ENT				
SHFT	L	D	V	1	4	0	0	ENT
SHFT	S	SHFT	U	B	B	→		
V	1	4	2	0	ENT			
OUT	SHFT	D	→	V	1	5	0	0	ENT

Subtract Binary Double (SUBBD)

DS	Used
HPP	Used

Subtract Binary Double is a 32-bit instruction that subtracts the binary value (Aaaa), which is either two consecutive V-memory locations or an 8-digit (max.) binary constant, from the binary value in the accumulator. The result resides in the accumulator.

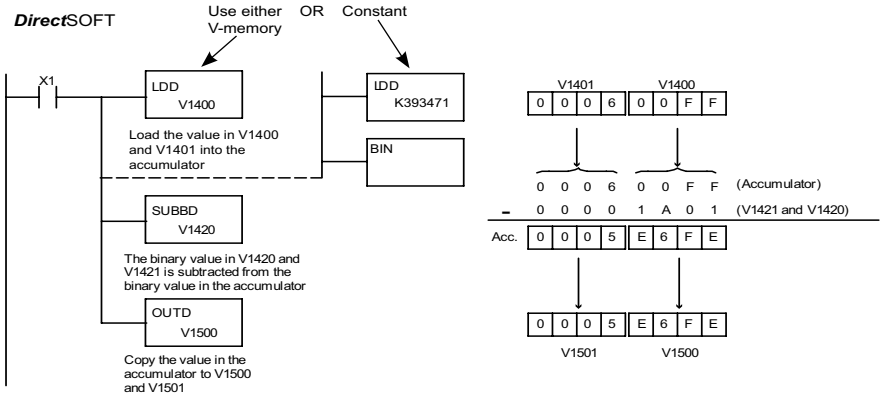


Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	K	0-FFFF FFFF

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP64	On when the 16-bit subtraction instruction results in a borrow.
SP65	On when the 32-bit subtraction instruction results in a borrow.
SP70	On anytime the value in the accumulator is negative.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V1400 and V1401 will be loaded into the accumulator using the Load Double instruction. The binary value in V1420 and V1421 is subtracted from the binary value in the accumulator using the Subtract Binary Double instruction. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.



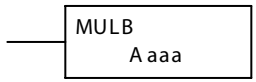
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT									
SHFT	L ANDST	D 3	D 3	→	B 1	E 4	A 0	A 0	ENT			
SHFT	S RST	SHFT	U ISG	B 1	B 1	D 3	→	B 1	E 4	C 2	A 0	ENT
GX OUT	SHFT	D 3	→	B 1	F 5	A 0	A 0	ENT				

Multiply Binary (MULB)

DS	Used
HPP	Used

Multiply Binary is a 16-bit instruction that multiplies the binary value (Aaaa), which is either a V-memory location or a 4-digit (max.) binary constant, by the binary value in the accumulator. The result can be up to 32 bits and resides in the accumulator.

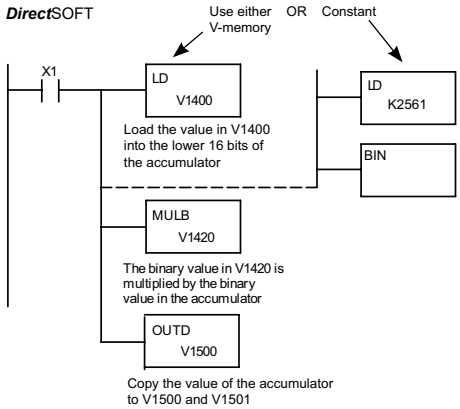


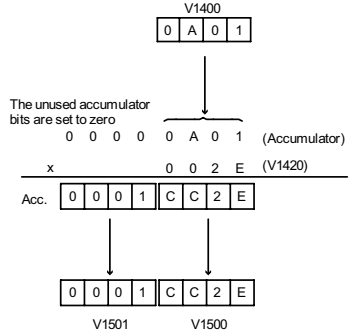
Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	K	0-FFFF

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V1400 will be loaded into the accumulator using the Load instruction. The binary value in V1420 is multiplied by the binary value in the accumulator using the Multiply Binary instruction. The value in the accumulator is copied to V1500 using the Out instruction.





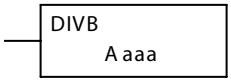
Handheld Programmer Keystrokes

STR	→	X	1	ENT						
SHFT	L	D	V	1	4	0	0	ENT		
SHFT	M	U	L	B	→	V	1	4	2	0	ENT
OUT	SHFT	D	→	V	1	5	0	0	ENT	

Divide Binary (DIVB)

DS	Used
HPP	Used

Divide Binary is a 16-bit instruction that divides the binary value in the accumulator by a binary value (Aaaa), which is either a V-memory location or a 16-bit (max.) binary constant. The first part of the quotient resides in the accumulator and the remainder resides in the first stack location.

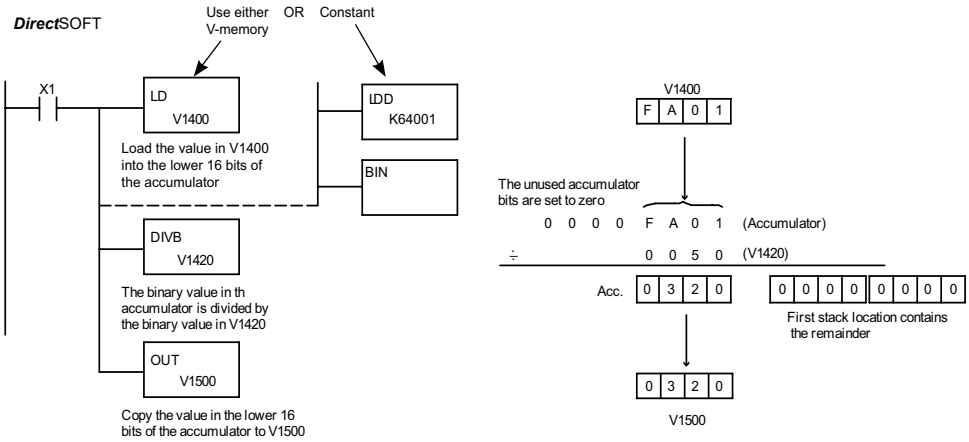


Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	K	0-FFFF

Discrete Bit Flags	Description
SP53	On when the value of the operand is larger than the accumulator can work with.
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V1400 will be loaded into the accumulator using the Load instruction. The binary value in the accumulator is divided by the binary value in V1420 using the Divide Binary instruction. The value in the accumulator is copied to V1500 using the Out instruction.



Handheld Programmer Keystrokes

STR	→	X	1	ENT						
SHFT	L	D	V	1	4	0	0	ENT		
SHFT	D	I	V	B	→	V	1	4	2	0	ENT
OUT	SHFT	D	→	V	1	5	0	0	ENT	

Increment Binary (INCB)

DS	Used
HPP	Used

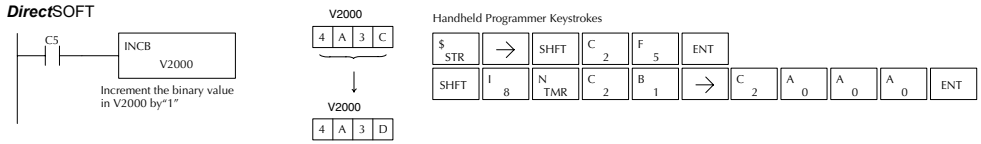
The Increment Binary instruction increments a binary value in a specified V-memory location by "1" each time the instruction is executed.

INCB
A aaa

Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.

In the following example when C5 is on, the binary value in V2000 is increased by 1.



Decrement Binary (DECB)

DS	Used
HPP	Used

The Decrement Binary instruction decrements a binary value in a specified V-memory location by "1" each time the instruction is executed.

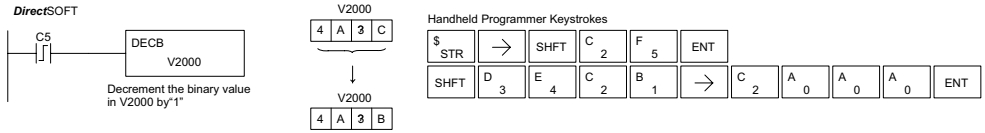
DECB
A aaa

Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.

NOTE: *Status flags are valid only until another instruction uses the same flag.*

In the following example when C5 is on, the value in V2000 is decreased by 1.



Add Formatted (ADDF)

DS	Used
HPP	Used

Add Formatted is a 32-bit instruction that adds the BCD value in the accumulator with the BCD value (Aaaa) which is a range of discrete bits. The specified range (Kbbb) can be 1 to 32 consecutive bits. The result resides in the accumulator.

```
ADDF      A aaa
          K bbb
```

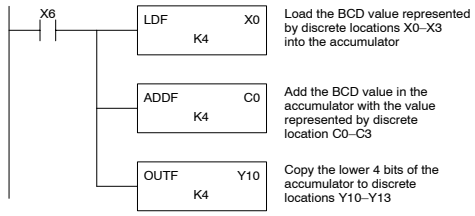
Operand Data Type	DL06 Range		
	A	aaa	bbb
Inputs	X	0–777	—
Outputs	Y	0–777	—
Control Relays	C	0–1777	—
Stage Bits	S	0–1777	—
Timer Bits	T	0–377	—
Counter Bits	CT	0–177	—
Special Relays	SP	0-137 320-717	—
Global I/O	GX	0-3777	—
Constant	K	—	1–32

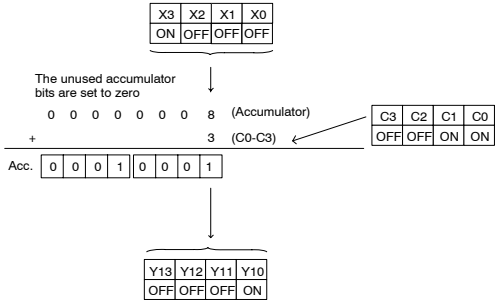
Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP66	On when the 16-bit addition instruction results in a carry.
SP67	On when the 32 bit addition instruction results in a carry.
SP70	On anytime the value in the accumulator is negative.
SP73	On when a signed addition or subtraction results in an incorrect sign bit.
SP75	On when a BCD instruction is executed and a NON-BCD number was encountered.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X6 is on, the BCD value formed by discrete locations X0–X3 is loaded into the accumulator using the LDF instruction. The BCD value formed by discrete locations C0–C3 is added to the value in the accumulator using the ADDF instruction. The value in the lower four bits of the accumulator is copied to Y10–Y13 using the OUTF instruction.

DirectSOFT





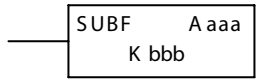
Handheld Programmer Keystrokes

$ STR	→	G 6	ENT												
SHFT	L ANDST	D 3	F 5	→	A 0	→	E 4	ENT							
SHFT	A 0	D 3	D 3	F 5	→	NEXT	NEXT	NEXT	NEXT	A 0	→	E 4	ENT		
GX OUT	SHFT	F 5	→	B 1	A 0	→	E 4	ENT							

Subtract Formatted (SUBF)

DS	Used
HPP	Used

Subtract Formatted is a 32-bit instruction that subtracts the BCD value (Aaaa), which is a range of discrete bits, from the BCD value in the accumulator. The specified range (Kbbb) can be 1 to 32 consecutive bits. The result resides in the accumulator.



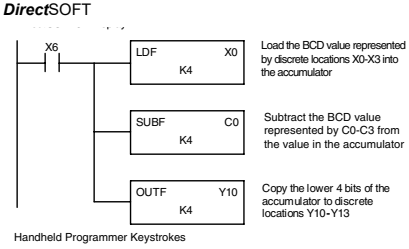
Operand Data Type		DL06 Range	
	A	aaa	bbb
Inputs	X	0–777	—
Outputs	Y	0–777	—
Control Relays	C	0–1777	—
Stage Bits	S	0–1777	—
Timer Bits	T	0–377	—
Counter Bits	CT	0–177	—
Special Relays	SP	0-137 320-717	—
Global I/O	GX	0-3777	—
Constant	K	—	1–32

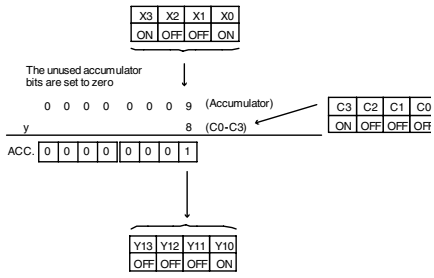
Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP64	On when the 16-bit subtraction instruction results in a borrow.
SP65	On when the 32 bit subtraction instruction results in a borrow
SP70	On any time the value in the accumulator is negative.
SP73	On when a signed addition or subtraction results in an incorrect sign bit.
SP75	On when a BCD instruction is executed and a NON-BCD number was encountered.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X6 is on, the BCD value formed by discrete locations X0–X3 is loaded into the accumulator using the LDF instruction. The BCD value formed by discrete location C0–C3 is subtracted from the BCD value in the accumulator using the SUBF instruction. The value in the lower four bits of the accumulator is copied to Y10–Y13 using the OUTF instruction.

DirectSOFT





Handheld Programmer Keystrokes

$ STR	→	G 6	ENT												
SHFT	L ANDST	D 3	F 5	→	A 0	→	E 4	ENT							
SHFT	S RST	SHFT	U ISG	B 1	F 5	→	NEXT	NEXT	NEXT	NEXT	A 0	→	E 4	ENT	
GX OUT	SHFT	F 5	→	B 1	A 0	→	E 4	ENT							

Multiply Formatted (MULF)

DS	Used
HPP	Used

Multiply Formatted is a 16-bit instruction that multiplies the BCD value in the accumulator by the BCD value (Aaaa) which is a range of discrete bits. The specified range (Kbbb) can be 1 to 16 consecutive bits. The result resides in the accumulator.

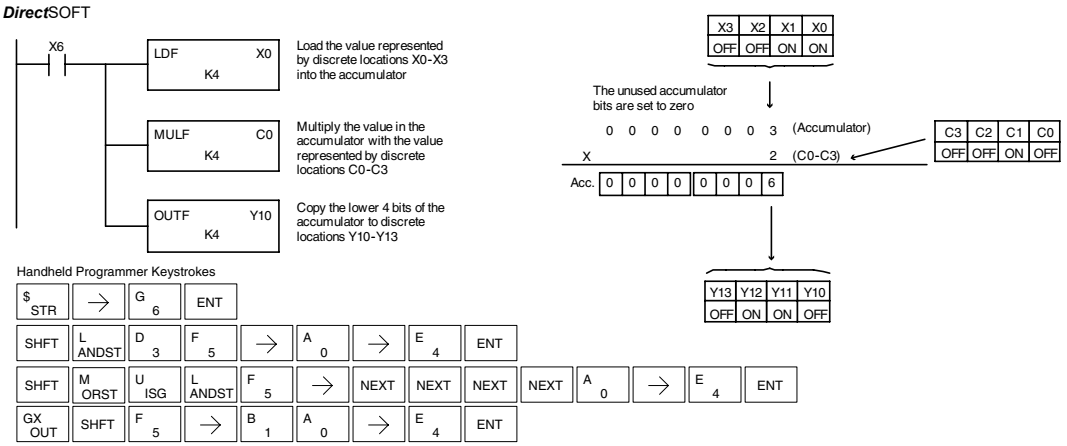
```
MULF    A aaa
        K bbb
```

Operand Data Type		DL06 Range	
	A	aaa	bbb
Inputs	X	0–777	—
Outputs	Y	0–777	—
Control Relays	C	0–1777	—
Stage Bits	S	0–1777	—
Timer Bits	T	0–377	—
Counter Bits	CT	0–177	—
Special Relays	SP	0-137 320-717	—
Global I/O	GX	0-3777	—
Constant	K	—	1–16

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On any time the value in the accumulator is negative.
SP75	On when a BCD instruction is executed and a NON-BCD number was encountered.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

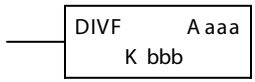
In the following example, when X6 is on, the value formed by discrete locations X0–X3 is loaded into the accumulator using the Load Formatted instruction. The value formed by discrete locations C0–C3 is multiplied by the value in the accumulator using the Multiply Formatted instruction. The value in the lower four bits of the accumulator is copied to Y10–Y13 using the Out Formatted instruction.



Divide Formatted (DIVF)

DS	Used
HPP	Used

Divide Formatted is a 16-bit instruction that divides the BCD value in the accumulator by the BCD value (Aaaa), a range of discrete bits. The specified range (Kbbb) can be 1 to 16 consecutive bits. The first part of the quotient resides in the accumulator and the remainder resides in the first stack location..

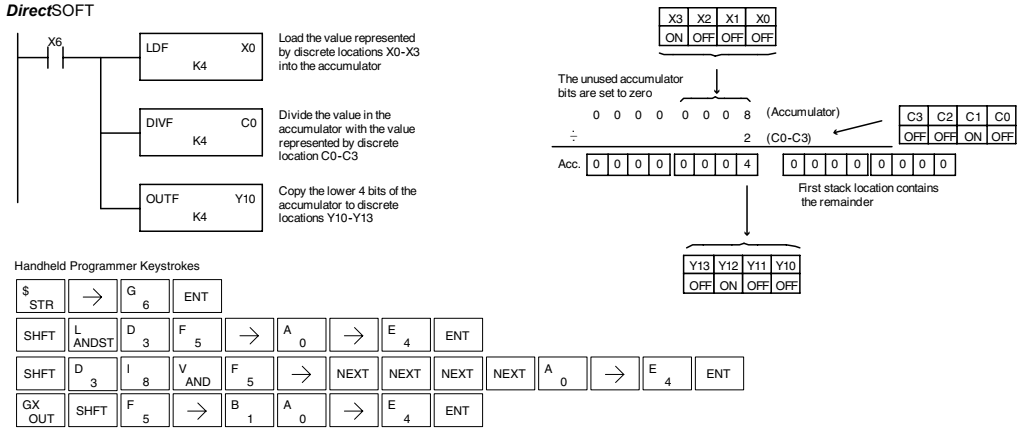


Operand Data Type	DL06 Range		
	A	aaa	bbb
Inputs	X	0–777	—
Outputs	Y	0–777	—
Control Relays	C	0–1777	—
Stage Bits	S	0–1777	—
Timer Bits	T	0–377	—
Counter Bits	CT	0–177	—
Special Relays	P	0-137 320-717	—
Global I/O	X	0-3777	—
Constant	K	—	1–16

Discrete Bit Flags	Description
SP53	On when the value of the operand is larger than the accumulator can work with.
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On any time the value in the accumulator is negative.
SP75	On when a BCD instruction is executed and a NON-BCD number was encountered.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

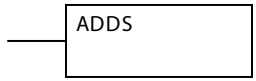
In the following example, when X6 is on, the value formed by discrete locations X0–X3 is loaded into the accumulator using the Load Formatted instruction. The value in the accumulator is divided by the value formed by discrete location C0–C3 using the Divide Formatted instruction. The value in the lower four bits of the accumulator is copied to Y10–Y13 using the Out Formatted instruction.



Add Top of Stack (ADDS)

DS	Used
HPP	Used

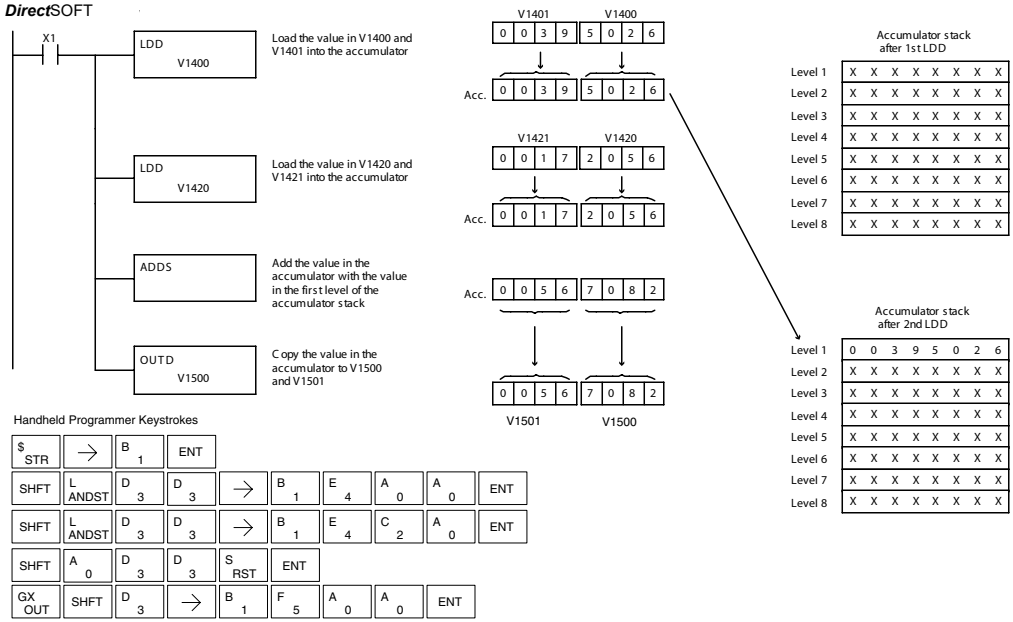
Add Top of Stack is a 32-bit instruction that adds the BCD value in the accumulator with the BCD value in the first level of the accumulator stack. The result resides in the accumulator. The value in the first level of the accumulator stack is removed and all stack values are moved up one level.



Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP66	On when the 16-bit addition instruction results in a carry.
SP67	On when the 32 bit addition instruction results in a carry.
SP70	On anytime the value in the accumulator is negative.
SP73	On when a signed addition or subtraction results in an incorrect sign bit.
SP75	On when a BCD instruction is executed and a NON-BCD number was encountered.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

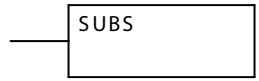
In the following example, when X1 is on, the value in V1400 and V1401 will be loaded into the accumulator using the Load Double instruction. The value in V1420 and V1421 is loaded into the accumulator using the Load Double instruction, pushing the value previously loaded in the accumulator onto the accumulator stack. The value in the first level of the accumulator stack is added with the value in the accumulator using the Add Stack instruction. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.



Subtract Top of Stack (SUBS)

DS	Used
HPP	Used

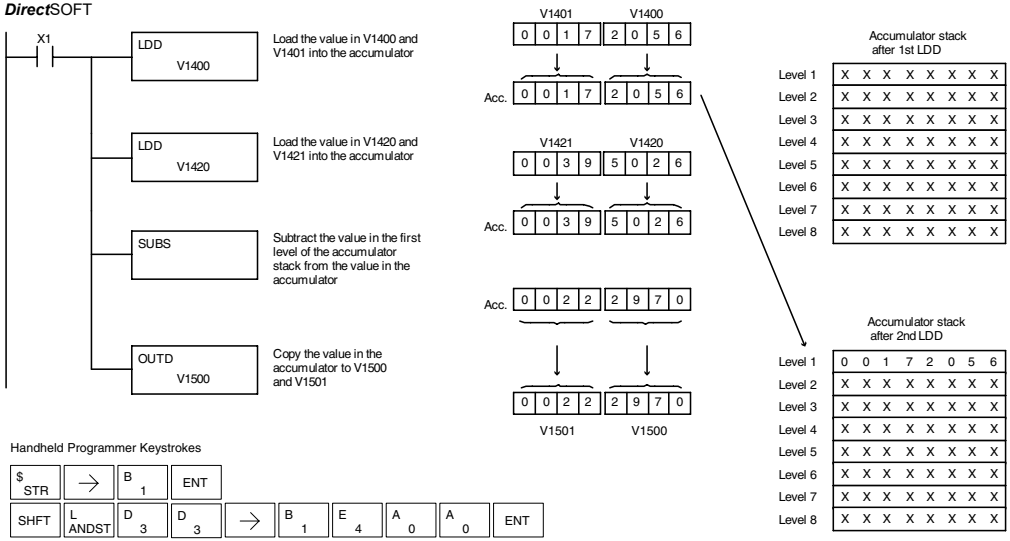
Subtract Top of Stack is a 32-bit instruction that subtracts the BCD value in the first level of the accumulator stack from the BCD value in the accumulator. The result resides in the accumulator. The value in the first level of the accumulator stack is removed and all stack values are moved up one level.



Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP64	On when the 16-bit subtraction instruction results in a borrow.
SP65	On when the 32 bit subtraction instruction results in a borrow.
SP70	On anytime the value in the accumulator is negative.
SP73	On when a signed addition or subtraction results in an incorrect sign bit.
SP75	On when a BCD instruction is executed and a NON-BCD number was encountered.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V1400 and V1401 will be loaded into the accumulator using the Load Double instruction. The value in V1420 and V1421 is loaded into the accumulator using the Load Double instruction, pushing the value previously loaded into the accumulator onto the accumulator stack. The BCD value in the first level of the accumulator stack is subtracted from the BCD value in the accumulator using the Subtract Stack instruction. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.



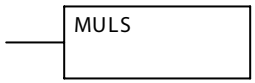
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	D 3	→	B 1	E 4	A 0	A 0	ENT
SHFT	L ANDST	D 3	D 3	→	B 1	E 4	C 2	A 0	ENT
SHFT	S RST	SHFT	U ISG	B 1	S RST	ENT			
GX OUT	SHFT	D 3	→	B 1	F 5	A 0	A 0	ENT	

Multiply Top of Stack (MULS)

DS	Used
HPP	Used

Multiply Top of Stack is a 16-bit instruction that multiplies a 4-digit BCD value in the first level of the accumulator stack by a 4-digit BCD value in the accumulator. The result resides in the accumulator. The value in the first level of the accumulator stack is removed and all stack values are moved up one level.

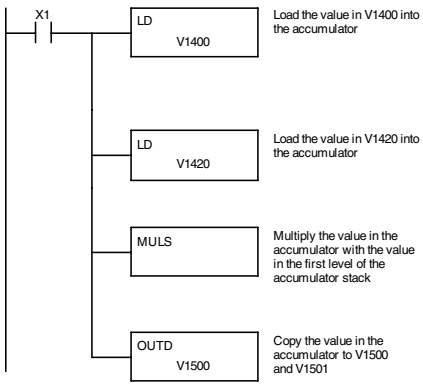


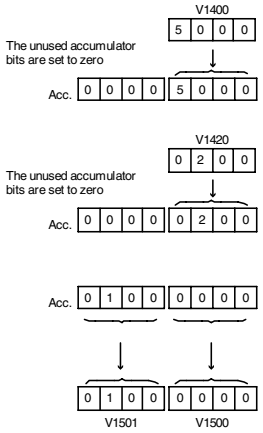
Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On any time the value in the accumulator is negative.
SP75	On when a BCD instruction is executed and a NON-BCD number was encountered.

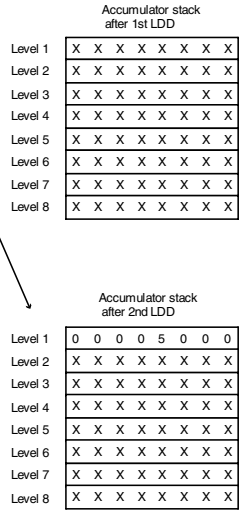
***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V1400 will be loaded into the accumulator using the Load instruction. The value in V1420 is loaded into the accumulator using the Load instruction, pushing the value previously loaded in the accumulator onto the accumulator stack. The BCD value in the first level of the accumulator stack is multiplied by the BCD value in the accumulator using the Multiply Stack instruction. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.

***Direct*SOFT**







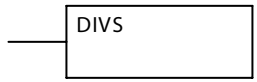
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT					
SHFT	L ANDST	D 3	→	B 1	E 4	A 0	A 0	ENT
SHFT	L ANDST	D 3	→	B 1	E 4	C 2	A 0	ENT
SHFT	M ORST	U ISG	L ANDST	S RST	ENT			
GX OUT	SHFT	D 3	→	B 1	F 5	A 0	A 0	ENT

Divide by Top of Stack (DIVS)

DS	Used
HPP	Used

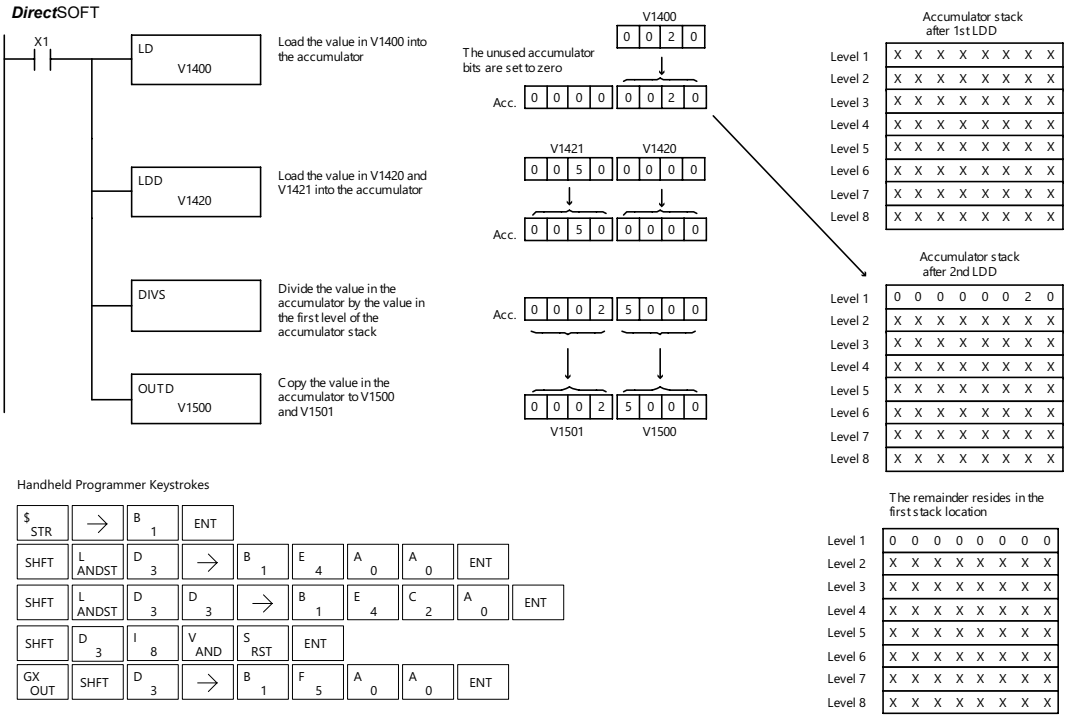
Divide Top of Stack is a 32-bit instruction that divides the 8-digit BCD value in the accumulator by a 4-digit BCD value in the first level of the accumulator stack. The result resides in the accumulator and the remainder resides in the first level of the accumulator stack.



Discrete Bit Flags	Description
SP53	On when the value of the operand is larger than the accumulator can work with.
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On any time the value in the accumulator is negative.
SP75	On when a BCD instruction is executed and a NON-BCD number was encountered.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

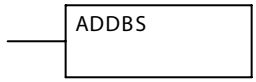
In the following example, when X1 is on, the Load instruction loads the value in V1400 into the accumulator. The value in V1420 is loaded into the accumulator using the Load Double instruction, pushing the value previously loaded in the accumulator onto the accumulator stack. The BCD value in the accumulator is divided by the BCD value in the first level of the accumulator stack using the Divide Stack instruction. The Out Double instruction copies the value in the accumulator to V1500 and V1501.



Add Binary Top of Stack (ADDBS)

DS	Used
HPP	Used

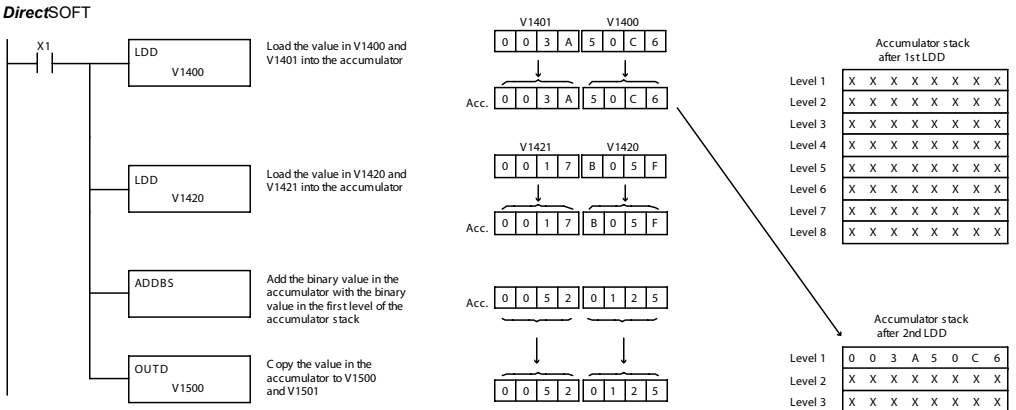
Add Binary Top of Stack instruction is a 32-bit instruction that adds the binary value in the accumulator with the binary value in the first level of the accumulator stack. The result resides in the accumulator. The value in the first level of the accumulator stack is removed and all stack values are moved.



Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP66	On when the 16-bit addition instruction results in a carry.
SP67	On when the 32 bit addition instruction results in a carry.
SP70	On anytime the value in the accumulator is negative.
SP73	On when a signed addition or subtraction results in an incorrect sign bit.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V1400 and V1401 will be loaded into the accumulator using the Load Double instruction. The value in V1420 and V1421 is loaded into the accumulator using the Load Double instruction, pushing the value previously loaded in the accumulator onto the accumulator stack. The binary value in the first level of the accumulator stack is added with the binary value in the accumulator using the Add Stack instruction. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.



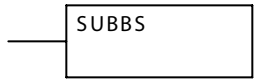
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	D 3	→	B 1	E 4	A 0	A 0	ENT
SHFT	L ANDST	D 3	D 3	→	B 1	E 4	C 2	A 0	ENT
SHFT	A 0	D 3	D 3	B 1	S RST	ENT			
GX OUT	SHFT	D 3	→	B 1	F 5	A 0	A 0	ENT	

Subtract Binary Top of Stack (SUBBS)

DS	Used
HPP	Used

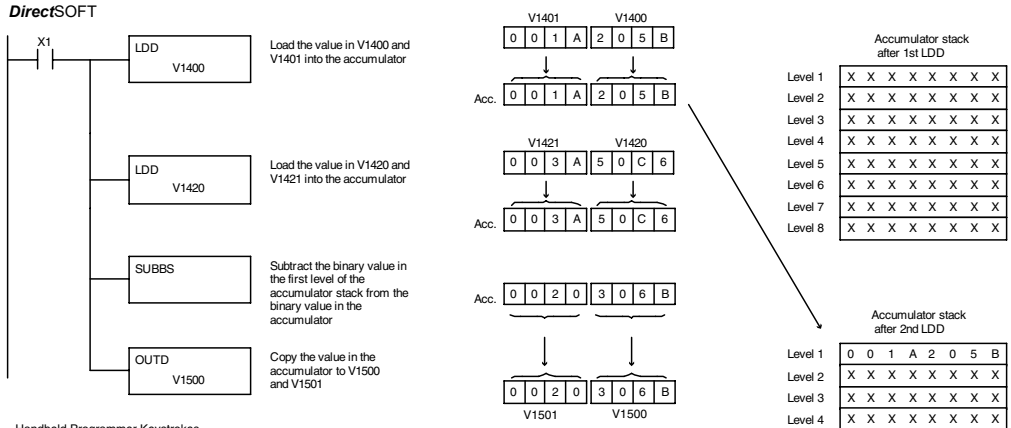
Subtract Binary Top of Stack is a 32-bit instruction that subtracts the binary value in the first level of the accumulator stack from the binary value in the accumulator. The result resides in the accumulator. The value in the first level of the accumulator stack is removed and all stack locations are moved up one level



Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP64	On when the 16-bit subtraction instruction results in a borrow.
SP65	On when the 32-bit subtraction instruction results in a borrow.
SP70	On any time the value in the accumulator is negative.
SP73	On when a signed addition or subtraction results in an incorrect sign bit.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V1400 and V1401 will be loaded into the accumulator using the Load Double instruction. The value in V1420 and V1421 is loaded into the accumulator using the Load Double instruction, pushing the value previously loaded in the accumulator onto the accumulator stack. The binary value in the first level of the accumulator stack is subtracted from the binary value in the accumulator using the Subtract Stack instruction. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.



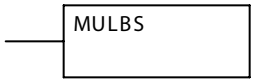
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	D 3	→	B 1	E 4	A 0	A 0	ENT
SHFT	L ANDST	D 3	D 3	→	B 1	E 4	C 2	A 0	ENT
SHFT	S RST	SHFT	U ISG	B 1	B 1	S RST	ENT		
GX OUT	SHFT	D 3	→	B 1	F 5	A 0	A 0	ENT	

Multiply Binary Top of Stack (MULBS)

DS	Used
HPP	Used

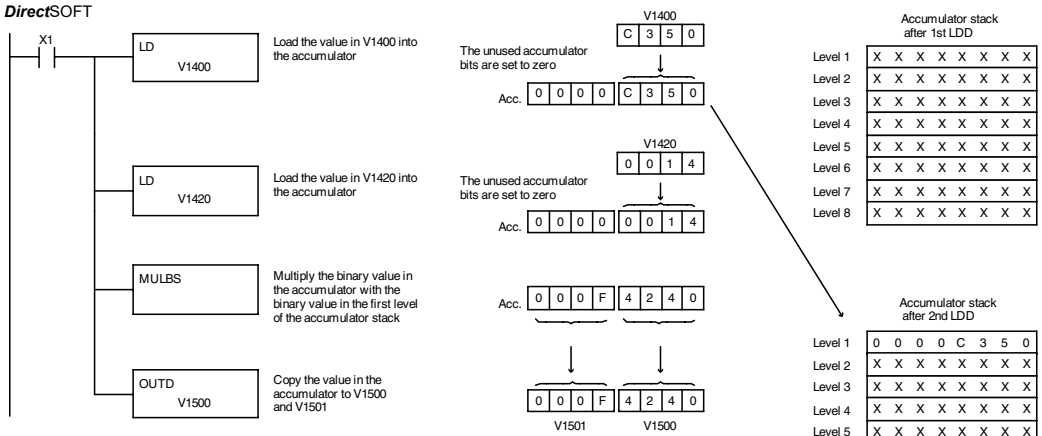
Multiply Binary Top of Stack is a 16-bit instruction that multiplies the 16-bit binary value in the first level of the accumulator stack by the 16-bit binary value in the accumulator. The result resides in the accumulator and can be 32 bits (8 digits max.). The value in the first level of the accumulator stack is removed and all stack locations are moved up one level



Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On any time the value in the accumulator is negative.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the Load instruction moves the value in V1400 into the accumulator. The value in V1420 is loaded into the accumulator using the Load instruction, pushing the value previously loaded in the accumulator onto the stack. The binary value in the accumulator stack's first level is multiplied by the binary value in the accumulator using the Multiply Binary Stack instruction. The Out Double instruction copies the value in the accumulator to V1500 and V1501.



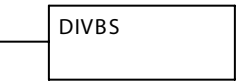
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT					
SHFT	L ANDST	D 3	→	B 1	E 4	A 0	A 0	ENT
SHFT	L ANDST	D 3	→	B 1	E 4	C 2	A 0	ENT
SHFT	M ORST	U ISG	L ANDST	B 1	S RST	ENT		
GX OUT	SHFT	D 3	→	B 1	F 5	A 0	A 0	ENT

Divide Binary by Top OF Stack (DIVBS)

DS	Used
HPP	Used

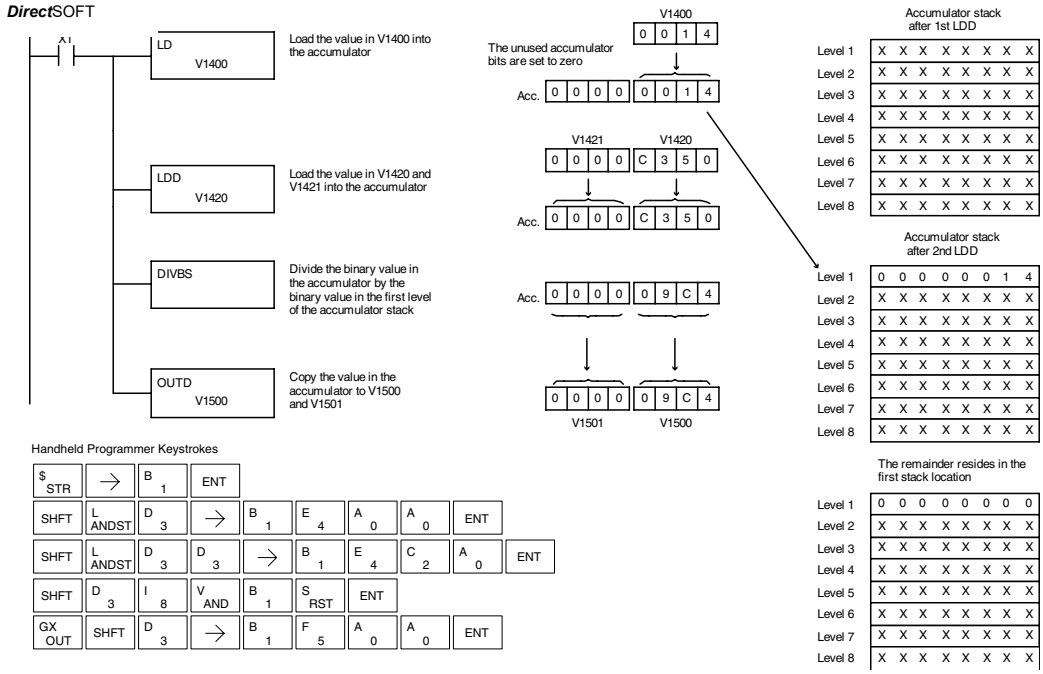
Divide Binary Top of Stack is a 32-bit instruction that divides the 32-bit binary value in the accumulator by the 16-bit binary value in the first level of the accumulator stack. The result resides in the accumulator and the remainder resides in the first level of the accumulator stack.



Discrete Bit Flags	Description
SP53	On when the value of the operand is larger than the accumulator can work with.
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On any time the value in the accumulator is negative.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V1400 will be loaded into the accumulator using the Load instruction. The value in V1420 and V1421 is loaded into the accumulator using the Load Double instruction also, pushing the value previously loaded in the accumulator onto the accumulator stack. The binary value in the accumulator is divided by the binary value in the first level of the accumulator stack using the Divide Binary Stack instruction. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.



Transcendental Functions

The DL06 CPU features special numerical functions to complement its real number capability. The transcendental functions include the trigonometric sine, cosine, and tangent, and also their inverses (arc sine, arc cosine, and arc tangent). The square root function is also grouped with these other functions.

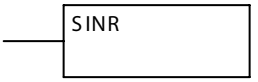
The transcendental math instructions operate on a real number in the accumulator (it cannot be BCD or binary). The real number result resides in the accumulator. The square root function operates on the full range of positive real numbers. The sine, cosine and tangent functions require numbers expressed in radians. You can work with angles expressed in degrees by first converting them to radians with the Radian (RADR) instruction, then performing the trig function. All transcendental functions utilize the following flag bits.

Discrete Bit Flags	Description
SP53	On when the value of the operand is larger than the accumulator can work with.
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.
SP72	On anytime the value in the accumulator is an invalid floating point number
SP73	On when a signed addition or subtraction results in an incorrect sign bit.
SP75	On when a real number instruction is executed and a non-real number encountered.

Sine Real (SINR)

DS	Used
HPP	N/A

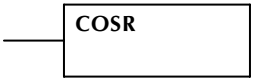
The Sine Real instruction takes the sine of the real number stored in the accumulator. The result resides in the accumulator. Both the original number and the result must be Real data type (IEEE floating point format).



Cosine Real (COSR)

DS	Used
HPP	N/A

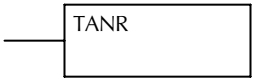
The Cosine Real instruction takes the cosine of the real number stored in the accumulator. The result resides in the accumulator. Both the original number and the result must be Real data type (IEEE floating point format)..



Tangent Real (TANR)

DS	Used
HPP	N/A

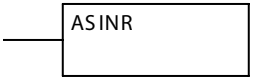
The Tangent Real instruction takes the tangent of the real number stored in the accumulator. The result resides in the accumulator. Both the original number and the result must be Real data type (IEEE floating point format).



Arc Sine Real (ASINR)

DS	Used
HPP	N/A

The Arc Sine Real instruction takes the inverse sine of the real number stored in the accumulator. The result resides in the accumulator. Both the original number and the result must be Real data type (IEEE floating point format).



Arc Cosine Real (ACOSR)

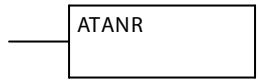
DS	Used
HPP	N/A

The Arc Cosine Real instruction takes the inverse cosine of the real number stored in the accumulator. The result resides in the accumulator. Both the original number and the result must be Real data type (IEEE floating point format).

Arc Tangent Real (ATANR)

DS	Used
HPP	N/A

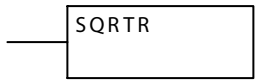
The Arc Tangent Real instruction takes the inverse tangent of the real number stored in the accumulator. The result resides in the accumulator. Both the original number and the result must be Real data type (IEEE floating point format).



Square Root Real (SQRTR)

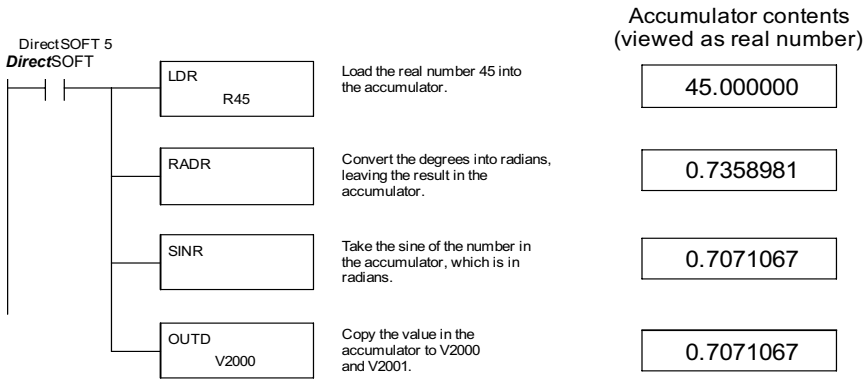
DS	Used
HPP	N/A

The Square Root Real instruction takes the square root of the real number stored in the accumulator. The result resides in the accumulator. Both the original number and the result must be Real data type (IEEE floating point format).



***NOTE**: The square root function can be useful in several situations. However, if you are trying to do the square-root extract function for an orifice flow meter measurement, as the PV to a PID loop, note that the PID loop already has the square-root extract function built in.*

The following example takes the **sine** of 45 degrees. Since these transcendental functions operate only on real numbers, we do an LDR (load real) 45. The trig functions operate only in radians, so we must convert the degrees to radians by using the RADR command. After using the SINR (Sine Real) instruction, we use an OUTD (Out Double) instruction to move the result from the accumulator to V-memory. The result is 32-bits wide, requiring the Out Double to move it.



***NOTE**: The current HPP does not support real number entry with automatic conversion to the 32-bit IEEE format. You must use **Direct**SOFT for entering real numbers, using the LDR (Load Real) instruction.*

Bit Operation Instructions

Sum (SUM)

DS	Used
HPP	Used

The Sum instruction counts number of bits that are set to "1" in the accumulator. The HEX result resides in the accumulator.

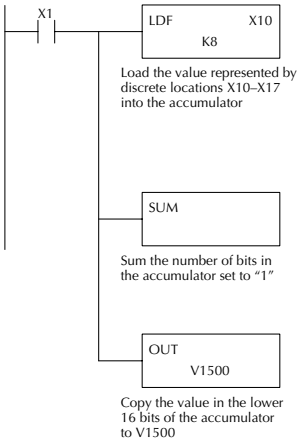
SUM

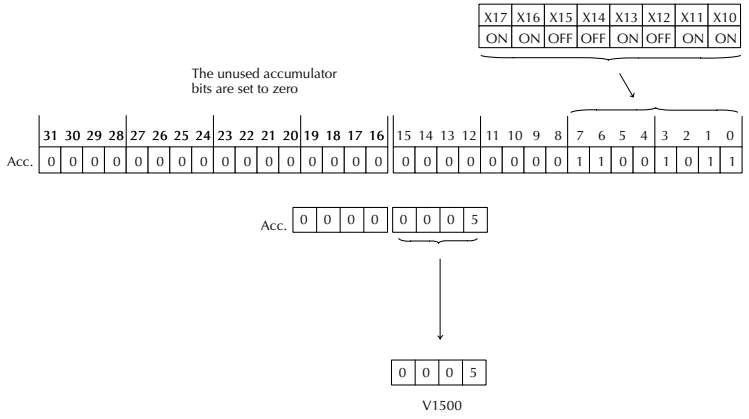
Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.

In the following example, when X1 is on, the value formed by discrete locations X10–X17 is loaded into the accumulator using the Load Formatted instruction. The number of bits in the accumulator set to "1" is counted using the Sum instruction. The value in the accumulator is copied to V1500 using the Out instruction.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

DirectSOFT





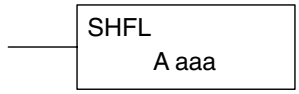
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	F 5	→	B 1	A 0	→	I 8	ENT
SHFT	S RST	SHFT	U ISG	M ORST	→	ENT			
GX OUT	→	PREV	PREV	PREV	B 1	F 5	A 0	A 0	ENT

Shift Left (SHFL)

DS	Used
HPP	Used

Shift Left is a 32-bit instruction that shifts the bits in the accumulator a specified number (Aaaa) of places to the left. The vacant positions are filled with zeros and the bits shifted out of the accumulator are discarded.



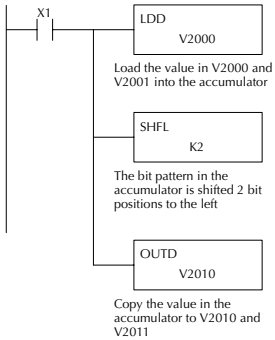
Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Constant	K	1-32

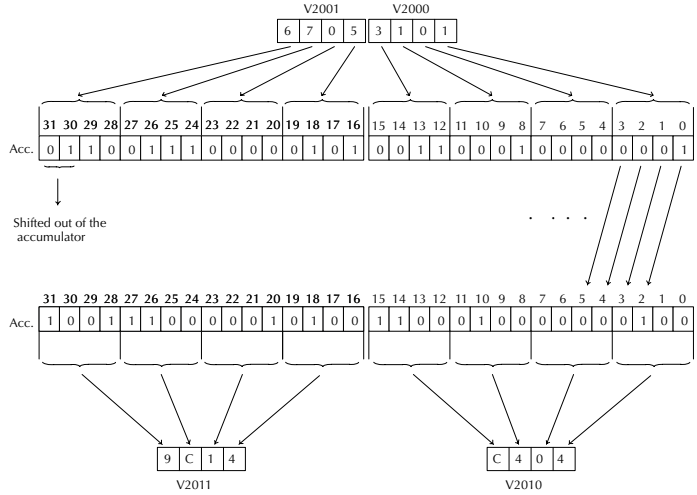
Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.

In the following example, when X1 is on, the value in V2000 and V2001 will be loaded into the accumulator using the Load Double instruction. The bit pattern in the accumulator is shifted 2 bits to the left using the Shift Left instruction. The value in the accumulator is copied to V2010 and V2011 using the Out Double instruction.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

***Direct**SOFT*





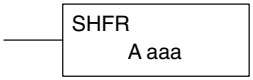
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	D 3	→	C 2	A 0	A 0	A 0	ENT
SHFT	S RST	SHFT	H 7	F 5	L ANDST	→	C 2	ENT	
GX OUT	SHFT	D 3	→	C 2	A 0	B 1	A 0	ENT	

Shift Right (SHFR)

DS	Used
HPP	Used

Shift Right is a 32-bit instruction that shifts the bits in the accumulator a specified number (Aaaa) of places to the right. The vacant positions are filled with zeros and the bits shifted out of the accumulator are lost.



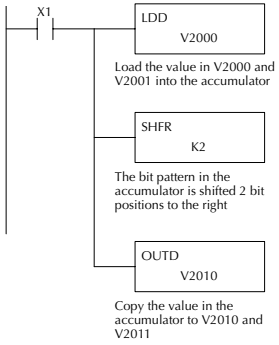
Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Constant	K	1-32

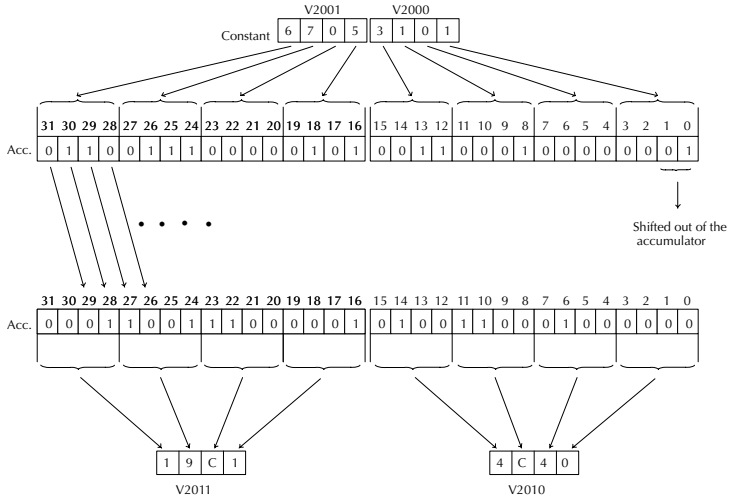
Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.

In the following example, when X1 is on, the value in V2000 and V2001 will be loaded into the accumulator using the Load Double instruction. The bit pattern in the accumulator is shifted 2 bits to the right using the Shift Right instruction. The value in the accumulator is copied to V2010 and V2011 using the Out Double instruction.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*

DirectSOFT





Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	D 3	→	C 2	A 0	A 0	A 0	ENT
SHFT	S RST	SHFT	H 7	F 5	R ORN	→	C 2	ENT	
GX OUT	SHFT	D 3	→	C 2	A 0	B 1	A 0	ENT	

Rotate Left (ROTL)

DS	Used
HPP	Used

Rotate Left is a 32-bit instruction that rotates the bits in the accumulator a specified number (Aaaa) of places to the left.

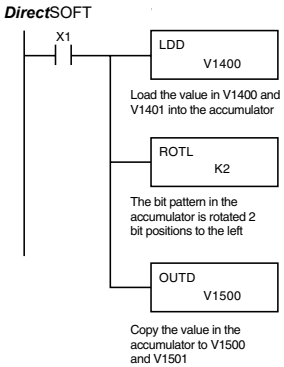
```
ROTL
 A aaa
```

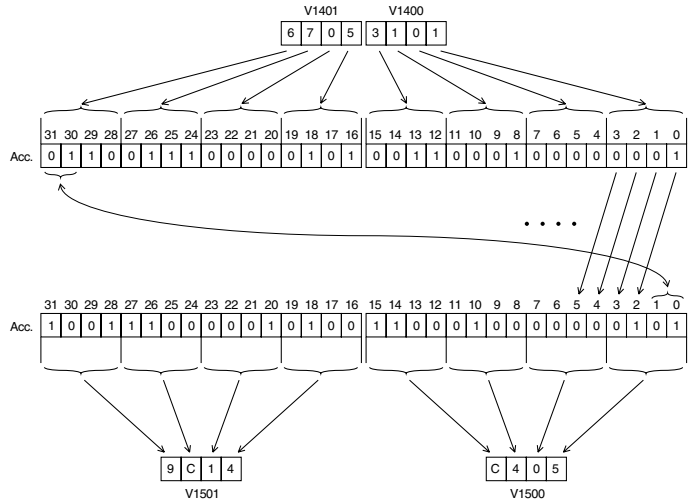
Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Constant	K	1-32

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.

In the following example, when X1 is on, the value in V1400 and V1401 will be loaded into the accumulator using the Load Double instruction. The bit pattern in the accumulator is rotated 2 bit positions to the left using the Rotate Left instruction. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.

***NOTE**: Status flags are valid only until another instruction uses the same flag.*





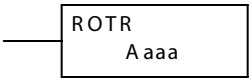
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	D 3	→	B 1	E 4	A 0	A 0	ENT
SHFT	R ORN	O INST#	T MLR	L ANDST	→	C 2	ENT		
GX OUT	SHFT	D 3	→	B 1	F 5	A 0	A 0	ENT	

Rotate Right (ROTR)

DS	Used
HPP	Used

Rotate Right is a 32-bit instruction that rotates the bits in the accumulator a specified number (Aaaa) of places to the right.

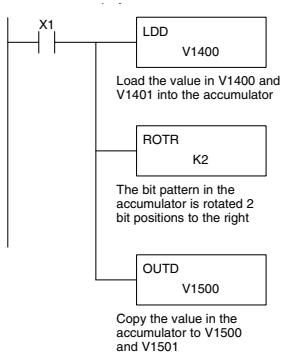


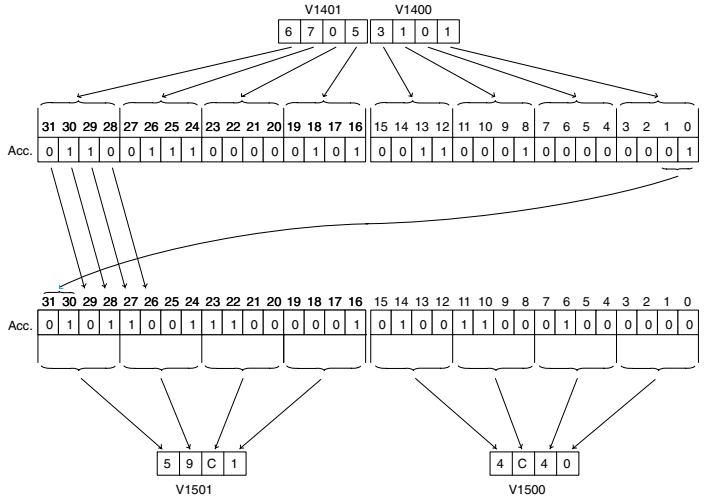
Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Constant	K	1-32

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.

In the following example, when X1 is on, the value in V1400 and V1401 will be loaded into the accumulator using the Load Double instruction. The bit pattern in the accumulator is rotated 2 bit positions to the right using the Rotate Right instruction. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.

DirectSOFT





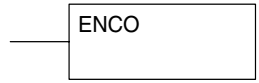
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	D 3	→	B 1	E 4	A 0	A 0	ENT
SHFT	R ORN	O INST#	T MLR	R ORN	→	C 2	ENT		
GX OUT	SHFT	D 3	→	B 1	F 5	A 0	A 0	ENT	

Encode (ENCO)

DS	Used
HPP	Used

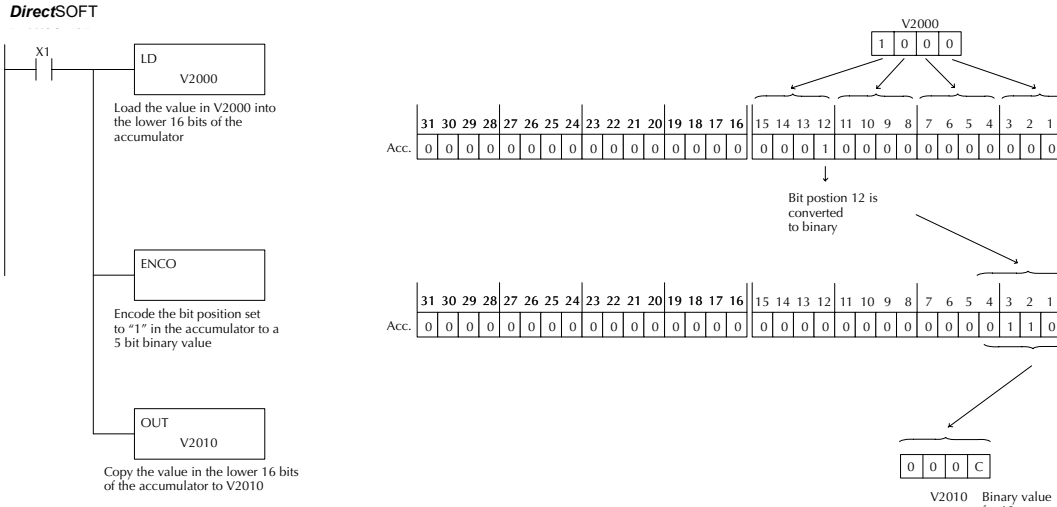
The Encode instruction encodes the bit position in the accumulator having a value of 1, and returns the appropriate binary representation. If the most significant bit is set to 1 (Bit 31), the Encode instruction would place the value HEX 1F (decimal 31) in the accumulator. If the value to be encoded is 0000 or 0001, the instruction will place a zero in the accumulator. If the value to be encoded has more than one bit position set to a "1", the least significant "1" will be encoded and SP53 will be set on.



Discrete Bit Flags	Description
SP53	On when the value of the operand is larger than the accumulator can work with.

***NOTE**: The status flags are only valid until another instruction that uses the same flags is executed.*

In the following example, when X1 is on, The value in V2000 is loaded into the accumulator using the Load instruction. The bit position set to a "1" in the accumulator is encoded to the corresponding 5 bit binary value using the Encode instruction. The value in the lower 16 bits of the accumulator is copied to V2010 using the Out instruction.



Handheld Programmer Keystrokes

$ STR	→	B 1	ENT					
SHFT	L ANDST	D 3	→	C 2	A 0	A 0	A 0	ENT
SHFT	E 4	N TMR	C 2	O INST#	ENT			
GX OUT	→	SHFT	V AND	C 2	A 0	B 1	A 0	ENT

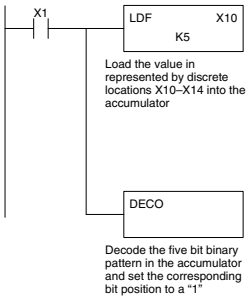
Decode (DECO)

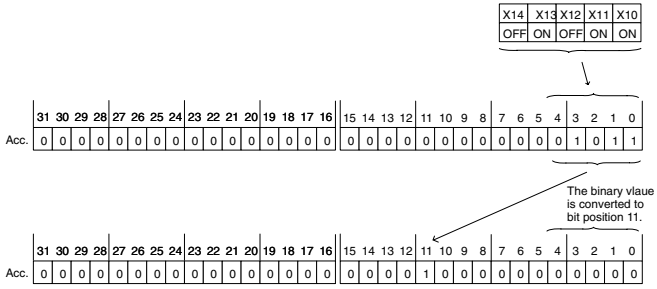
DS	Used
HPP	Used

The Decode instruction decodes a 5-bit binary value of 0–31 (0–1Fh) in the accumulator by setting the appropriate bit position to a 1. If the accumulator contains the value Fh (HEX), bit 15 will be set in the accumulator. If the value to be decoded is greater than 31, the number is divided by 32 until the value is less than 32 and then the value is decoded.

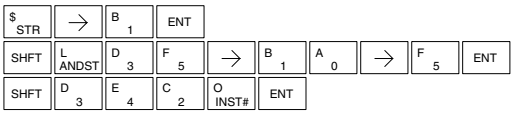
In the following example when X1 is on, the value formed by discrete locations X10–X14 is loaded into the accumulator using the Load Formatted instruction. The 5- bit binary pattern in the accumulator is decoded by setting the corresponding bit position to a "1" using the Decode instruction.

DirectSOFT





Handheld Programmer Keystrokes

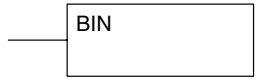


Number Conversion Instructions (Accumulator)

Binary (BIN)

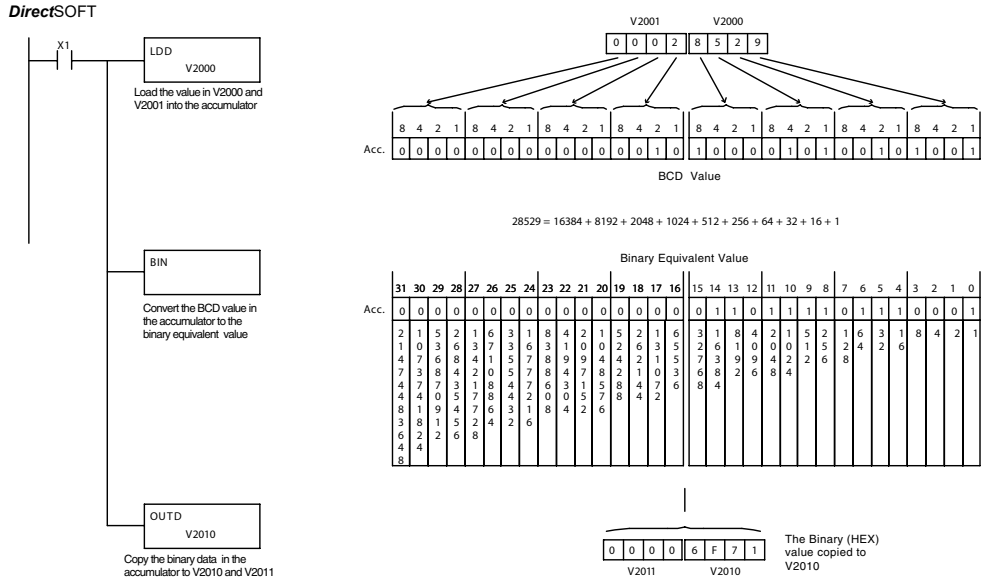
DS	Used
HPP	Used

The Binary instruction converts a BCD value in the accumulator to the equivalent binary, or decimal, value. The result resides in the accumulator.

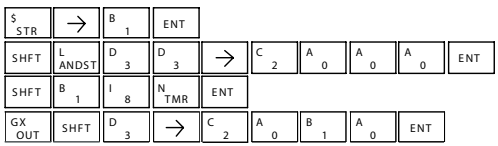


Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.
SP75	On when a BCD instruction is executed and a NON–BCD number was encountered.

In the following example, when X1 is on, the value in V2000 and V2001 is loaded into the accumulator using the Load Double instruction. The BCD value in the accumulator is converted to the binary (HEX) equivalent using the BIN instruction. The binary value in the accumulator is copied to V2010 and V2011 using the Out Double instruction. (The handheld programmer will display the binary value in V2010 and V2011 as a HEX value.)



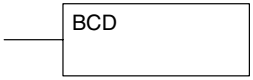
Handheld Programmer Keystrokes



Binary Coded Decimal (BCD)

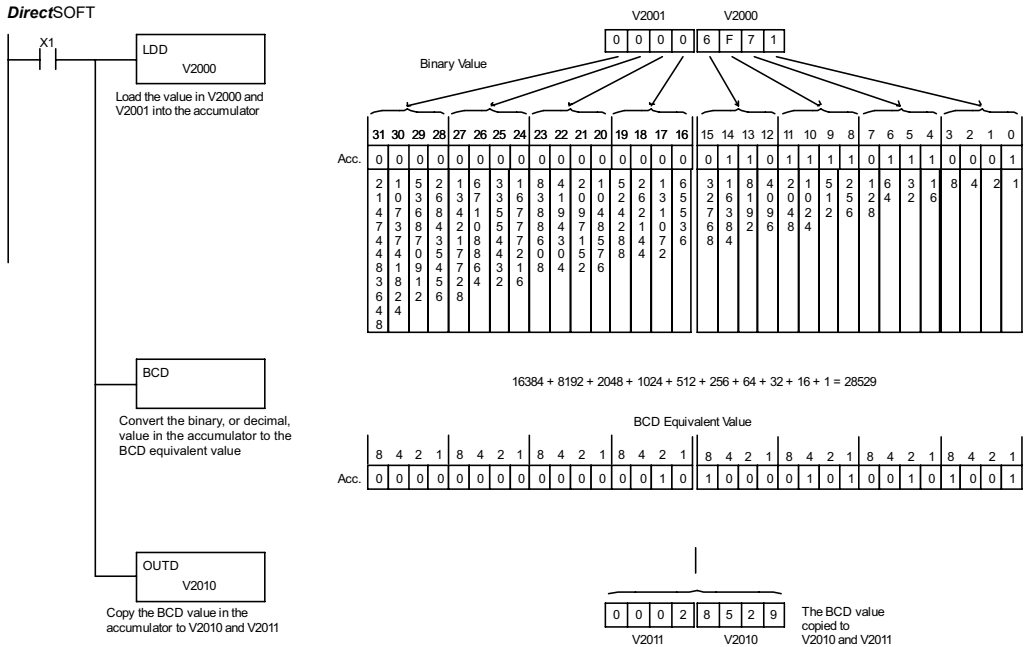
DS	Used
HPP	Used

The Binary Coded Decimal instruction converts a binary, or decimal, value in the accumulator to the equivalent BCD value. The result resides in the accumulator.



Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.

In the following example, when X1 is on, the binary, or decimal, value in V2000 and V2001 is loaded into the accumulator using the Load Double instruction. The value in the accumulator is converted to the BCD equivalent value using the BCD instruction. The BCD value in the accumulator is copied to V2010 and V2011 using the Out Double instruction.



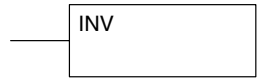
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	D 3	→	C 2	A 0	A 0	A 0	ENT
SHFT	B 1	C 2	D 3	ENT					
GX OUT	SHFT	D 3	→	C 2	A 0	B 1	A 0	ENT	

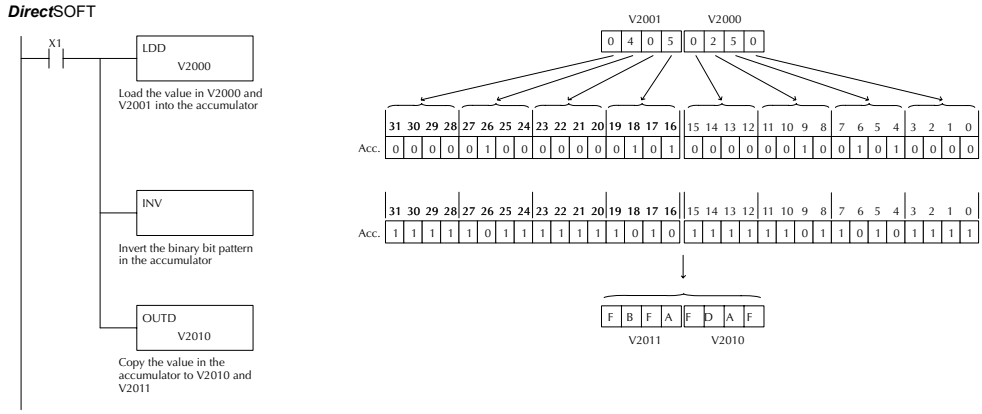
Invert (INV)

DS	Used
HPP	Used

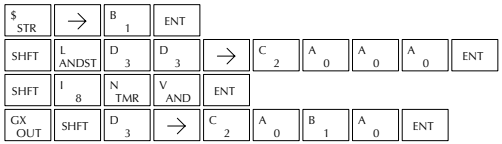
The Invert instruction inverts or takes the one's complement of the 32-bit value in the accumulator. The result resides in the accumulator.



In the following example, when X1 is on, the value in V2000 and V2001 will be loaded into the accumulator using the Load Double instruction. The value in the accumulator is inverted using the Invert instruction. The value in the accumulator is copied to V2010 and V2011 using the Out Double instruction.



Handheld Programmer Keystrokes



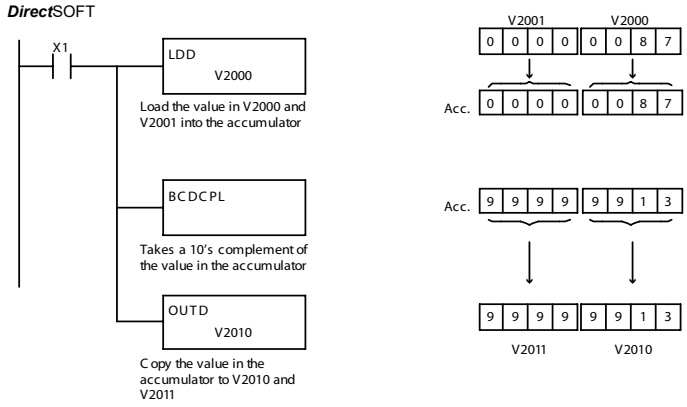
Ten's Complement (BCDCPL)

DS	Used
HPP	Used

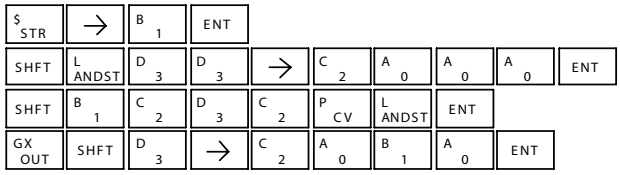
The Ten's Complement instruction takes the 10's complement (BCD) of the 8 digit accumulator. The result resides in the accumulator. The calculation for this instruction is:

100000000
– accumulator
10's complement value

In the following example when X1 is on, the value in V2000 and V2001 is loaded into the accumulator. The 10's complement is taken for the 8 digit accumulator using the Ten's Complement instruction. The value in the accumulator is copied to V2010 and V2011 using the Out Double instruction.



Handheld Programmer Keystrokes



Binary to Real Conversion (BTOR)

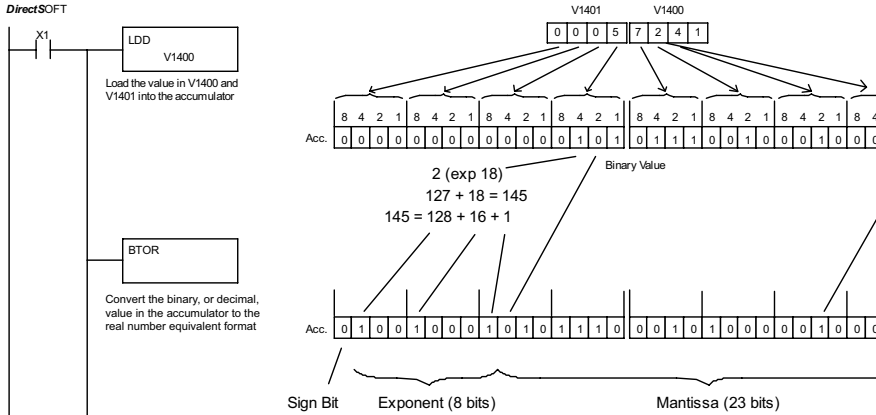
DS	Used
HPP	Used

The Binary-to-Real instruction converts a binary, or decimal, value in the accumulator to its equivalent real number (floating point) format. The result resides in the accumulator. Both the binary and the real number may use all 32 bits of the accumulator.

***NOTE**: This instruction only works with unsigned **binary, or decimal,** values. It will not work with signed decimal values.*

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.

In the following example, when X1 is on, the value in V1400 and V1401 is loaded into the accumulator using the Load Double instruction. The BTOR instruction converts the binary, or decimal, value in the accumulator to the equivalent real number format. The binary weight of the MSB is converted to the real number exponent by adding it to 127 (decimal). Then the remaining bits are copied to the mantissa as shown. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction. The handheld programmer would display the binary value in V1500 and V1501 as a HEX value.



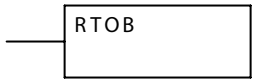
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	D 3	→	B 1	E 4	A 0	A 0	ENT
SHFT	B 1	T MLR	O INST#	R ORN	ENT				
GX OUT	SHFT	D 3	→	B 1	F 5	A 0	A 0	ENT	

Real to Binary Conversion (RTOB)

DS	Used
HPP	Used

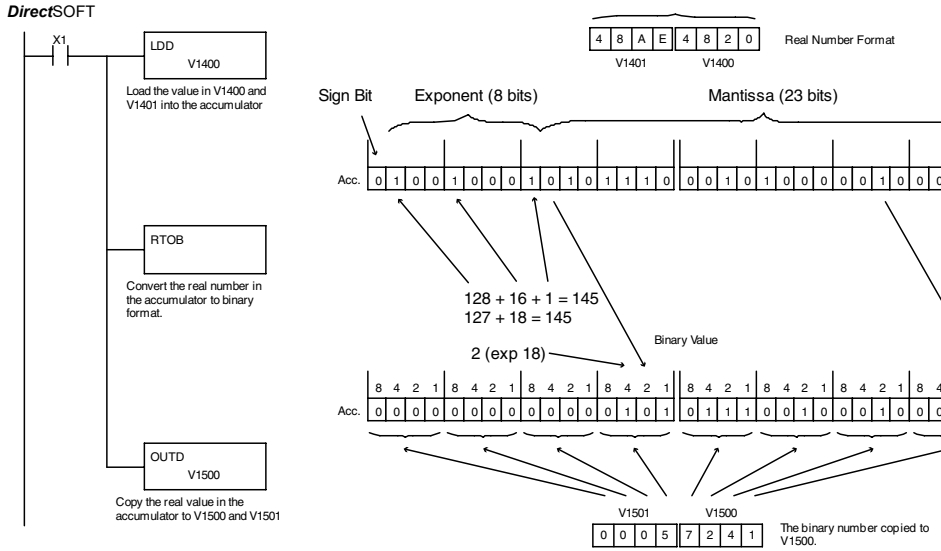
The Real-to-Binary instruction converts the real number in the accumulator to a binary value. The result resides in the accumulator. Both the binary and the real number may use all 32 bits of the accumulator.



NOTE$_1$: *The decimal portion of the result will be rounded down (14.1 to 14; -14.1 to -15).*
NOTE$_2$: *if the real number is negative, it becomes a signed decimal value.*

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.
SP72	On anytime the value in the accumulator is an invalid floating point number.
SP73	On when a signed addition or subtraction results in an incorrect sign bit.
SP75	On when a number cannot be converted to binary.

In the following example, when X1 is on, the value in V1400 and V1401 is loaded into the accumulator using the Load Double instruction. The RTOB instruction converts the real value in the accumulator the equivalent binary number format. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction. The handheld programmer would display the binary value in V1500 and V1501 as a HEX value.



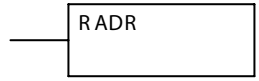
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	D 3	→	B 1	E 4	A 0	A 0	ENT
SHFT	R ORN	T MLR	O INST#	B 1	ENT				
GX OUT	SHFT	D 3	→	B 1	F 5	A 0	A 0	ENT	

Radian Real Conversion (RADR)

DS	Used
HPP	N/A

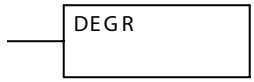
The Radian Real Conversion instruction converts the real degree value stored in the accumulator to the equivalent real number in radians. The result resides in the accumulator.



Degree Real Conversion (DEGR)

DS32	Used
HPP	N/A

The Degree Real instruction converts the degree real radian value stored in the accumulator to the equivalent real number in degrees. The result resides in the accumulator.

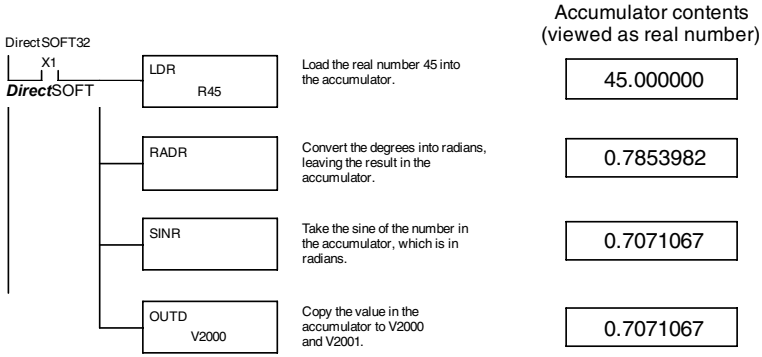


Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.
SP72	On anytime the value in the accumulator is an invalid floating point number.
SP73	On when a signed addition or subtraction results in an incorrect sign bit.
SP75	On when a number cannot be converted to binary.

The two instructions described above convert real numbers into the accumulator from degree format to radian format, and vice-versa. In degree format, a circle contains 360 degrees. In radian format, a circle contains 2π (about 6.28) radians. These convert between both positive and negative real numbers, and for angles greater than a full circle. These functions are very useful when combined with the transcendental trigonometric functions (see the section on math instructions).

***NOTE**: The current HPP does not support real number entry with automatic conversion to the 32-bit IEEE format. You must use **Direct**SOFT for entering real numbers, using the LDR (Load Real) instruction.*

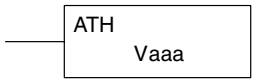
The following example takes the sine of 45 degrees. Since transcendental functions operate only on real numbers, we do an LDR (load real) 45. The trig functions operate only in radians, so we must convert the degrees to radians by using the RADR command. After using the SINR (Sine Real) instruction, we use an OUTD (Out Double) instruction to move the result from the accumulator to V-memory. The result is 32-bits wide, requiring the Out Double to move it.



ASCII to HEX (ATH)

DS	Used
HPP	N/A

The ASCII TO HEX instruction converts a table of ASCII values to a specified table of HEX values. ASCII values are two digits and their HEX equivalents are one digit. This means an ASCII table of four V-memory locations would only require two V-memory locations for the equivalent HEX table. The function parameters are loaded into the accumulator stack and the accumulator by two additional instructions. Listed below are the steps necessary to program an ASCII to HEX table function. The example on the following page shows a program for the ASCII to HEX table function.



Step 1: Load the number of V-memory locations for the ASCII table into the first level of the accumulator stack.

Step 2: Load the starting V-memory location for the ASCII table into the accumulator. This parameter must be a HEX value.

Step 3: Specify the starting V-memory location (Vaaa) for the HEX table in the ATH instruction.

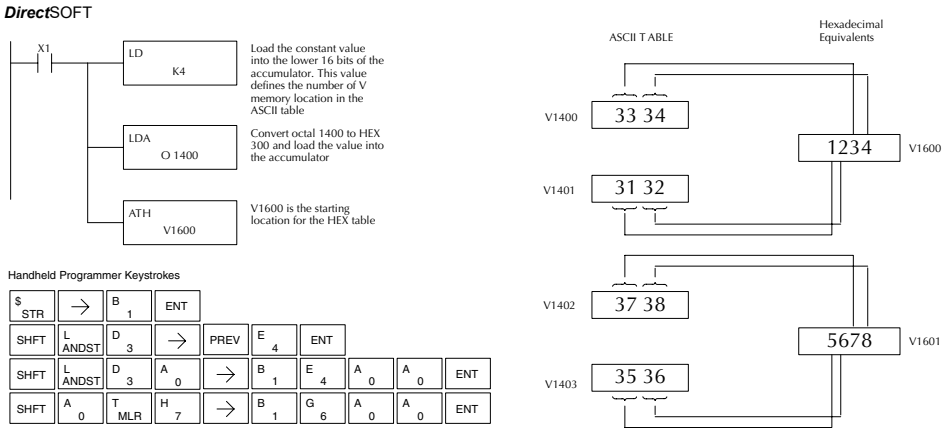
Helpful Hint: — For parameters that require HEX values when referencing memory locations, the LDA instruction can be used to convert an octal address to the HEX equivalent and load the value into the accumulator.

Operand Data Type		DL06 Range
		aaa
V-memory	V	See memory map

Discrete Bit Flags	Description
SP53	On when the value of the operand is larger than the accumulator can work with.

In the example on the following page, when X1 is ON the constant (K4) is loaded into the accumulator using the Load instruction and will be placed in the first level of the accumulator stack when the next Load instruction is executed. The starting location for the ASCII table (V1400) is loaded into the accumulator using the Load Address instruction. The starting location for the HEX table (V1600) is specified in the ASCII to HEX instruction. The table below lists valid ASCII values for ATH conversion.

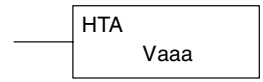
ASCII Values Valid for ATH Conversion			
ASCII Value	Hex Value	ASCII Value	Hex Value
30	0	38	8
31	1	39	9
32	2	41	A
33	3	42	B
34	4	43	C
35	5	44	D
36	6	45	E
37	7	46	F



HEX to ASCII (HTA)

DS	Used
HPP	N/A

The HEX to ASCII instruction converts a table of HEX values to a specified table of ASCII values. HEX values are one digit and their ASCII equivalents are two digits.



This means a HEX table of two V-memory locations would require four V-memory locations for the equivalent ASCII table. The function parameters are loaded into the accumulator stack and the accumulator by two additional instructions. Listed below are the steps necessary to program a HEX to ASCII table function. The example on the following page shows a program for the HEX to ASCII table function.

Step 1: Load the number of V-memory locations in the HEX table into the first level of the accumulator stack.

Step 2: Load the starting V-memory location for the HEX table into the accumulator. This parameter must be a HEX value.

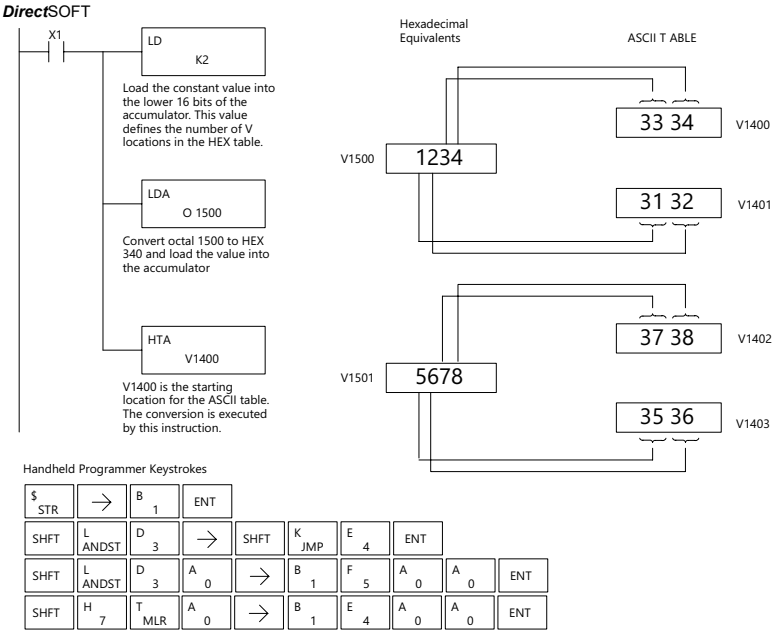
Step 3:Specify the starting V-memory location (Vaaa) for the ASCII table in the HTA instruction.

Helpful Hint: — For parameters that require HEX values when referencing memory locations, the LDA instruction can be used to convert an octal address to the HEX equivalent and load the value into the accumulator.

Operand Data Type		DL06 Range
		aaa
V-memory	V	See memory map

Discrete Bit Flags	Description
SP53	On when the value of the operand is larger than the accumulator can work with.

In the following example, when X1 is ON, the constant (K2) is loaded into the accumulator using the Load instruction. The starting location for the HEX table (V1500) is loaded into the accumulator using the Load Address instruction. The starting location for the ASCII table (V1400) is specified in the HEX to ASCII instruction.



The table below lists valid ASCII values for HTA conversion.

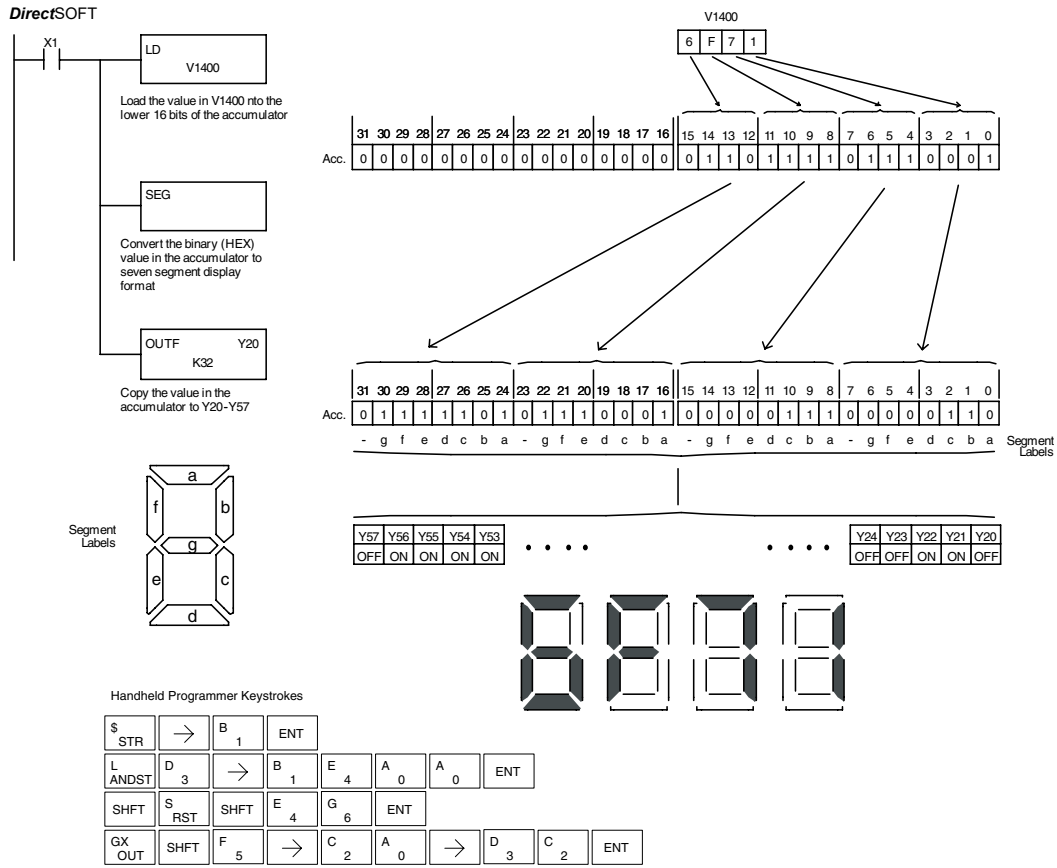
ASCII Values Valid for HTA Conversion			
Hex Value	ASCII Value	Hex Value	ASCII Value
0	30	8	38
1	31	9	39
2	32	A	41
3	33	B	42
4	34	C	43
5	35	D	44
6	36	E	45
7	37	F	46

Segment (SEG)

DS	Used
HPP	Used

The BCD / Segment instruction converts a four digit HEX value in the accumulator to seven segment display format. The result resides in the accumulator.

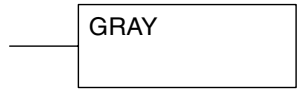
In the following example, when X1 is on, the value in V1400 is loaded into the lower 16 bits of the accumulator using the Load instruction. The HEX value in the accumulator is converted to seven segment format using the Segment instruction. The bit pattern in the accumulator is copied to Y20–Y57 using the Out Formatted instruction.



Gray Code (GRAY)

DS	Used
HPP	Used

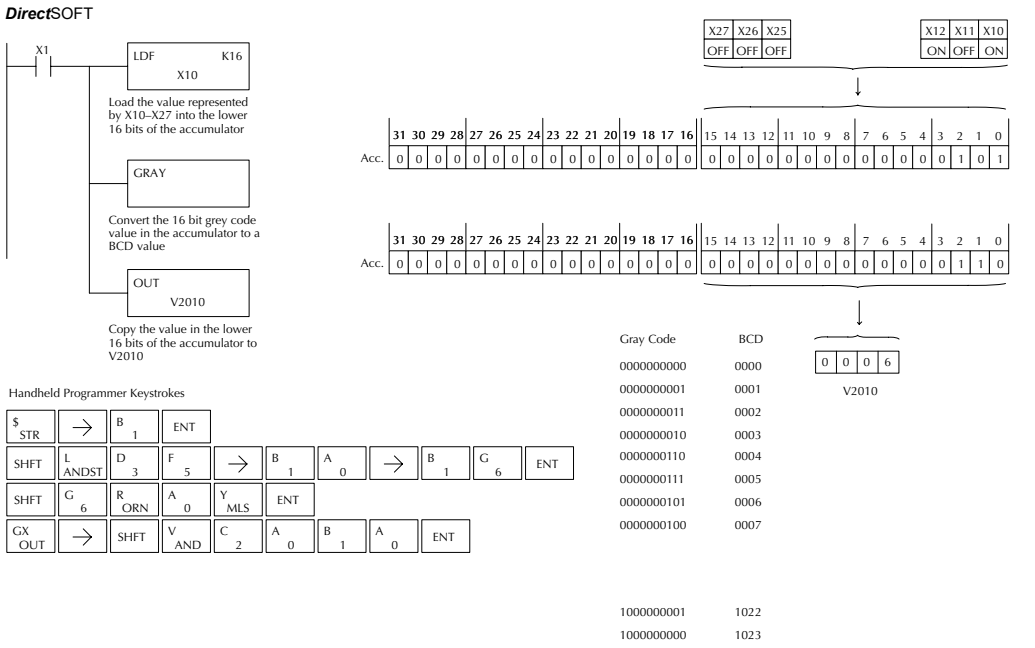
The Gray code instruction converts a 16-bit gray code value to a BCD value. The BCD conversion requires 10 bits of the accumulator. The upper 22 bits are set to "0"



This instruction is designed for use with devices (typically encoders) that use the gray code numbering scheme. The Gray Code instruction will directly convert a gray code number to a BCD number for devices having a resolution of 512 or 1024 counts per revolution. If a device having a resolution of 360 counts per revolution is to be used, you must subtract a BCD value of 76 from the converted value to obtain the proper result. For a device having a resolution of 720 counts per revolution, you must subtract a BCD value of 152.

In the following example, when X1 is ON, the binary value represented by X10–X27 is loaded into the accumulator using the Load Formatted instruction. The gray code value in the accumulator is converted to BCD using the Gray Code instruction. The value in the lower 16 bits of the accumulator is copied to V2010.

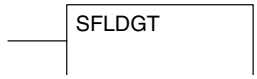
Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.



Shuffle Digits (SFLDGT)

DS	Used
HPP	Used

The Shuffle Digits instruction shuffles a maximum of 8 digits, rearranging them in a specified order. This function requires parameters to be loaded into the first level of the accumulator stack and the accumulator with two additional instructions. Listed below are the steps necessary to use the shuffle digit function. The example on the following page shows a program for the Shuffle Digits function.



Step 1: Load the value (digits) to be shuffled into the first level of the accumulator stack.

Step 2: Load the order that the digits will be shuffled to into the accumulator.

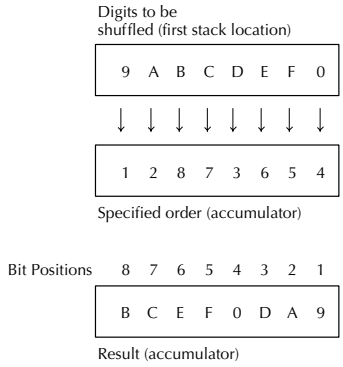
Step 3: Insert the SFLDGT instruction.

***NOTE**: If the number used to specify the order contains a 0 or 9–F, the corresponding position will be set to 0.*

Shuffle Digits Block Diagram

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.

There are a maximum of 8 digits that can be shuffled. The bit positions in the first level of the accumulator stack define the digits to be shuffled. They correspond to the bit positions in the accumulator that define the order the digits will be shuffled. The digits are shuffled and the result resides in the accumulator.

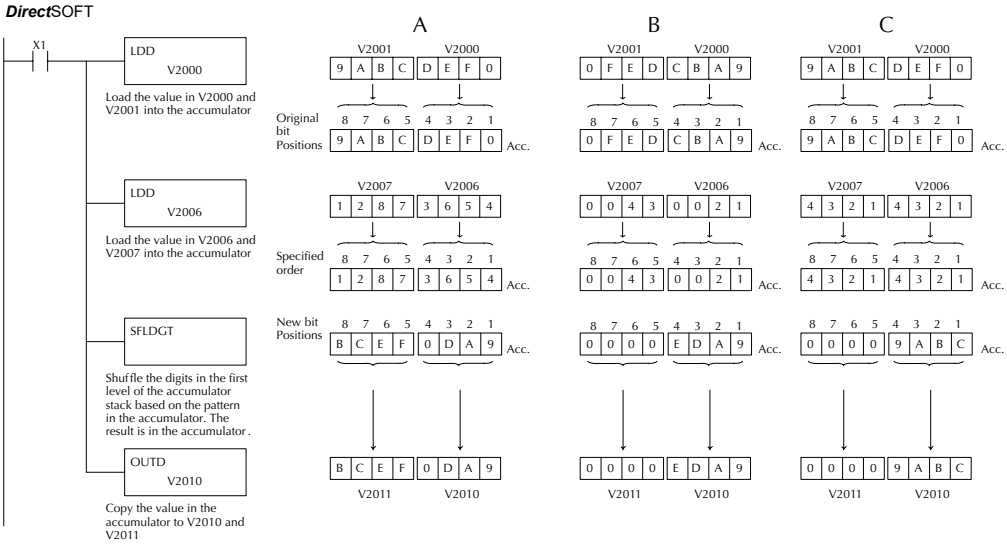


In the following example, when X1 is on, the value in the first level of the accumulator stack will be reorganized in the order specified by the value in the accumulator.

Example A shows how the shuffle digits works when 0 or 9–F is not used when specifying the order the digits are to be shuffled. Also, there are no duplicate numbers in the specified order.

Example B shows how the Shuffle Digits works when a 0 or 9–F is used when specifying the order the digits are to be shuffled. Notice when the Shuffle Digits instruction is executed, the bit positions in the first stack location that had a corresponding 0 or 9–F in the accumulator (order specified) are set to "0".

Example C shows how the Shuffle Digits works when duplicate numbers are used specifying the order the digits are to be shuffled. Notice when the Shuffle Digits instruction is executed, the most significant duplicate number in the order specified is used in the result.



Handheld Programmer Keystrokes

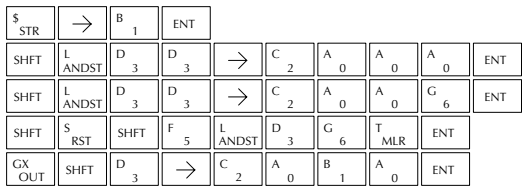
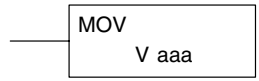


Table Instructions

Move (MOV)

DS	Used
HPP	Used

The Move instruction moves the values from a V-memory table to another V-memory table the same length (a table being a consecutive group of V-memory locations). The function parameters are loaded into the first level of the accumulator stack and the accumulator by two additional instructions. The MOV instruction can be used to write data to non-volatile V-memory (see Appendix F). Listed below are the steps necessary to program the MOV function.



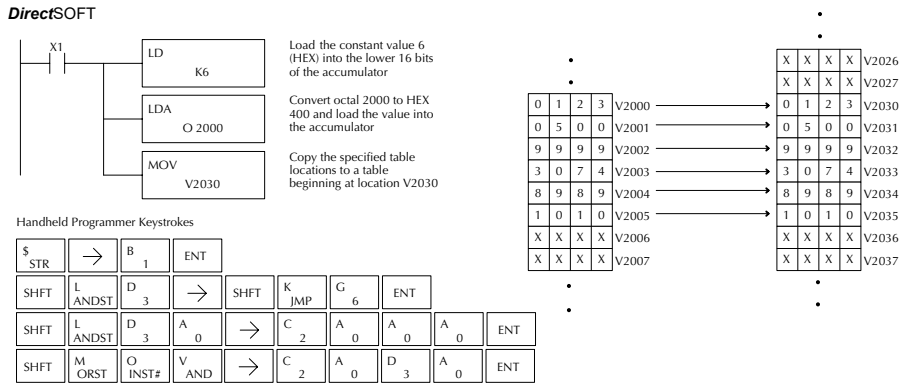
- Step 1 Load the number of V-memory locations to be moved into the first level of the accumulator stack. This parameter is a HEX value (KFFF max, 7777 octal, 4096 decimal).
- Step 2 Load the starting V-memory location for the locations to be moved into the accumulator. This parameter is a HEX value.
- Step 3 Insert the MOV instruction which specifies starting V-memory location (Vaaa) for the destination table.

Helpful Hint: — For parameters that require HEX values when referencing memory locations, the LDA instruction can be used to convert an octal address to the HEX equivalent and load the value into the accumulator.

Operand Data Type		DL06 Range
		aaa
V-memory	V	See memory map
Pointer	P	See memory map

Discrete Bit Flags	Description
SP53	On when the value of the operand is larger than the accumulator can work with.

In the following example, when X1 is on, the constant value (K6) is loaded into the accumulator using the Load instruction. This value specifies the length of the table and is placed in the first stack location after the Load Address instruction is executed. The octal address 2000 (V2000), the starting location for the source table, is loaded into the accumulator. The destination table location (V2030) is specified in the Move instruction.

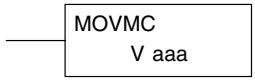


Move Memory Cartridge (MOVMC)

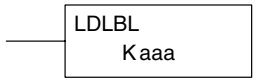
Load Label (LDLBL)

DS	Used
HPP	Used

The Move Memory Cartridge instruction is used to copy data between V-memory and program ladder memory. The Load Label instruction is *only* used with the MOVMC instruction when copying data *from* program ladder memory to V-memory.



To copy data between V-memory and program ladder memory, the function parameters are loaded into the first two levels of the accumulator stack and the accumulator by two additional instructions. Listed below are the steps necessary to program the MOVMC and LDLBL functions.



- Step 1: Load the number of words to be copied into the second level of the accumulator stack.
- Step 2: Load the offset for the data label area in ladder memory and the beginning of the V-memory block into the first level of the stack.
- Step 3: Load the *source data label* (LDLBL Kaaa) into the accumulator when copying data from ladder memory to V-memory. Load the *source address* into the accumulator when copying data from V-memory to ladder memory. This is where the value will be copied from. If the source address is a V-memory location, the value must be entered in HEX.
- Step 4: Insert the MOVMC instruction which specifies destination in V-memory (Vaaa). This is the copy destination.

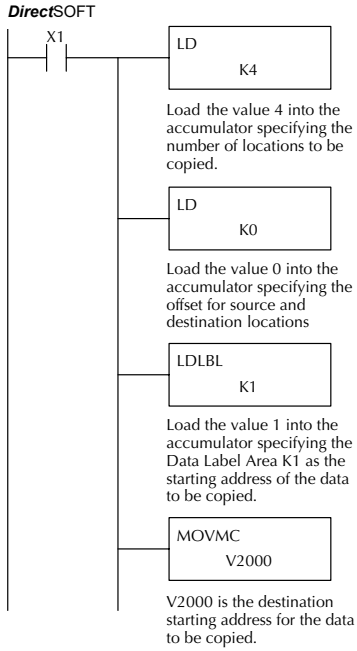
Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map

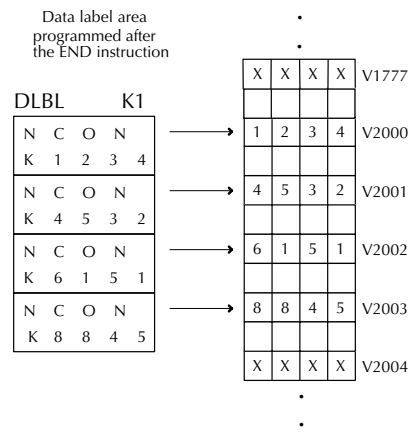
NOTE: *Refer to page 5-188 for an example.*

WARNING: The offset for this usage of the instruction starts at 0, but may be any number that does not result in data outside of the source data area being copied into the destination table. When an offset is outside of the source information boundaries, then unknown data values will be transferred into the destination table.

Copy Data From a Data Label Area to V-memory

In the example below, data is copied from a Data Label Area to V-memory. When X1 is on, the constant value (K4) is loaded into the accumulator using the Load (LD) instruction. This value specifies the length of the table and is placed in the second stack location after the next Load and Load Label (LDLBL) instructions are executed. The constant value (K0) is loaded into the accumulator, specifying the offset for the source and destination data. It is placed in the first stack location after the LDLBL instruction is executed. The source address where data is being copied from is loaded into the accumulator using the LDLBL instruction. The MOVMC instruction specifies the destination starting location and executes the copying of data from the Data Label Area to V-memory.





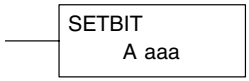
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT																
SHFT	L ANDST	D 3	→	SHFT	K JMP	E 4	ENT												
SHFT	L ANDST	D 3	→	SHFT	K JMP	A 0	ENT												
SHFT	L ANDST	D 3	L ANDST	B 1	L ANDST	→	B 1	ENT											
SHFT	M ORST	O INST#	V AND	M ORST	C 2	→	C 2	A 0	A 0	A 0	ENT								

SETBIT

DS	Used
HPP	Used

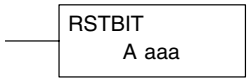
The Set Bit instruction sets a single bit to one within a range of V-memory locations.



RSTBIT

DS	Used
HPP	Used

The Reset Bit instruction resets a single bit to zero within a range of V-memory locations.



The following description applies to both the Set Bit and Reset Bit table instructions.

Step 1: Load the length of the table (number of V-memory locations) into the first level of the accumulator stack. This parameter must be a HEX value, 0 to FF.

Step 2: Load the starting V-memory location for the table into the accumulator. This parameter must be a HEX value. You can use the LDA instruction to convert an octal address to hex.

Step 3: Insert the Set Bit or Reset Bit instruction. This specifies the reference for the bit number of the bit you want to set or reset. The bit number is in octal, and the first bit in the table is number "0".

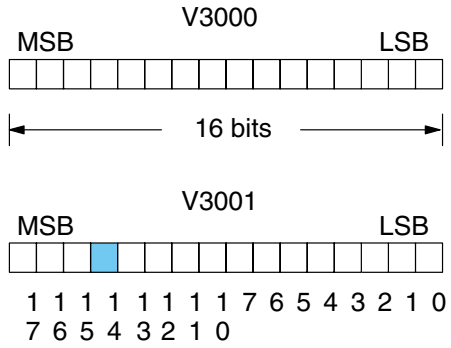
Helpful hint: — Remember that each V-memory location contains 16 bits. So, the bits of the first word of the table are numbered from 0 to 17 octal. For example, if the table length is six words, then 6 words = (6 x 16) bits, = 96 bits (decimal), or 140 octal. The permissible range of bit reference numbers would be 0 to 137 octal. SP 53 will be set if the bit specified is outside the range of the table.

Operand Data Type		DL06 Range
		aaa
V-memory	V	See memory map

Discrete Bit Flags	Description
SP53	On when the specified bit is outside the range of the table.

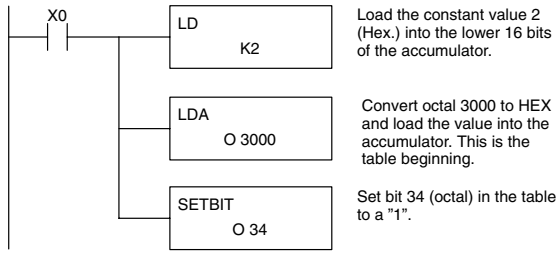
***NOTE**: Status flags are only valid until the end of the scan or until another instruction that uses the same flag is executed.*

For example, suppose we have a table starting at V3000 that is two words long, as shown to the right. Each word in the table contains 16 bits, or 0 to 17 in octal. To set bit 12 in the second word, we use its octal reference (bit 14). Then we compute the bit's octal address from the start of the table, so 17 + 14 = 34 octal. The following program shows how to set the bit as shown to a "1".



In this ladder example, we will use input X0 to trigger the Set Bit operation. First, we will load the table length (2 words) into the accumulator stack. Next, we load the starting address into the accumulator. Since V3000 is an octal number, we have to convert it to hex by using the LDA command. Finally, we use the Set Bit (or Reset Bit) instruction and specify the octal address of the bit (bit 34), referenced from the table.

DirectSOFT



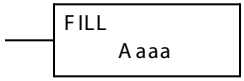
Handheld Programmer Keystrokes

$ STR	→	A 0	ENT						
SHFT	L ANDST	D 3	→	PREV	C 2	ENT			
SHFT	L ANDST	D 3	A 0	→	D 3	A 0	A 0	A 0	ENT
X SET	SHFT	B 1	I 8	T MLR	NEXT	D 3	E 4	ENT	

Fill (FILL)

DS	Used
HPP	Used

The Fill instruction fills a table of up to 255 V-memory locations with a value (Aaaa), which is either a V-memory location or a 4-digit constant. The function parameters are loaded into the first level of the accumulator stack and the accumulator by two additional instructions



Listed below are the steps necessary to program the Fill function.

Step 1: Load the number of V-memory locations to be filled into the first level of the accumulator stack. This parameter must be a HEX value, 0–FF.

Step 2: Load the starting V-memory location for the table into the accumulator. This parameter must be a HEX value.

Step 3: Insert the Fill instruction which specifies the value to fill the table with.

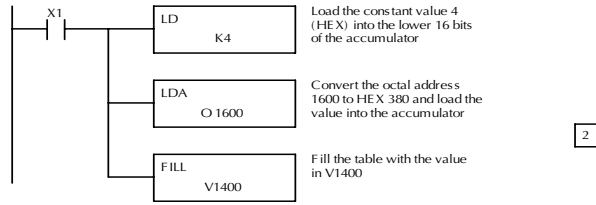
Helpful Hint: — For parameters that require HEX values when referencing memory locations, the LDA instruction can be used to convert an octal address to the HEX equivalent and load the value into the accumulator.

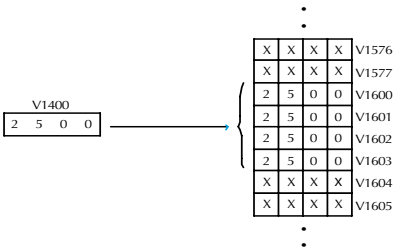
Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	K	0–FF

Discrete Bit Flags	Description
SP53	On if the V-memory address is out of range.

In the following example, when X1 is on, the constant value (K4) is loaded into the accumulator using the Load instruction. This value specifies the length of the table and is placed on the first level of the accumulator stack when the Load Address instruction is executed. The octal address 1600 (V1600) is the starting location for the table and is loaded into the accumulator using the Load Address instruction. The value to fill the table with (V1400) is specified in the Fill instruction.

*Direct*SOFT





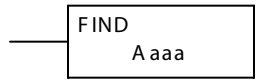
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	→	PREV	E 4	ENT			
SHFT	L ANDST	D 3	A 0	→	B 1	G 6	A 0	A 0	ENT
SHFT	F 5	I 8	L ANDST	L ANDST	→	B 1	E 4	A 0	A 0	ENT

Find (FIND)

DS	Used
HPP	Used

The Find instruction is used to search for a specified value in a V-memory table of up to 255 locations. The function parameters are loaded into the first and second levels of the accumulator stack and the accumulator by three additional instructions.



Listed below are the steps necessary to program the Find function.

Step 1: Load the length of the table (number of V-memory locations) into the second level of the accumulator stack. This parameter must be a HEX value, 0–FF.

Step 2: Load the starting V-memory location for the table into the first level of the accumulator stack. This parameter must be a HEX value.

Step 3: Load the offset from the starting location to begin the search. This parameter must be a HEX value.

Step 4: Insert the Find instruction which specifies the first value to be found in the table.

Results:— The offset from the starting address to the first V-memory location which contains the search value (in HEX) is returned to the accumulator. SP53 will be set On if an address outside the table is specified in the offset or the value is not found. If the value is not found 0 will be returned in the accumulator.

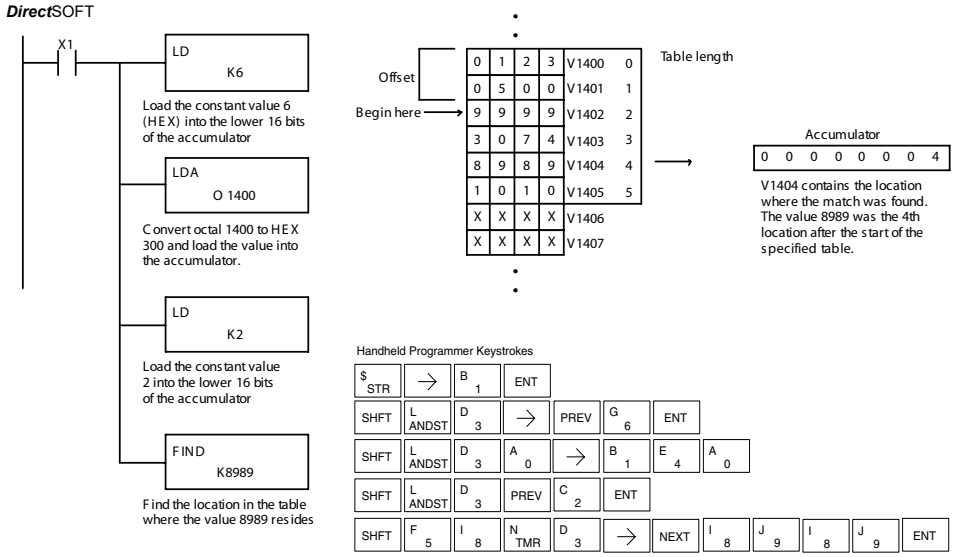
Helpful Hint: — For parameters that require HEX values when referencing memory locations, the LDA instruction can be used to convert an octal address to the HEX equivalent and load the value into the accumulator.

Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Constant	K	0–FF

Discrete Bit Flags	Description
SP53	On if there is no value in the table that is equal to the search value.

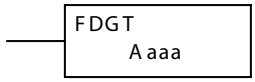
***NOTE**: Status flags are only valid until another instruction that uses the same flags is executed. The pointer for this instruction starts at 0 and resides in the accumulator.*

In the following example, when X1 is on, the constant value (K6) is loaded into the accumulator using the Load instruction. This value specifies the length of the table and is placed in the second stack location when the following Load Address and Load instruction is executed. The octal address 1400 (V1400) is the starting location for the table and is loaded into the accumulator. This value is placed in the first level of the accumulator stack when the following Load instruction is executed. The offset (K2) is loaded into the lower 16 bits of the accumulator using the Load instruction. The value to be found in the table is specified in the Find instruction. If a value is found equal to the search value, the offset (from the starting location of the table) where the value is located will reside in the accumulator.



Find Greater Than (FDGT)

The Find Greater Than instruction is used to search for the first occurrence of a value in a V-memory table that is greater than the specified value (Aaaa), which can be either a V-memory location or a 4-digit constant. The function parameters are loaded into the first level of the accumulator stack and the accumulator by two additional instructions. Listed below are the steps necessary to program the Find Greater Than function.



Step 1: Load the length of the table (up to 255 locations) into the first level of the accumulator stack. This parameter must be a HEX value, 0–FF.

Step 2: Load the starting V-memory location for the table into the accumulator. This parameter must be a HEX value.

Step 3: Insert the FDGT instruction which specifies the greater than search value. Results:— The offset from the starting address to the first V-memory location which contains the greater than search value (in HEX) which is returned to the accumulator. SP53 will be set On if the value is not found and 0 will be returned in the accumulator.

Helpful Hint: — For parameters that require HEX values when referencing memory locations, the LDA instruction can be used to convert an octal address to the HEX equivalent and load the value into the accumulator.

***NOTE**: This instruction does not have an offset, such as the one required for the FIND instruction.*

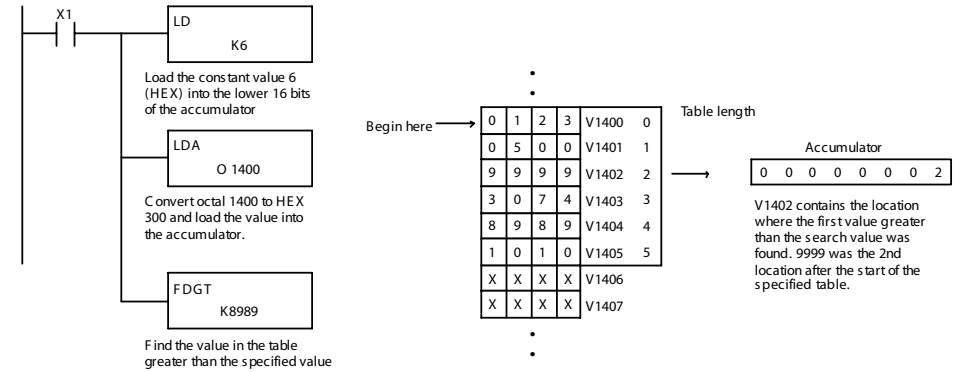
Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Constant	K	0–FF

Discrete Bit Flags	Description
SP53	On if there is no value in the table that is equal to the search value.

***NOTE**: Status flags are only valid until another instruction that uses the same flags is executed. The pointer for this instruction starts at 0 and resides in the accumulator.*

In the following example, when X1 is on, the constant value (K6) is loaded into the accumulator using the Load instruction. This value specifies the length of the table and is placed in the first stack location after the Load Address instruction is executed. The octal address 1400 (V1400) is the starting location for the table and is loaded into the accumulator. The Greater Than search value is specified in the Find Greater Than instruction. If a value is found greater than the search value, the offset (from the starting location of the table) where the value is located will reside in the accumulator. If there is no value in the table that is greater than the search value, a zero is stored in the accumulator and SP53 will come ON.

***Direct*SOFT**



Handheld Programmer Keystrokes

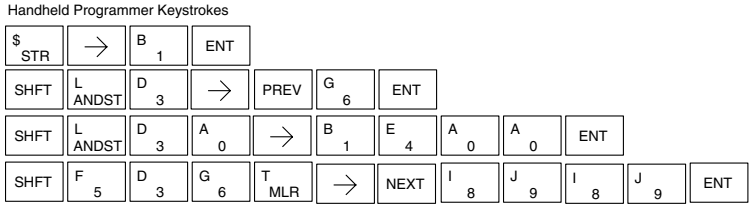
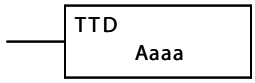


Table to Destination (TTD)

DS	Used
HPP	Used

The Table To Destination instruction moves a value from a V-memory table to a V-memory location and increments the table pointer by 1. The first V-memory location in the table contains the table pointer which indicates the next location in the table to be moved. The instruction will be executed once per scan provided the input remains on. The table pointer will reset to 1 when the value equals the last location in the table. The function parameters are loaded into the first level of the accumulator stack and the accumulator by two additional instructions.



Listed below are the steps necessary to program the Table To Destination function.

Step 1: Load the length of the data table (number of V-memory locations) into the first level of the accumulator stack. This parameter must be a HEX value, 0 to FF.

Step 2: Load the starting V-memory location for the table into the accumulator. (Remember, the starting location of the table is used as the table pointer.) This parameter must be a HEX value.

Step 3: Insert the TTD instruction which specifies destination V-memory location (Vaaa).

Helpful Hint: — For parameters that require HEX values when referencing memory locations, the LDA instruction can be used to convert an octal address to the HEX equivalent and load the value into the accumulator.

Helpful Hint: — The instruction will be executed every scan if the input logic is on. If you do not want the instruction to execute for more than one scan, a one-shot (PD) should be used in the input logic.

Helpful Hint: — The pointer location should be set to the value where the table operation will begin. The special relay SP0 or a one-shot (PD) should be used so the value will only be set in one scan and will not affect the instruction operation.

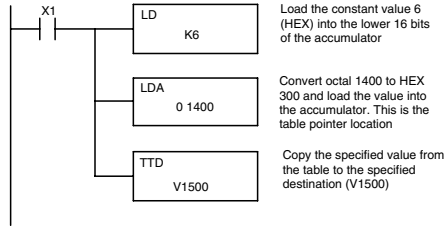
Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map

Discrete Bit Flags	Description
SP56	On when the table pointer equals the table length.

NOTE: *Status flags (SPs) are only valid until another instruction that uses the same flag is executed, or the end of the scan. The pointer for this instruction starts at 0 and resets when the table length is reached. At first glance it may appear that the pointer should reset to 0. However, it resets to 1, not 0.*

In the following example, when X1 is on, the constant value (K6) is loaded into the accumulator using the Load instruction. This value specifies the length of the table and is placed in the first stack location after the Load Address instruction is executed. The octal address 1400 (V1400) is the starting location for the source table and is loaded into the accumulator. Remember, V1400 is used as the pointer location, and is not actually part of the table data source. The destination location (V1500) is specified in the Table to Destination instruction. The table pointer (V1400 in this case) will be increased by "1" after each execution of the TTD instruction.

DirectSOFT

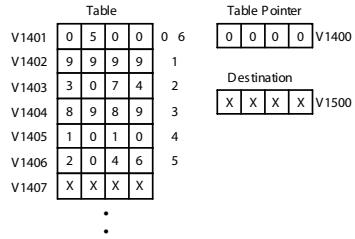


Handheld Programmer Keystrokes

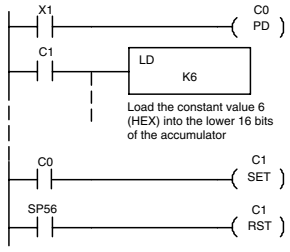
$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	→	PREV	G 6	ENT			
SHFT	L ANDST	D 3	A 0	→	B 1	E 4	A 0	A 0	ENT
SHFT	T MLR	T MLR	D 3	→	B 1	F 5	A 0	A 0	ENT

It is important to understand how the table locations are numbered. If you examine the example table, you'll notice that the first data location, V1401, will be used when the pointer is equal to zero, and again when the pointer is equal to six. Why? Because the pointer is only equal to zero before the very first execution. From then on, it increments from one to six, and then resets to one.

Also, our example uses a normal input contact (X1) to control the execution. Since the CPU scan is extremely fast, and the pointer increments automatically, the table would cycle through the locations very quickly. If this is a problem, you have an option of using SP56 in conjunction with a one-shot (PD) and a latch (C1 for example) to allow the table to cycle through all locations one time and then stop. The logic shown here is not required, it's just an optional method.



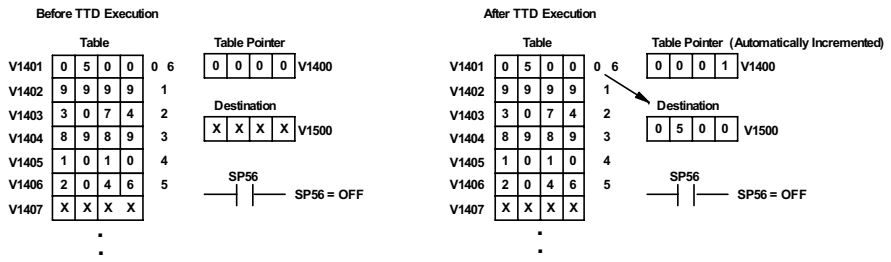
DirectSOFT (optional latch example using SP56)



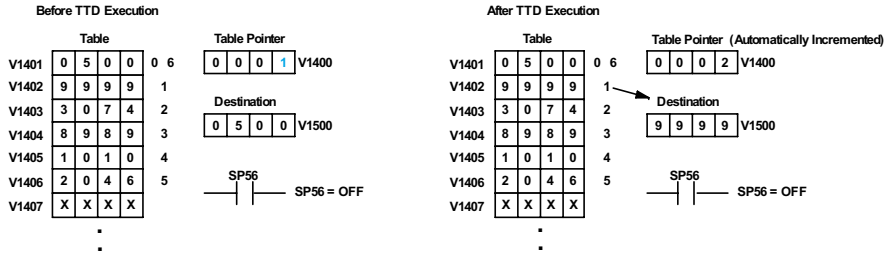
Since Special Relays are reset at the end of the scan, this latch must follow the TTD instruction in the program

The following diagram shows the scan-by-scan results of the execution for our example program. Notice how the pointer automatically cycles from 0 – 6, and then starts over at 1 instead of 0. Also, notice how SP56 is only on until the end of the scan.

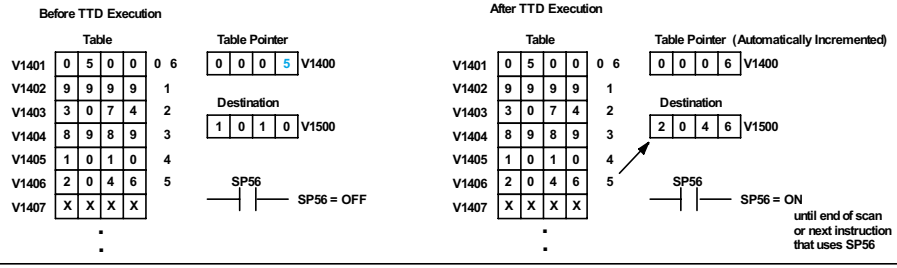
Scan N



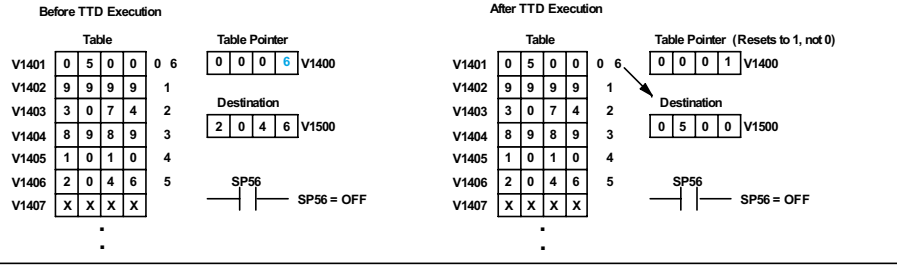
Scan N+1



Scan N+5



Scan N+6



Remove from Bottom (RFB)

DS	Used
HPP	Used

The Remove From Bottom instruction moves a value from the bottom of a V-memory table to a V-memory location and decrements a table pointer by 1. The first V-memory location in the table contains the table pointer which indicates the next location in the table to be moved. The instruction will be executed once per scan provided the input remains on. The instruction will stop operation when the pointer equals 0. The function parameters are loaded into the first level of the accumulator stack and the accumulator by two additional instructions. Listed below are the steps necessary to program the Remove From Bottom function.

Step 1: Load the length of the table (number of V-memory locations) into the first level of the accumulator stack. This parameter must be a HEX value, 0 to FF.

Step 2: Load the starting V-memory location for the table into the accumulator. (Remember, the starting location of the table blank is used as the table pointer.) This parameter must be a HEX value.

Step 3: Insert the RFB instruction which specifies destination V-memory location (Vaaa).

Helpful Hint: — For parameters that require HEX values when referencing memory locations, the LDA instruction can be used to convert an octal address to the HEX equivalent and load the value into the accumulator.

Helpful Hint: — The instruction will be executed every scan if the input logic is on. If you do not want the instruction to execute for more than one scan, a one-shot (PD) should be used in the input logic.

Helpful Hint: — The pointer location should be set to the value where the table operation will begin. The special relay SP0 or a one-shot (PD) should be used so the value will only be set in one scan and will not affect the instruction operation.

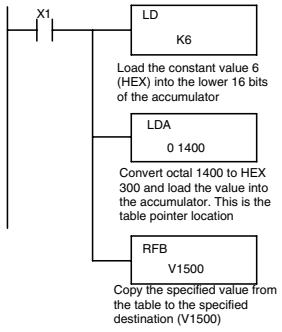
Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map

Discrete Bit Flags	Description
SP56	On when the table pointer equals zero..

***NOTE**: Status flags (SPs) are only valid until another instruction that uses the same flag is executed or the end of the scan The pointer for this instruction can be set to start anywhere in the table. It is not set automatically. You must load a value into the pointer somewhere in your program.*

In the following example, when X1 is on, the constant value (K6) is loaded into the accumulator using the Load instruction. This value specifies the length of the table and is placed in the first stack location after the Load Address instruction is executed. The octal address 1400 (V1400) is the starting location for the source table and is loaded into the accumulator. Remember, V1400 is used as the pointer location, and is not actually part of the table data source. The destination location (V1500) is specified in the Remove From Bottom. The table pointer (V1400 in this case) will be decremented by "1" after each execution of the RFB instruction.

DirectSOFT



Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	→	PREV	G 6	ENT			
SHFT	L ANDST	D 3	A 0	→	B 1	E 4	A 0	A 0	ENT
SHFT	R ORN	F 5	B 1	→	B 1	F 5	A 0	A 0	ENT

It is important to understand how the table locations are numbered. If you examine the example table, you'll notice that the first data location, V1401, will be used when the pointer is equal to one. The second data location, V1402, will be used when the pointer is equal to two, etc.

Table

V1401	0	5	0	0	1
V1402	9	9	9	9	2
V1403	3	0	7	4	3
V1404	8	9	8	9	4
V1405	1	0	1	0	5
V1406	2	0	4	6	6
V1407	X	X	X	X	

Table Pointer

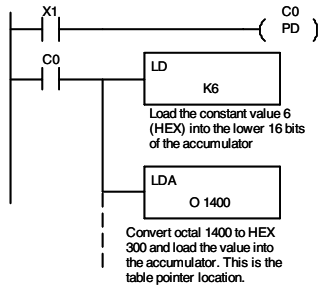
0	0	0	0	V1400

Destination

X	X	X	X	V1500

Also, our example uses a normal input contact (X1) to control the execution. Since the CPU scan is extremely fast, and the pointer decrements automatically, the table would cycle through the locations very quickly. If this is a problem for your application, you have an option of using a one-shot (PD) to remove one value each time the input contact transitions from low to high.

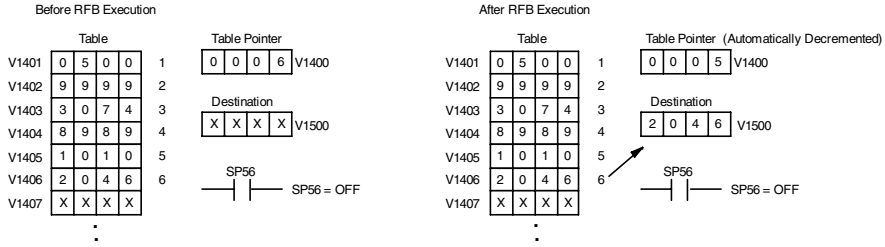
DirectSOFT (optional one-shot method)



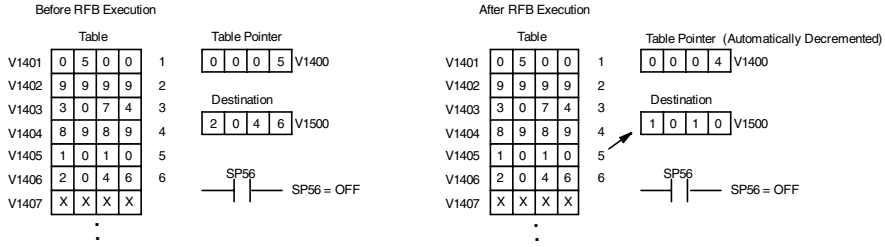
The following diagram shows the scan-by-scan results of the execution for our example program. Notice how the pointer automatically decrements from 6 to 0. Also, notice how SP56 is only on until the end of the scan.

Example of Execution

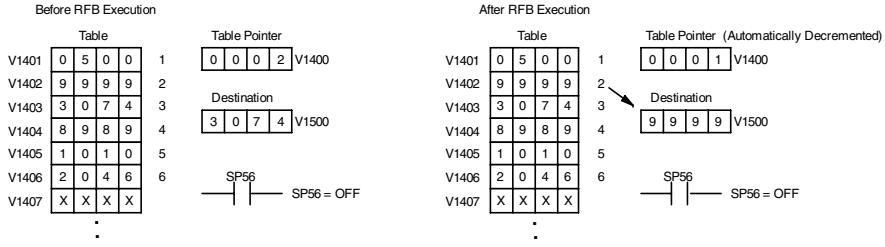
Scan N



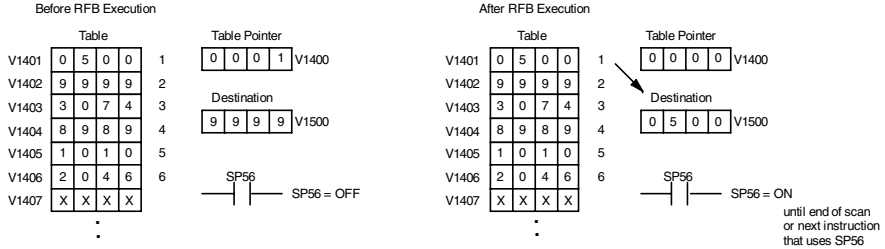
Scan N+1



Scan N+4



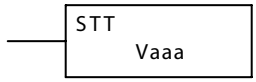
Scan N+5



Source to Table (STT)

DS	Used
HPP	Used

The Source To Table instruction moves a value from a V-memory location into a V-memory table and increments a table pointer by 1. When the table pointer reaches the end of the table, it resets to 1. The first V-memory location in the table contains the table pointer which indicates the next location in the table to store a value. The instruction will be executed once per scan, provided the input remains on. The function parameters are loaded into the first level of the accumulator stack and the accumulator with two additional instructions.



Listed below are the steps necessary to program the Source To Table function.

Step 1: Load the length of the table (number of V-memory locations) into the first level of the accumulator stack. This parameter must be a HEX value, 0 to FF.

Step 2: Load the starting V-memory location for the table into the accumulator. (Remember, the starting location of the table is used as the table pointer.) This parameter must be a HEX value.

Step 3: Insert the STT instruction which specifies the source V-memory location (Vaaa). This is where the value will be moved from.

Helpful Hint: — For parameters that require HEX values when referencing memory locations, the LDA instruction can be used to convert an octal address to the HEX equivalent and load the value into the accumulator.

Helpful Hint:— The instruction will be executed every scan if the input logic is on. If you do not want the instruction to execute for more than one scan, a one-shot (PD) should be used in the input logic.

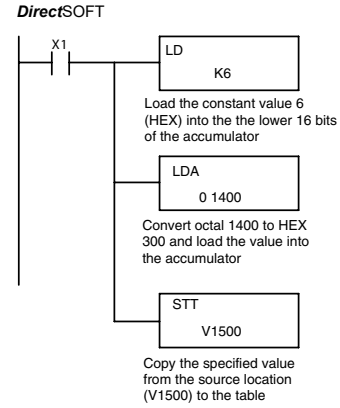
Helpful Hint: — The table counter value should be set to indicate the starting point for the operation. Also, it must be set to a value that is within the length of the table. For example, if the table is 6 words long, then the allowable range of values that could be in the pointer should be between 0 and 6. If the value is outside of this range, the data will not be moved. Also, a one-shot (PD) should be used so the value will only be set in one scan and will not affect the instruction operation.

Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map

Discrete Bit Flags	Description
SP56	On when the table pointer equals the table length.

***NOTE**: Status flags (SPs) are only valid until another instruction that uses the same flag is executed, or the end of the scan. The pointer for this instruction starts at 0 and resets to 1 automatically when the table length is reached.*

In the following example, when X1 is on, the constant value (K6) is loaded into the accumulator using the Load instruction. This value specifies the length of the table and is placed in the first stack location after the Load Address instruction is executed. The octal address 1400 (V1400), which is the starting location for the destination table and table pointer, is loaded into the accumulator. The data source location (V1500) is specified in the Source to Table instruction. The table pointer will be increased by "1" after each time the instruction is executed.

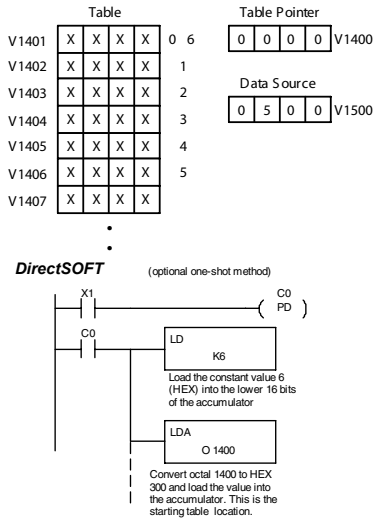


Handheld Programmer Keystrokes

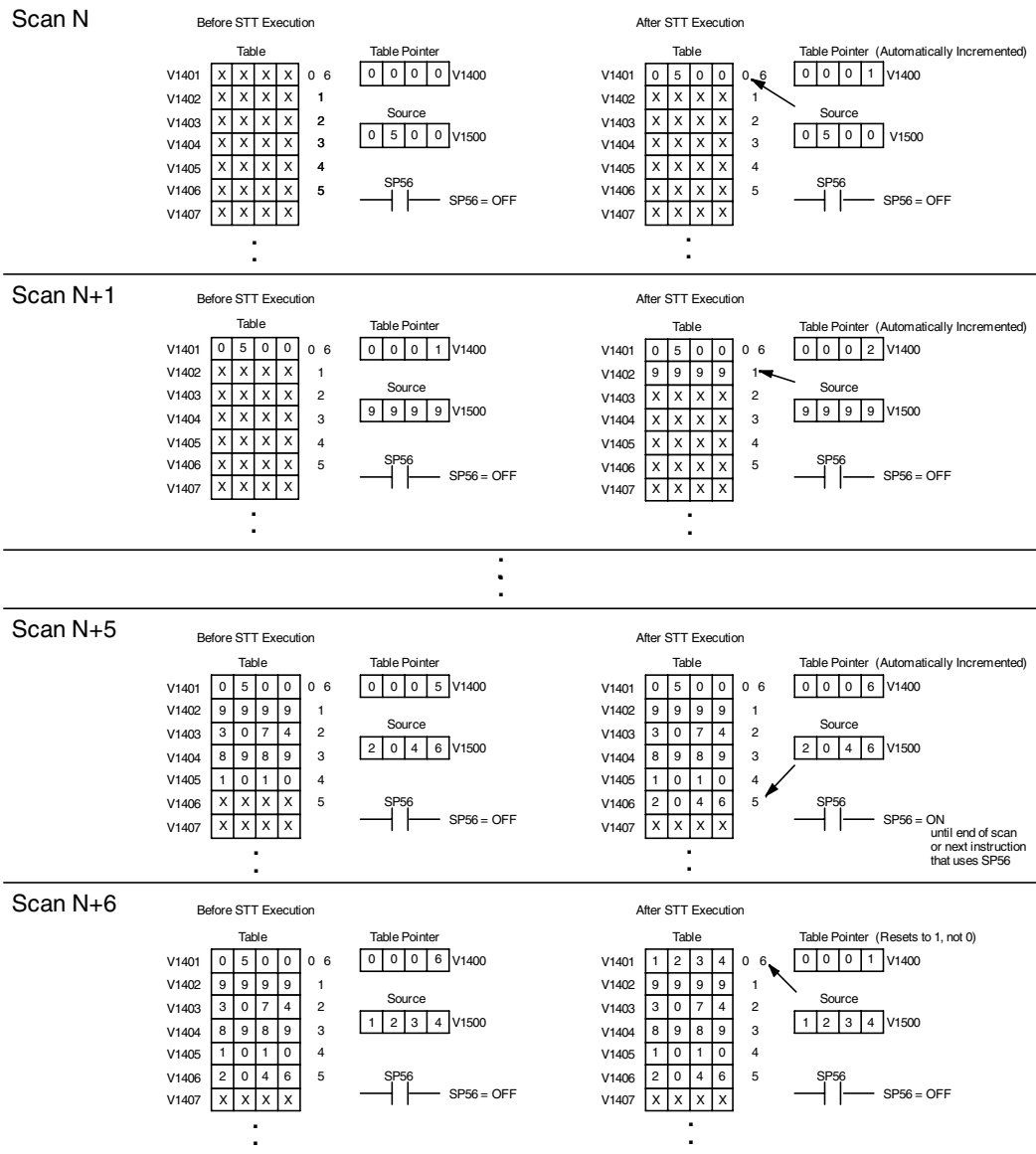
$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	→	PREV	G 6	ENT			
SHFT	L ANDST	D 3	A 0	→	B 1	E 4	A 0	A 0	ENT
SHFT	S RST	SHFT	T MLR	T MLR	→	B 1	F 5	A 0	A 0	ENT

It is important to understand how the table locations are numbered. If you examine the example table, you'll notice that the first data storage location, V1401, will be used when the pointer is equal to zero, and again when the pointer is equal to six. Why? Because the pointer is only equal to zero before the very first execution. From then on, it increments from one to six, and then resets to one.

Also, our example uses a normal input contact (X1) to control the execution. Since the CPU scan is extremely fast, and the pointer increments automatically, the source data would be moved into all the table locations very quickly. If this is a problem for your application, you have an option of using a one-shot (PD) to move one value each time the input contact transitions from low to high.



The following diagram shows the scan-by-scan results of the execution for our example program. Notice how the pointer automatically cycles from 0 to 6, and then starts over at 1 instead of 0. Also, notice how SP56 is affected by the execution. Although our example does not show it, we are assuming that there is another part of the program that changes the value in V1500 (data source) prior to the execution of the STT instruction. This is not required, but it makes it easier to see how the data source is copied into the table.



Remove from Table (RFT)

DS	Used
HPP	Used

The Remove From Table instruction pops a value off a table and stores it in a V-memory location. When a value is removed from the table all other values are shifted up 1 location. The first V-memory location in the table contains the table length counter. The table counter decrements by 1 each time the instruction is executed. If the length counter is zero or greater than the maximum table length (specified in the first level of the accumulator stack) the instruction will not execute and SP56 will be On.

The instruction will be executed once per scan, provided the input remains on. The function parameters are loaded into the first level of the accumulator stack and the accumulator by two additional instructions. Listed below are the steps necessary to program the Remove From Table function.

Step 1: Load the length of the table (number of V-memory locations) into the first level of the accumulator stack. This parameter must be a HEX value, 0 to FF.

Step 2: Load the starting V-memory location for the table into the accumulator. (Remember, the starting location of the table is used as the table length counter.) This parameter must be a HEX value.

Step 3: Insert the RFT instructions which specifies destination V-memory location (Vaaa). This is where the value will be moved to.

Helpful Hint: — For parameters that require HEX values when referencing memory locations, the LDA instruction can be used to convert an octal address to the HEX equivalent and load the value into the accumulator.

Helpful Hint:— The instruction will be executed every scan if the input logic is on. If you do not want the instruction to execute for more than one scan, a one-shot (PD) should be used in the input logic.

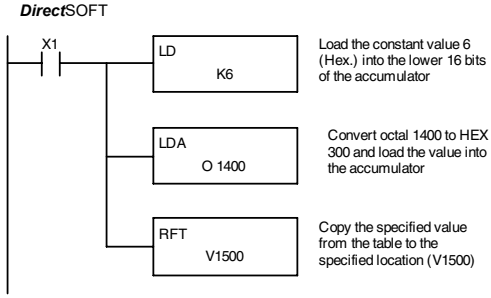
Helpful Hint: — The table counter value should be set to indicate the starting point for the operation. Also, it must be set to a value that is within the length of the table. For example, if the table is 6 words long, then the allowable range of values that could be in the table counter should be between 1 and 6. If the value is outside of this range or zero, the data will not be moved from the table. Also, a one-shot (PD) should be used so the value will only be set in one scan and will not affect the instruction operation.

Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map

Discrete Bit Flags	Description
SP56	On when the table pointer equals zero.

***NOTE**: Status flags (SPs) are only valid until another instruction that uses the same flag is executed, or the end of the scan The pointer for this instruction can be set to start anywhere in the table. It is not set automatically. You must load a value into the pointer somewhere in your program.*

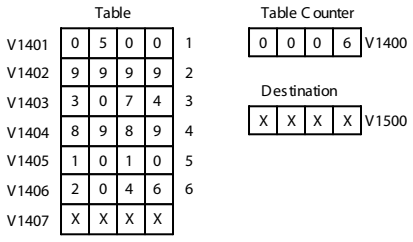
In the following example, when X1 is on, the constant value (K6) is loaded into the accumulator using the Load instruction. This value specifies the length of the table and is placed in the first stack location after the Load Address instruction is executed. The octal address 1400 (V1400) is the starting location for the source table and is loaded into the accumulator. The destination location (V1500) is specified in the Remove from Table instruction. The table counter will be decreased by "1" after the instruction is executed.



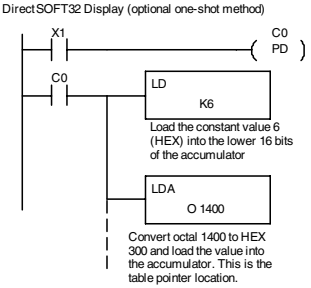
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT					
SHFT	L ANDST	D 3	→	PREV	G 6	ENT		
SHFT	L ANDST	D 3	A 0	→	B 1	E 4	A 0	A 0	ENT
SHFT	R ORN	F 5	T MLR	→	B 1	F 5	A 0	A 0	ENT

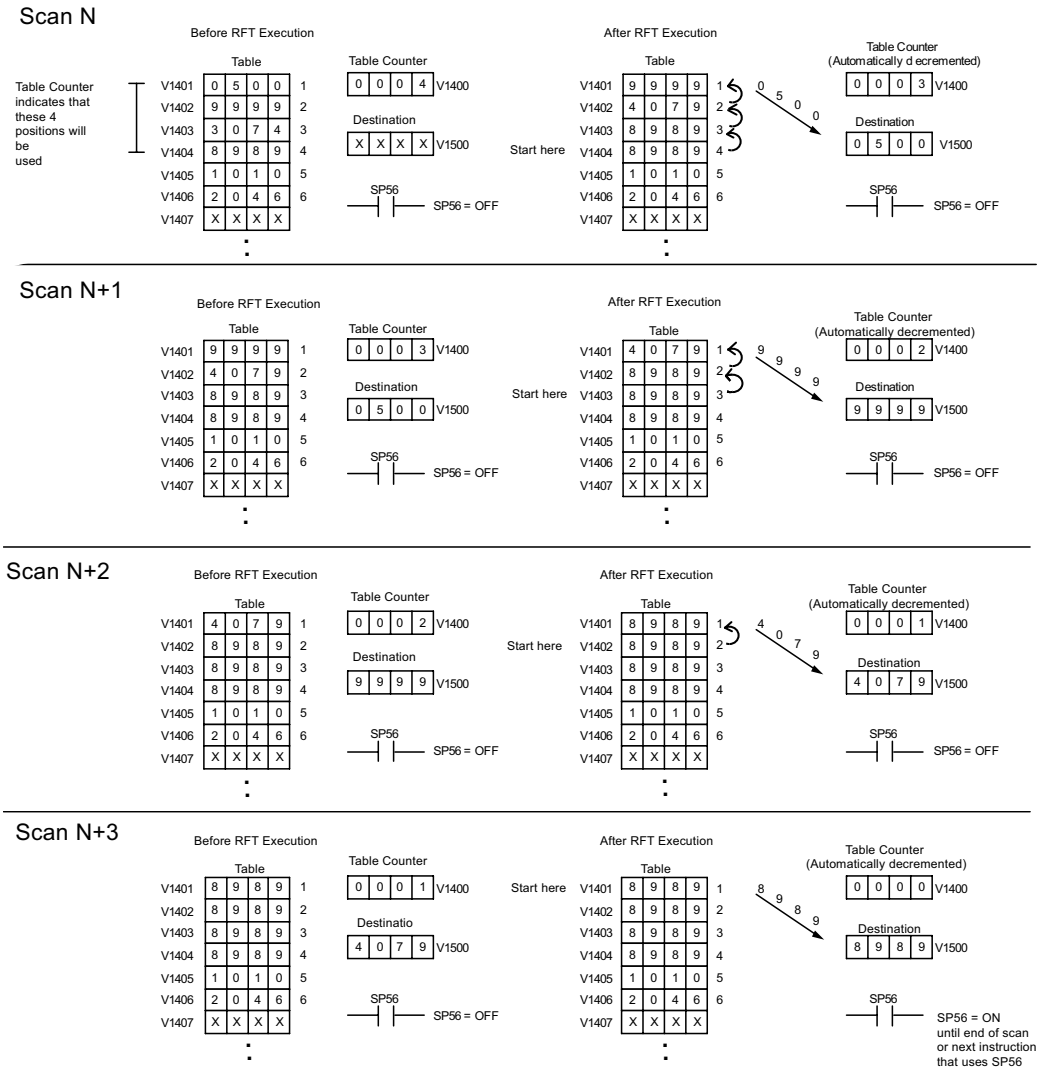
Since the table counter specifies the range of data that will be removed from the table, it is important to understand how the table locations are numbered. If you examine the example table, you'll notice that the data locations are numbered from the top of the table. For example, if the table counter started at 6, then all six of the locations would be affected during the instruction execution.



Also, our example uses a normal input contact (X1) to control the execution. Since the CPU scan is extremely fast, and the pointer decrements automatically, the data would be removed from the table very quickly. If this is a problem for your application, you have an option of using a one-shot (PD) to remove one value each time the input contact transitions from low to high.



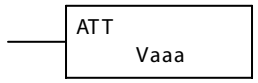
The following diagram shows the scan-by-scan results of the execution for our example program. In our example, we show the table counter set to 4, initially. (Remember, you can set the table counter to any value that is within the range of the table.) The table counter automatically decrements from 4 to 0 as the instruction is executed. Notice how the last two table positions, 5 and 6, are not moved up through the table. Also, notice that SP56, which comes on when the table counter is zero, is only on until the end of the scan.



Add to Top (ATT)

DS	Used
HPP	Used

The Add To Top instruction pushes a value on to a V-memory table from a V-memory location. When the value is added to the table all other values are pushed down 1 location.



The instruction will be executed once per scan, provided the input remains on. The function parameters are loaded into the first level of the accumulator stack and the accumulator by two additional instructions. Listed below are the steps necessary to program the Add To Top function.

Step 1: Load the length of the table (number of V-memory locations) into the first level of the accumulator stack. This parameter must be a HEX value, 0 to FF.

Step 2: Load the starting V-memory location for the table into the accumulator. (Remember, the starting location of the table is used as the table length counter.) This parameter must be a HEX value.

Step 3: Insert the ATT instructions which specifies source V-memory location (Vaaa). This is where the value will be moved from.

Helpful Hint:— The instruction will be executed every scan if the input logic is on. If you do not want the instruction to execute for more than one scan, a one-shot (PD) should be used in the input logic.

Helpful Hint: — For parameters that require HEX values when referencing memory locations, the LDA instruction can be used to convert an octal address to the HEX equivalent and load the value into the accumulator.

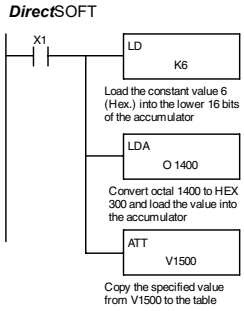
Helpful Hint: — The table counter value should be set to indicate the starting point for the operation. Also, it must be set to a value that is within the length of the table. For example, if the table is 6 words long, then the allowable range of values that could be in the table counter should be between 1 and 6. If the value is outside of this range or zero, the data will not be moved into the table. Also, a one-shot (PD) should be used so the value will only be set in one scan and will not affect the instruction operation.

Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map

Discrete Bit Flags	Description
SP56	On when the table pointer equal to the table size.

***NOTE**: Status flags (SPs) are only valid until another instruction that uses the same flag is executed or the end of the scan. The pointer for this instruction can be set to start anywhere in the table. It is not set automatically. You must load a value into the pointer somewhere in your program.*

In the following example, when X1 is on, the constant value (K6) is loaded into the accumulator using the Load instruction. This value specifies the length of the table and is placed in the first stack location after the Load Address instruction is executed. The octal address 1400 (V1400), which is the starting location for the destination table and table counter, is loaded into the accumulator. The source location (V1500) is specified in the Add to Top instruction. The table counter will be increased by "1" after the instruction is executed.



Handheld Programmer Keystrokes

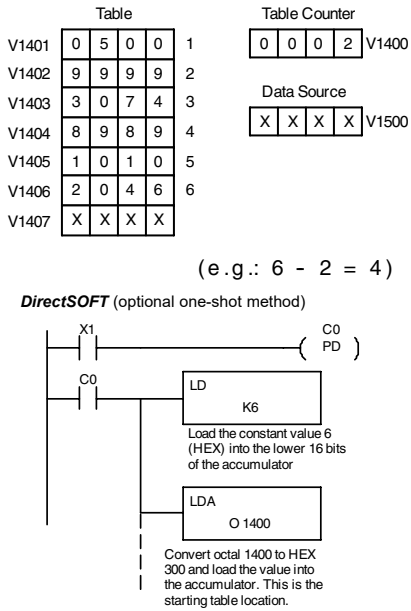
$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	→	PREV	G 6	ENT			
SHFT	L ANDST	D 3	A 0	→	B 1	E 4	A 0	A 0	ENT
SHFT	A 0	T MLR	T MLR	→	B 1	F 5	A 0	A 0	ENT

For the ATT instruction, the table counter determines the number of additions that can be made before the instruction will stop executing. So, it is helpful to understand how the system uses this counter to control the execution.

For example, if the table counter was set to 2, and the table length was 6 words, then there could only be 4 additions of data before the execution was stopped. This can easily be calculated by:

Table length – table counter = number of executions

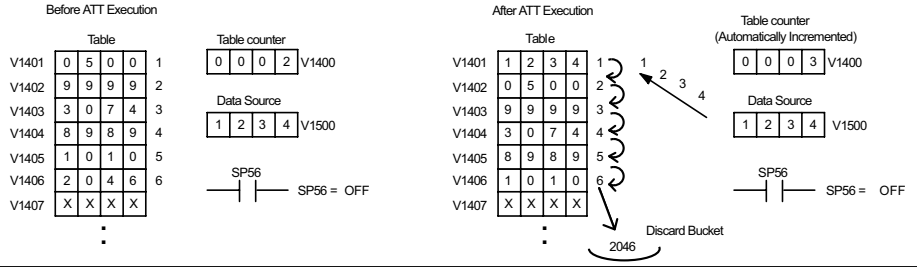
Also, our example uses a normal input contact (X1) to control the execution. Since the CPU scan is extremely fast, and the table counter increments automatically, the data would be moved into the table very quickly. If this is a problem for your application, you have an option of using a one-shot (PD) to add one value each time the input contact transitions from low to high.



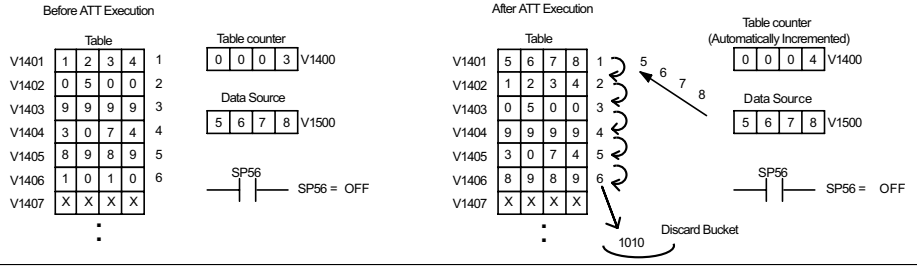
The following diagram shows the scan-by-scan results of the execution for our example program. The table counter is set to 2 initially, and it will automatically increment from 2 to 6 as the instruction is executed. Notice how SP56 comes on when the table counter is 6, which is equal to the table length. Plus, although our example does not show it, we are assuming that there is another part of the program that changes the value in V1500 (data source) prior to the execution of the ATT instruction.

Example of Execution

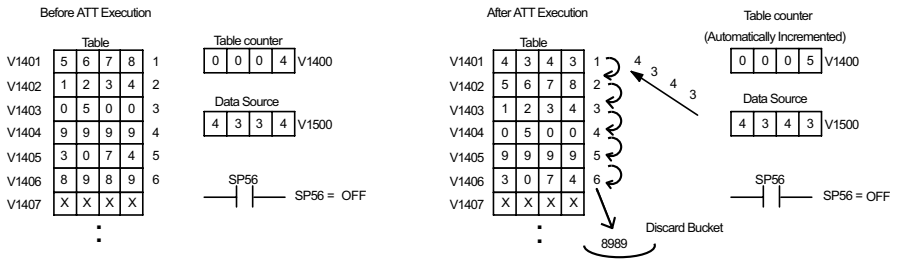
Scan N



Scan N+1



Scan N+2



Scan N+3

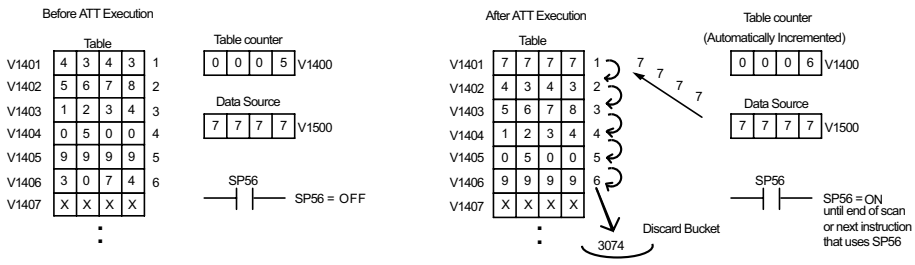


Table Shift Left (TSHFL)

DS	Used
HPP	Used

The Table Shift Left instruction shifts all the bits in a V-memory table to the left, the specified number of bit positions.

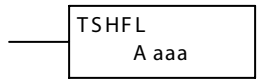
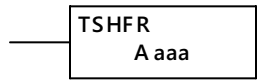


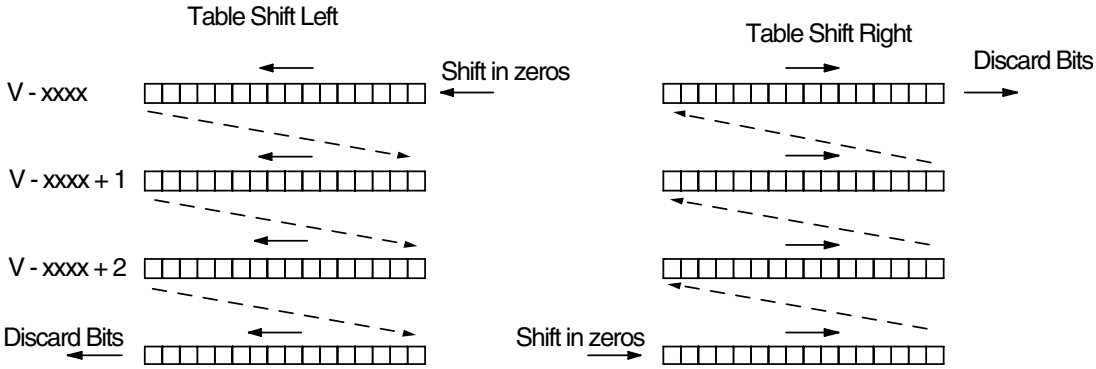
Table Shift Right (TSHFR)

DS	Used
HPP	Used

The Table Shift Right instruction shifts all the bits in a V-memory table to the right, a specified number of bit positions.



The following description applies to both the Table Shift Left and Table Shift Right instructions. A table is just a range of V-memory locations. The Table Shift Left and Table Shift Right instructions shift bits serially throughout the entire table. Bits are shifted out the end of one word and into the opposite end of an adjacent word. At the ends of the table, bits are either discarded, or zeros are shifted into the table. The example tables below are arbitrarily four words long.



Step 1: Load the length of the table (number of V-memory locations) into the first level of the accumulator stack. This parameter must be a HEX value, 0 to FF.

Step 2: Load the starting V-memory location for the table into the accumulator. This parameter must be a HEX value. You can use the LDA instruction to convert an octal address to hex.

Step 3: Insert the Table Shift Left or Table shift Right instruction. This specifies the number of bit positions you wish to shift the entire table. The number of bit positions must be in octal.

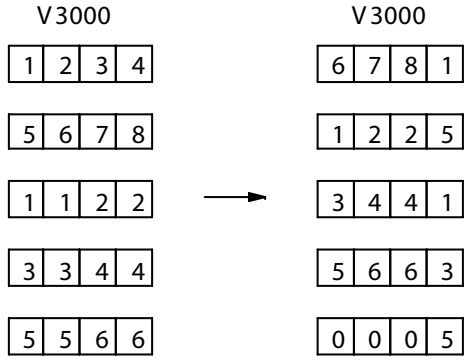
Helpful hint: — Remember that each V-memory location contains 16 bits. So, the bits of the first word of the table are numbered from 0 to 17 octal. If you want to shift the entire table by 20 bits, that is 24 octal. SP 53 will be set if the number of bits to be shifted is larger than the total bits contained within the table. Flag 67 will be set if the last bit shifted (just before it is discarded) is a "1".

Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map

Discrete Bit Flags	Description
SP53	On when the number of bits to be shifted is larger than the total bits contained within the table
SP67	On when the last bit shifted (just before it is discarded) is a 1

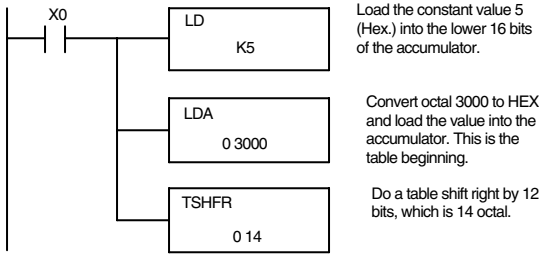
NOTE: *Status flags are only valid until the end of the scan or another instruction that uses the same flag is executed.*

The example table to the right contains BCD data as shown (for demonstration purposes). Suppose we want to do a table shift right by 3 BCD digits (12 bits). Converting to octal, 12 bits is 14 octal. Using the Table Shift Right instruction and specifying a shift by octal 14, we have the resulting table shown at the far right. Notice that the 2–3–4 sequence has been discarded, and the 0–0–0 sequence has been shifted in at the bottom.



The following ladder example assumes the data at V3000 to V3004 already exists as shown above. We will use input X0 to trigger the Table Shift Right operation. First, we will load the table length (5 words) into the accumulator stack. Next, we load the starting address into the accumulator. Since V3000 is an octal number, we have to convert it to hex by using the LDA command. Finally, we use the Table Shift Right instruction and specify the number of bits to be shifted (12 decimal), which is 14 octal.

DirectSOFT



Handheld Programmer Keystrokes

$ STR	→	A 0	ENT						
SHFT	L ANDST	D 3	→	PREV	F 5	ENT			
SHFT	L ANDST	D 3	A 0	→	D 3	A 0	A 0	A 0	ENT
SHFT	T MLR	SHFT	S RST	H 7	F 5	R ORN	→	NEXT	B 1	E 4	ENT

AND Move (ANDMOV)

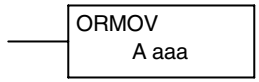
DS	Used
HPP	Used

The AND Move instruction copies data from a table to the specified memory location, ANDing each word with the accumulator data as it is written.

OR Move (ORMOV)

DS	Used
HPP	Used

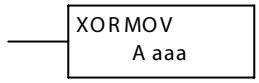
The OR Move instruction copies data from a table to the specified memory location, ORing each word with the accumulator contents as it is written



Exclusive OR Move (XORMOV)

DS	Used
HPP	Used

The Exclusive OR Move instruction copies data from a table to the specified memory location, XORing each word with the accumulator value as it is written.



The following description applies to the AND Move, OR Move, and Exclusive OR Move instructions. A table is just a range of V-memory locations. These instructions copy the data of a table to another specified location, preforming a logical operation on each word with the accumulator contents as the new table is written.

Step 1: Load the length of the table (number of V-memory locations) into the first level of the accumulator stack. This parameter must be a HEX value, 0 to FF.

Step 2: Load the starting V-memory location for the table into the accumulator. This parameter must be a HEX value. You can use the LDA instruction to convert an octal address to hex.

Step 3: Load the BCD/hex bit pattern into the accumulator which will be logically combined with the table contents as they are copied.

Step 4: Insert the AND Move, OR Move, or XOR Move instruction. This specifies the starting location of the copy of the original table. This new table will automatically be the same length as the original table.

Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map

The example table to the right contains BCD data as shown (for demonstration purposes). Suppose we want to move a table of two words at V3000 and AND it with K6666. The copy of the table at V3100 shows the result of the AND operation for each word.

The program on the next page performs the ANDMOV operation example above. It assumes that the data in the table at V3000 – V3001 already exists. First we load the table length (two words) into the accumulator. Next we load the starting address of the source table, using the LDA instruction. Then we load the data into the accumulator to be ANDed with the table. In the ANDMOV command, we specify the table destination, V3100.

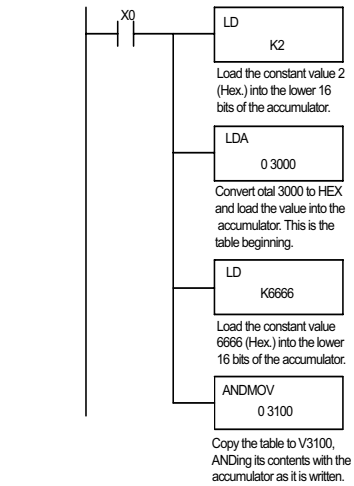
Handheld Programmer Keystrokes

$ STR	→	A 0	ENT						
SHFT	L ANDST	D 3	→	PREV	C 2	ENT			
SHFT	L ANDST	D 3	A 0	→	D 3	A 0	A 0	A 0	ENT
SHFT	L ANDST	D 3	→	PREV	G 6	G 6	G 6	G 6	ENT
V AND	SHFT	M ORST	O INST#	V AND	→	D 3	B 1	A 0	A 0	ENT

The example on top the right shows a table of two words at V3000 and logically ORs it with K8888. The copy of the table at V3100 shows the result of the OR operation for each word.

The program to the right performs the ORMOV example above. It assumes that the data in the table at V3000 – V3001 already exists. First we load the table length (two words) into the accumulator. Next we load the starting address of the source table, using the LDA instruction. Then we load the data into the accumulator to be OR'd with the table. In the ORMOV command, we specify the table destination, V3100.

DirectSOFT



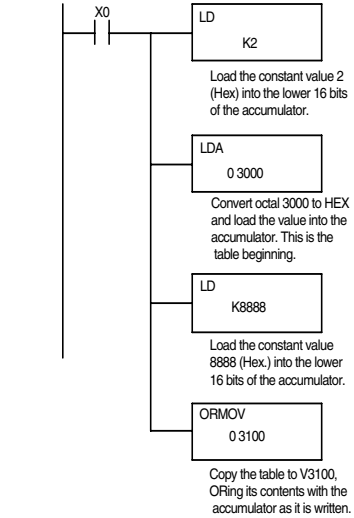
Handheld Programmer Keystrokes

$ STR	→	A 0	ENT						
SHFT	L ANDST	D 3	→	PREV	C 2	ENT			
SHFT	L ANDST	D 3	A 0	→	D 3	A 0	A 0	A 0	ENT
SHFT	L ANDST	D 3	→	PREV	I 8	I 8	I 8	I 8	ENT
Q OR	SHFT	M ORST	O INST#	V AND	→	D 3	B 1	A 0	A 0	ENT

The example to the right shows a table of two words at V3000 and logically XORs it with K3333. The copy of the table at V3100 shows the result of the XOR operation for each word.

The ladder program example for the XORMOV is similar to the one above for the ORMOV. Just use the XORMOV instruction. On the handheld programmer, you must use the SHFT key and spell "XORMOV" explicitly.

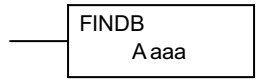
DirectSOFT



Find Block (FINDB)

DS	Used
HPP	N/A

The Find Block instruction searches for an occurrence of a specified block of values in a V-memory table. The function parameters are loaded into the first and second levels of the accumulator stack and the accumulator by three additional instructions. If the block is found, its starting address will be stored in the accumulator. If the block is not found, flag SP53 will be set.



Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
V-memory	P	See memory map

Discrete Bit Flags	Description
SP56	On when the specified block is not found.

The steps listed below are the steps necessary to program the Find Block function.

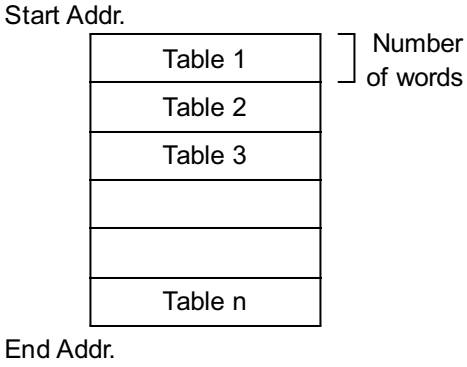
Step 1: Load the number of bytes in the block to be located. This parameter must be a HEX value, 0 to FF.

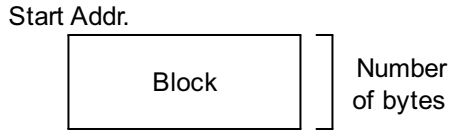
Step 2: Load the length of a table (number of words) to be searched. The Find Block will search multiple tables that are adjacent in V-memory. This parameter must be a HEX value, 0 to FF.

Step 3: Load the ending location for all the tables into the accumulator. This parameter must be a HEX value. You can use the LDA instruction to convert an octal address to hex.

Step 4: Load the table starting location for all the tables into the accumulator. This parameter must be a HEX value. You can use the LDA instruction to convert an octal address to hex.

Step 5: Insert the Find Block instruction. This specifies the starting location of the block of data you are trying to locate.

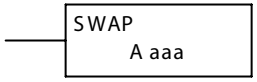




Swap (SWAP)

DS	Used
HPP	Used

The Swap instruction exchanges the data in two tables of eq length.



Step 1: Load the length of the tables (number of V-memory locations) into the first level of the accumulator stack. This parameter must be a HEX value, 0 to FF. Remember that the tables must be of equal length.

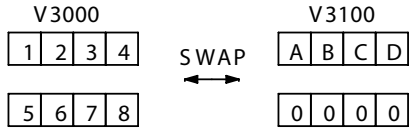
Step 2: Load the starting V-memory location for the first table into the accumulator. This parameter must be a HEX value. You can use the LDA instruction to convert an octal address to hex.

Step 3: Insert the Swap instruction. This specifies the starting address of the second table. This parameter must be a HEX value. You can use the LDA instruction to convert an octal address to hex.

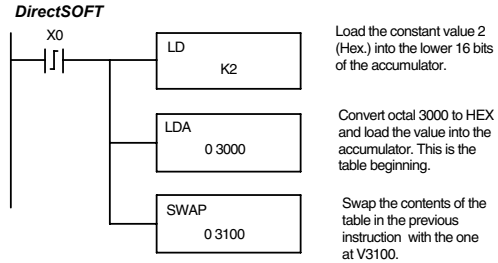
Helpful hint: — The data swap occurs within a single scan. If the instruction executes on multiple consecutive scans, it will be difficult to know the actual contents of either table at any particular time. So, remember to swap just on a single scan.

Operand Data Type		DL06 Range
		aaa
V-memory	V	See memory map

The example to the right shows a table of two words at V3000. We will swap its contents with another table of two words at 3100 by using the Swap instruction. The required ladder program is given below.



The example program below uses a PD contact (triggers for one scan for off-to-on transition). First, we load the length of the tables (two words) into the accumulator. Then we load the address of the first table (V3000) into the accumulator using the LDA instruction, converting the octal address to hex. Note that it does not matter which table we declare "first", because the swap results will be the same.



Handheld Programmer Keystrokes

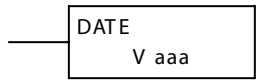
$ STR	SHFT	P CV	D 3	→	A 0	ENT			
SHFT	L ANDST	D 3	→	PREV	C 2	ENT			
SHFT	L ANDST	D 3	A 0	→	D 3	A 0	A 0	A 0	ENT
SHFT	S RST	SHFT	W ANDN	A 0	P CV	→	D 3	B 1	A 0	A 0	ENT

Clock/Calendar Instructions

Date (DATE)

DS	Used
HPP	Used

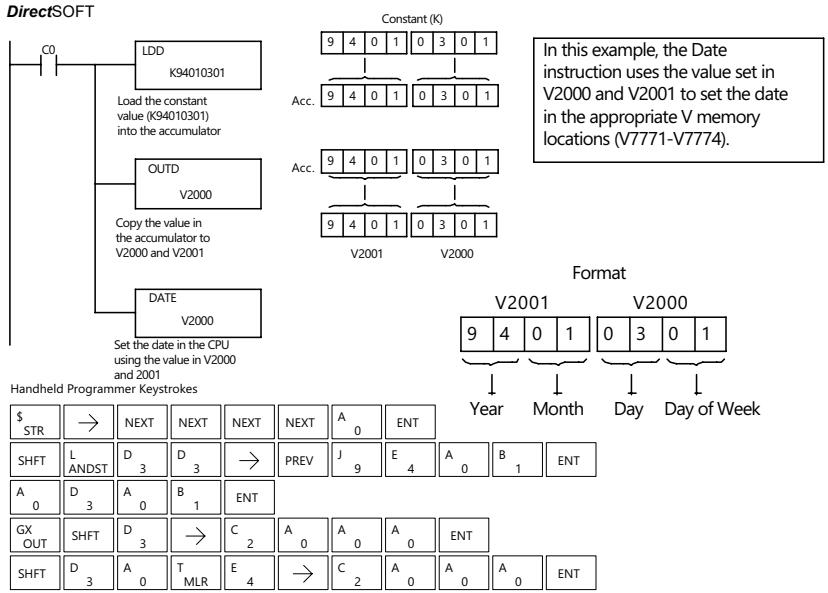
The Date instruction can be used to set the date in the CPU. The instruction requires two consecutive V-memory locations (Vaaa) to set the date. If the values in the specified locations are not valid, the date will not be set. The current date can be read from 4 consecutive V-memory locations (V7771–V7774).



In the following example, when C0 is on, the constant value (K94010301) is loaded into the accumulator using the Load Double instruction (C0 should be a contact from a one-shot (PD) instruction). The value in the accumulator is output to V2000 using the Out Double instruction. The Date instruction uses the value in V2000 to set the date in the CPU.

Date	Range	V-memory Location (BCD) (READ Only)
Year	0-99	V7774
Month	1-12	V7773
Day	1-31	V7772
Day of Week	0-06	V7771
The values entered for the day of week are: 0=Sunday, 1=Monday, 2=Tuesday, 3=Wednesday, 4=Thursday, 5=Friday, 6=Saturday		

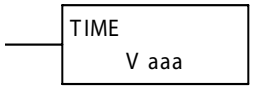
Operand Data Type		DL06 Range
		aaa
V-memory	V	See memory map



Time (TIME)

DS	Used
HPP	Used

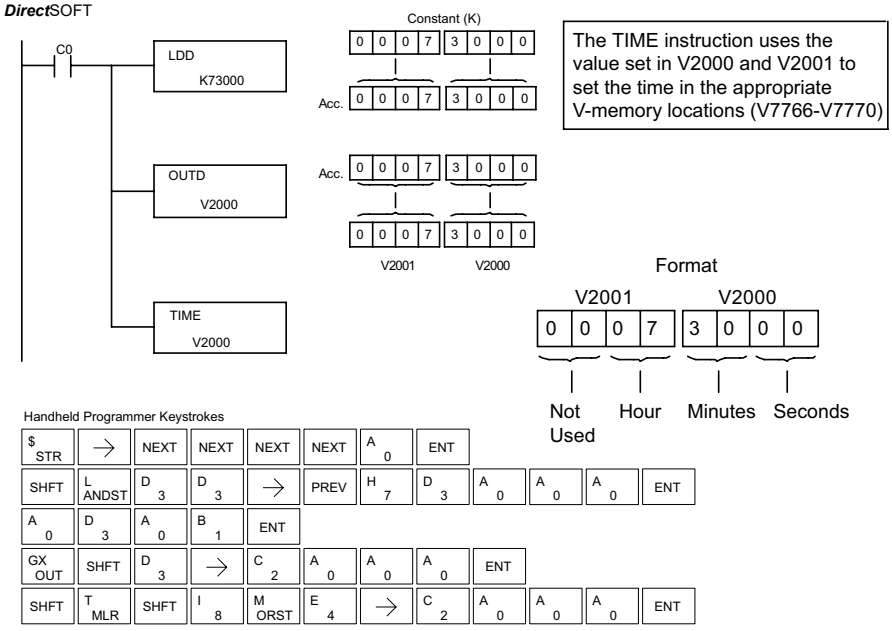
The Time instruction can be used to set the time (24 hour clock) in the CPU. The instruction requires two consecutive V-memory locations (Vaaa) which are used to set the time. If the values in the specified locations are not valid, the time will not be set. The current time can be read from memory locations V7747 and V7766–V7770.



Date	Range	VMemory Location (BCD) (READ Only)
1/100 seconds (10ms)	0-99	V7747
Seconds	0-59	V7766
Minutes	0-59	V7767
Hour	0-23	V7770

Operand Data Type		DL06 Range
		aaa
V-memory	V	See memory map

In the following example, when C0 is on, the constant value (K73000) is loaded into the accumulator using the Load Double instruction (C0 should be a contact from a one-shot (PD) instruction). The value in the accumulator is output to V2000 using the Out Double instruction. The Time instruction uses the value in V2000 to set the time in the CPU.

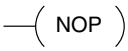


CPU Control Instructions

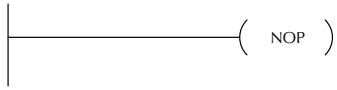
No Operation (NOP)

DS	Used
HPP	Used

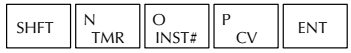
The No Operation is an empty (not programmed) memory location.



*Direct*SOFT



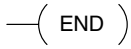
Handheld Programmer Keystrokes



End (END)

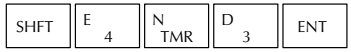
DS	Used
HPP	Used

The End instruction marks the termination point of the normal program scan. An End instruction is required at the end of the main program body. If the End instruction is omitted, an error will occur and the CPU will not enter the Run Mode. Data labels, subroutines and interrupt routines are placed after the End instruction. The End instruction is not conditional; therefore, no input contact is allowed.



*Direct*SOFT

Handheld Programmer Keystrokes



Stop (STOP)

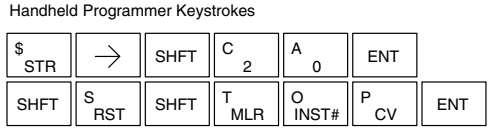
DS	Used
HPP	Used

The Stop instruction changes the operational mode of the CPU from Run to Program (Stop) mode. This instruction is typically used to stop PLC operation in an error condition.

In the following example, when C0 turns on, the CPU will stop operation and switch to the program mode.

*Direct*SOFT

Handheld Programmer Keystrokes



Discrete Bit Flags	Description
SP16	On when the DL06 goes into the TERM_PRG mode.
SP53	On when the DL06 goes into the PRG mode.

Reset Watch Dog Timer (RSTWT)

DS	Used
HPP	Used

The Reset Watch Dog Timer instruction resets the CPU scan timer. The default setting for the watch dog timer is 200ms. Scan times very seldom exceed 200ms, but it is possible. For/next loops, subroutines, interrupt routines, and table instructions can be programmed such that the scan becomes longer than 200ms. When instructions are used in a manner that could exceed the watch dog timer setting, this instruction can be used to reset the timer.

A software timeout error (E003) will occur and the CPU will enter the program mode if the scan time exceeds the watch dog timer setting. Placement of the RSTWT instruction in the program is very important. The instruction has to be executed before the scan time exceeds the watch dog timer's setting.

If the scan time is consistently longer than the watch dog timer's setting, the timeout value may be permanently increased from the default value of 200ms by AUX 55 on the HPP or the appropriate auxiliary function in your programming package. This eliminates the need for the RSTWT instruction.

In the following example, the CPU scan timer will be reset to 0 when the RSTWT instruction is executed. See the For/Next instruction for a detailed example.

DirectSOFT

Handheld Programmer Keystrokes

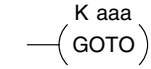
SHFT	R ORN	S RST	T MLR	W ANDN	T MLR	ENT

Program Control Instructions

Goto Label (GOTO) (LBL)

DS	Used
HPP	Used

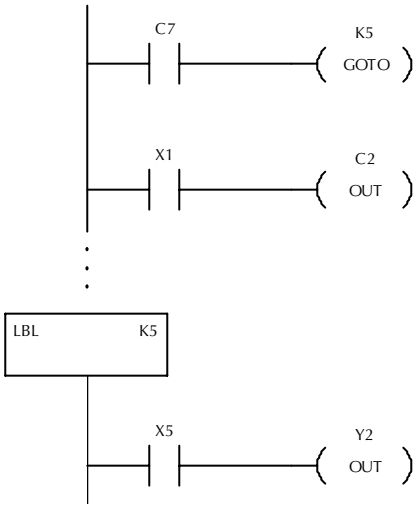
The Goto / Label skips all instructions between the Goto and the corresponding LBL instruction. The operand value for the Goto and the corresponding LBL instruction are the same. The logic between Goto and LBL instruction is not executed when the Goto instruction is enabled. Up to 256 Goto instructions and 256 LBL instructions can be used in the program.



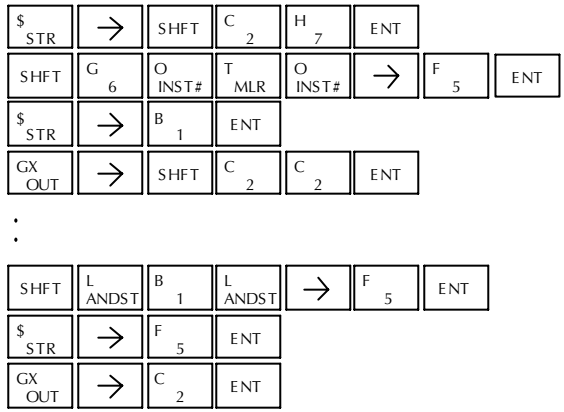
Operand Data Type		DL06 Range
		aaa
Constant	K	1-FFFF

In the following example, when C7 is on, all the program logic between the GOTO and the corresponding LBL instruction (designated with the same constant Kaaa value) will be skipped. The instructions being skipped will not be executed by the CPU.

***Direct*SOFT**



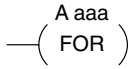
Handheld Programmer Keystrokes



For / Next (FOR) (NEXT)

DS	Used
HPP	Used

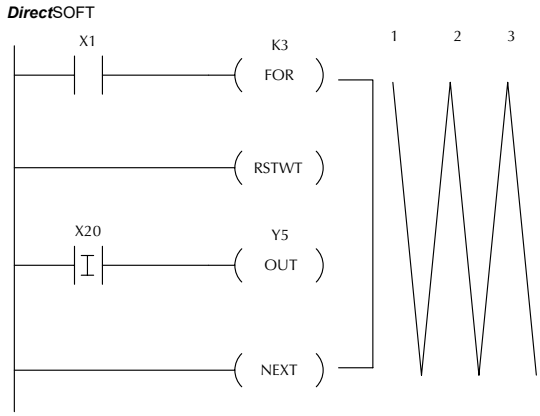
The For and Next instructions are used to execute a section of ladder logic between the For and Next instruction a specified numbers of times. When the For instruction is enabled, the program will loop the specified number of times. If the For instruction is not energized, the section of ladder logic between the For and Next instructions is not executed.



For / Next instructions cannot be nested. The normal I/O update and CPU housekeeping are suspended while executing the For / Next loop. The program scan can increase significantly, depending on the amount of times the logic between the For and Next instruction is executed. With the exception of immediate I/O instructions, I/O will not be updated until the program execution is completed for that scan. Depending on the length of time required to complete the program execution, it may be necessary to reset the watch dog timer inside of the For / Next loop using the RSTWT instruction.

Operand Data Type		DL06 Range
		aaa
V-memory	V	See memory map
Constant	K	1-9999

In the following example, when X1 is on, the application program inside the For / Next loop will be executed three times. If X1 is off, the program inside the loop will not be executed. The immediate instructions may or may not be necessary, depending on your application. Also, The RSTWT instruction is not necessary if the For / Next loop does not extend the scan time beyond the Watch Dog Timer setting. For more information on the Watch Dog Timer, refer to the RSTWT instruction.



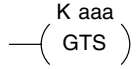
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT			
SHFT	F 5	O INST#	R ORN	→	D 3	ENT
SHFT	R ORN	S RST	T MLR	W ANDN	T MLR	ENT
$ STR	SHFT	I 8	→	C 2	A 0	ENT
GX OUT	→	F 5	ENT			
SHFT	N TMR	E 4	X SET	T MLR	ENT	

Goto Subroutine (GTS) (SBR)

DS	Used
HPP	Used

The Goto Subroutine instruction allows a section of ladder logic to be placed outside the main body of the program, to execute only when needed. There can be a maximum of 256 GTS instructions and 256 SBR instructions used in a program. The GTS instructions can be nested up to 8 levels. An error E412 will occur if the maximum limits are exceeded. Typically this will be used in an application where a block of program logic may be slow to execute and is not required to execute every scan. The subroutine label and all associated logic is placed after the End statement in the program. When the subroutine is called from the main program, the CPU will execute the subroutine (SBR) with the same constant number (K) as the GTS instruction which called the subroutine.



By placing code in a subroutine it is only scanned and executed when needed, since it resides after the End instruction. Code which is not scanned does not impact the overall scan time of the program.

Operand Data Type		DL06 Range
		aaa
Constant	K	1-FFFF

Subroutine Return (RT)

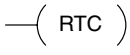
DS	Used
HPP	Used

When a Subroutine Return is executed in the subroutine the CPU will return to the point in the main body of the program from which it was called. The Subroutine Return is used as termination of the subroutine. It must be the last instruction in the subroutine and is a stand alone instruction (no input contact on the rung).

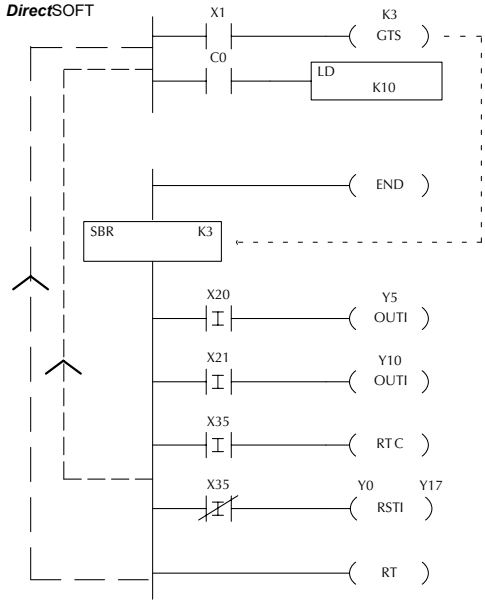
Subroutine Return Conditional (RTC)

DS	Used
HPP	Used

The Subroutine Return Conditional instruction is an optional instruction used with an input contact to implement a conditional return from the subroutine. The Subroutine Return (RT) is still required for termination of the Subroutine.



In the following example, when X1 is on, Subroutine K3 will be called. The CPU will jump to the Subroutine Label K3 and the ladder logic in the subroutine will be executed. If X35 is on, the CPU will return to the main program at the RTC instruction. If X35 is not on, Y0–Y17 will be reset to off and the CPU will return to the main body of the program.



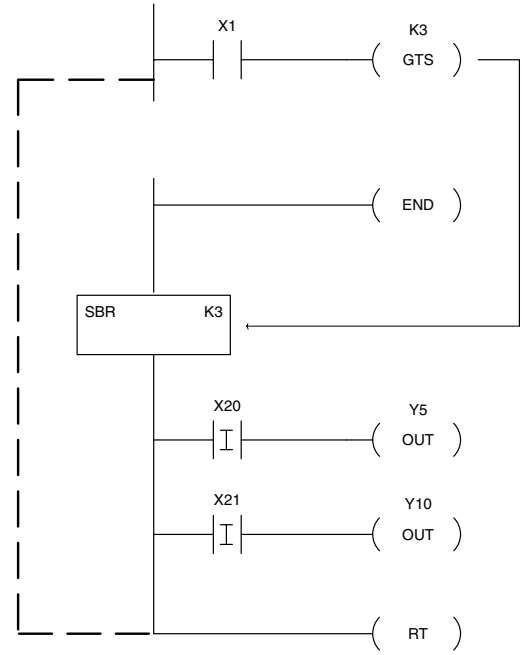
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT				
SHFT	G 6	T MLR	S RST	→	D 3	ENT	

SHFT	E 4	N TMR	D 3	ENT			
SHFT	S RST	SHFT	B 1	R ORN	→	D 3	ENT
$ STR	SHFT	I 8	→	C 2	A 0	ENT	
GX OUT	SHFT	I 8	→	F 5	ENT		
$ STR	SHFT	I 8	→	C 2	B 1	ENT	
GX OUT	SHFT	I 8	→	B 1	A 0	ENT	
$ STR	SHFT	I 8	→	D 3	F 5	ENT	
SHFT	R ORN	T MLR	C 2	ENT			
SP STRN	SHFT	I 8	→	D 3	F 5	ENT	
S RST	SHFT	I 8	→	A 0	→	B 1	H 7	ENT
SHFT	R ORN	T MLR	ENT				

In the following example, when X1 is on, Subroutine K3 will be called. The CPU will jump to the Subroutine Label K3 and the ladder logic in the subroutine will be executed. The CPU will return to the main body of the program after the RT instruction is executed.

DirectSOFT



Handheld Programmer Keystrokes

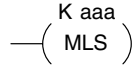
$ STR	→	B 1	ENT				
SHFT	G 6	T MLR	S RST	→	D 3	ENT	

SHFT	E 4	N TMR	D 3	ENT			
SHFT	S RST	SHFT	B 1	R ORN	→	D 3	ENT
$ STR	SHFT	I 8	→	C 2	A 0	ENT	
GX OUT	→	F 5	ENT				
$ STR	SHFT	I 8	→	C 2	B 1	ENT	
GX OUT	→	B 1	A 0	ENT			
SHFT	R ORN	T MLR	ENT				

Master Line Set (MLS)

DS	Used
HPP	Used

The Master Line Set instruction allows the program to control sections of ladder logic by forming a new power rail controlled by the main left power rail. The main left rail is always master line 0. When an MLS K1 instruction is used, a new power rail is created at level 1. Master Line Sets and Master Line Resets can be used to nest power rails up to seven levels deep.

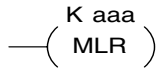


Operand Data Type		DL06 Range
		aaa
Constant	K	1-FFFF

Master Line Reset (MLR)

DS	Used
HPP	Used

The Master Line Reset instruction marks the end of control for the corresponding MLS instruction. The MLR reference is one less than the corresponding MLS.

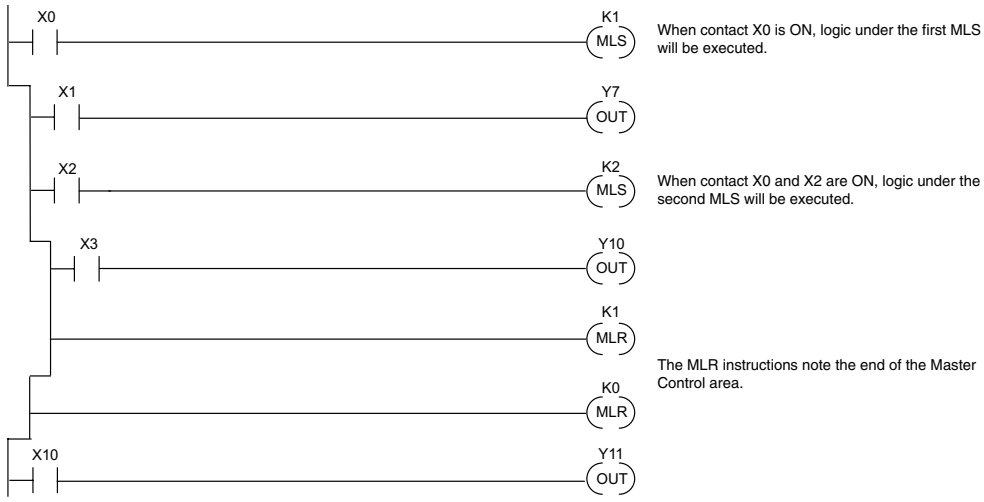


Operand Data Type		DL06 Range
		aaa
Constant	K	1-FFFF

Understanding Master Control Relays

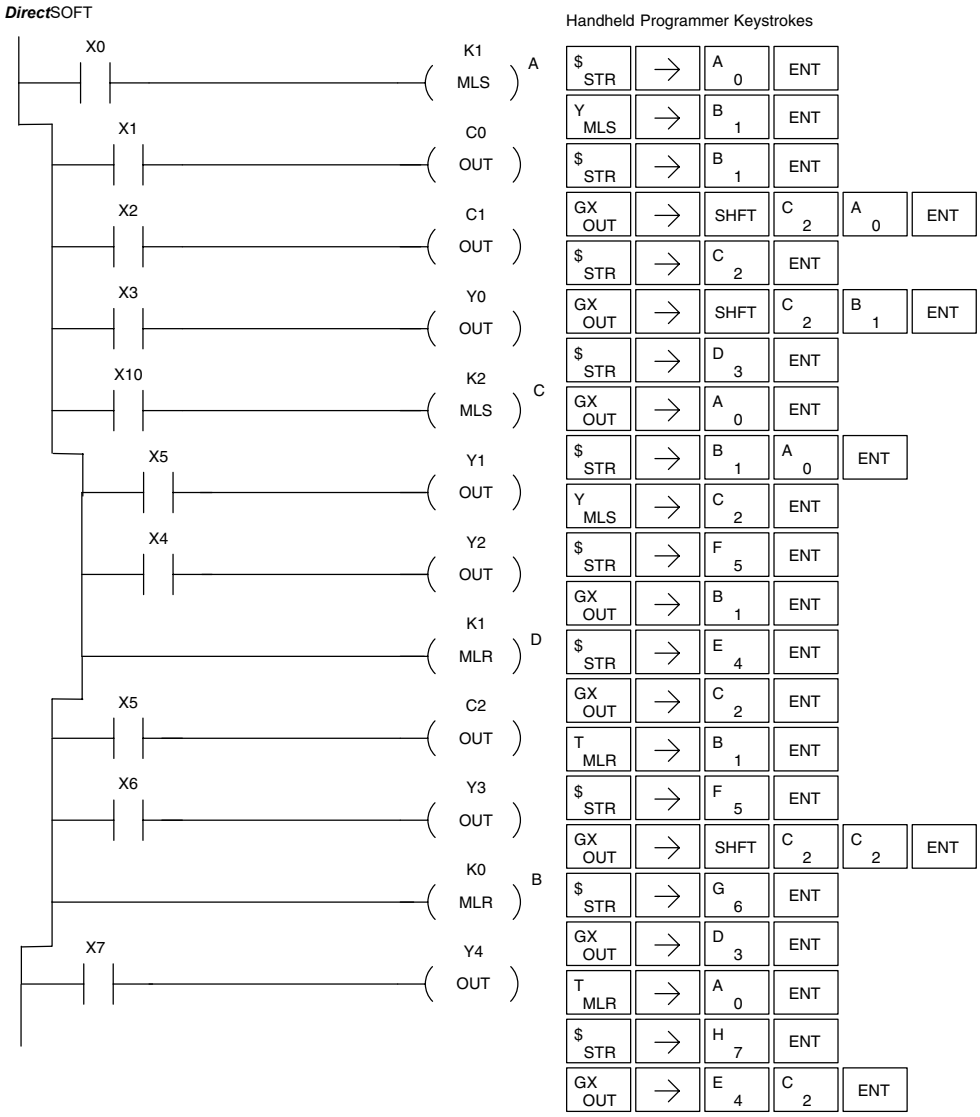
The Master Line Set (MLS) and Master Line Reset (MLR) instructions allow you to quickly enable (or disable) sections of the RLL program. This provides program control flexibility. The following example shows how the MLS and MLR instructions operate by creating a sub power rail for control logic.

DirectSOFT



MLS/MLR Example

In the following MLS/MLR example logic between the first MLS K1 (A) and MLR K0 (B) will function only if input X0 is on. The logic between the MLS K2 (C) and MLR K1 (D) will function only if input X10 and X0 is on. The last rung is not controlled by either of the MLS coils.

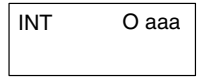


Interrupt Instructions

Interrupt (INT)

DS	Used
HPP	Used

The Interrupt instruction allows a section of ladder logic to be placed below the main body of the program and executed only when needed. High-Speed I/O Modes 10, 20, and 40 can generate an interrupt. With Mode 40, you may select an external interrupt (input X0), or a time-based interrupt (3–999 ms).



Typically, interrupts are used in an application when a fast response to an input is needed or a program section must execute faster than the normal CPU scan. The interrupt label and all associated logic must be placed after the End statement in the program. When an interrupt occurs, the CPU will complete execution of the current instruction it is processing in ladder logic, then execute the interrupt routine. After interrupt routine execution, the ladder program resumes from the point at which it was interrupted.

See Chapter 3, the section on Mode 40 (Interrupt) Operation for more details on interrupt configuration. In the DL06, only one software interrupt is available. The software interrupt uses interrupt #00 (INT 0), which means the hardware interrupt #0 and the software interrupt cannot be used together. Hardware interrupts are labeled in octal to correspond with the hardware input signal (e.g. X1 will initiate INT 1).

Operand Data Type		DL06 Range
		aaa
Constant	O	1-FFFF

Interrupt Return (IRT)

DS	Used
HPP	Used

An Interrupt Return is normally executed as the last instruction in the interrupt routine. It returns the CPU to the point in the main program from which it was called. The Interrupt Return is a stand-alone instruction (no input contact on the rung).

Interrupt Return Conditional (IRTC)

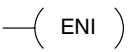
DS	Used
HPP	Used

The Interrupt Return Conditional instruction is a optional instruction used with an input contact to implement a conditional return from the interrupt routine. The Interrupt Return is required to terminate the interrupt routine.

Enable Interrupts (ENI)

DS	Used
HPP	Used

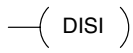
The Enable Interrupt instruction is placed in the main ladder program (before the End instruction), enabling the interrupt. The interrupt remains enabled until the program executes a Disable Interrupt instruction.



Disable Interrupts (DISI)

DS	Used
HPP	Used

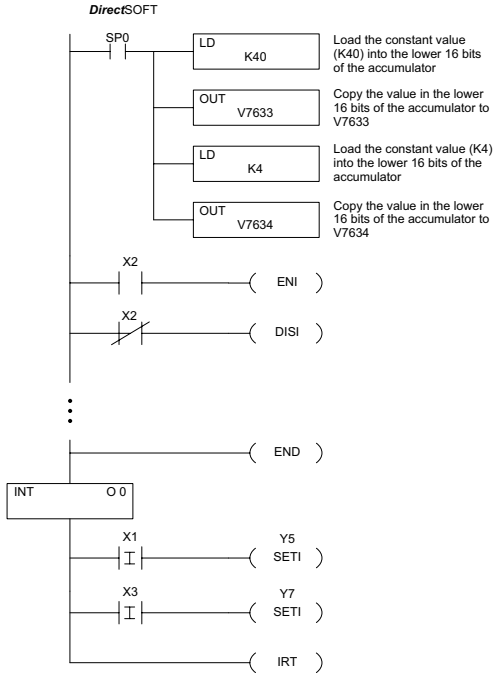
A Disable Interrupt instruction in the main body of the application program (before the End instruction) will disable the interrupt (either external or timed). The interrupt remains disabled until the program executes an Enable Interrupt instruction.



External Interrupt Program Example

In the following example, we do some initialization on the first scan, using the first-scan contact SP0. The interrupt feature is the HSIO Mode 40. Then, we configure X0 as the external interrupt by writing to its configuration register, V7634. See Appendix E, Mode 40 Operation for more details.

During program execution, when X2 is on, the interrupt is enabled. When X2 is off, the interrupt will be disabled. When an interrupt signal (X0) occurs, the CPU will jump to the interrupt label INT O 0. The application ladder logic in the interrupt routine will be performed. The CPU will return to the main body of the program after the IRT instruction is executed.



Handheld Programmer Keystrokes

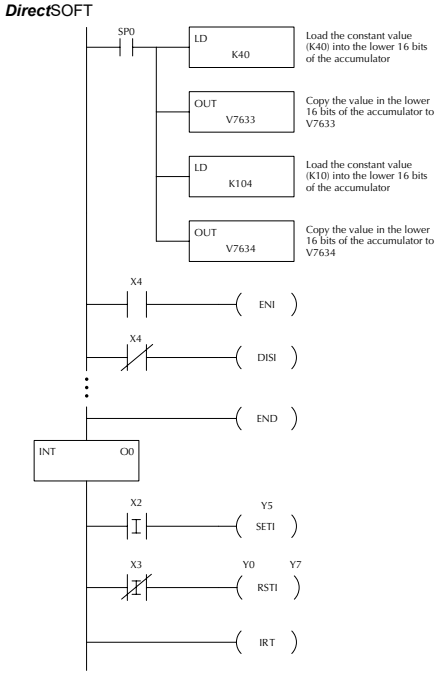
$ STR	→	SHFT	SP STRN	A 0	ENT					
SHFT	L ANDST	D 3	→	SHFT	K JMP	E 4	A 0	ENT		
GX OUT	→	SHFT	V AND	H 7	G 6	D 3	D 3	ENT		
SHFT	L ANDST	D 3	→	SHFT	K JMP	E 4	ENT			
GX OUT	→	SHFT	V AND	H 7	G 6	D 3	E 4	ENT		
$ STR	→	C 2	ENT							
SHFT	E 4	N TMR	I 8	ENT						
SP STRN	→	C 2	ENT							
SHFT	D 3	I 8	S RST	I 8	ENT					

SHFT	E 4	N TMR	D 3	ENT			
SHFT	I 8	N TMR	T MLR	→	A 0	ENT	
$ STR	SHFT	I 8	→	B 1	ENT		
X SET	SHFT	I 8	→	F 5	ENT		
$ STR	SHFT	I 8	→	D 3	ENT		
X SET	SHFT	I 8	→	H 7	ENT		
SHFT	I 8	R ORN	T MLR	ENT			

Timed Interrupt Program Example

In the following example, we do some initialization on the first scan, using the first-scan contact SP0. The interrupt feature is the HSIO Mode 40. Then we configure the HSIO timer as a 10 mS interrupt by writing K104 to the configuration register for X0 (V7634). See Appendix E, Mode 40 Operation for more details.

When X4 turns on, the interrupt will be enabled. When X4 turns off, the interrupt will be disabled. Every 10 mS the CPU will jump to the interrupt label INT O 0. The application ladder logic in the interrupt routine will be performed. If X3 is not on, Y0–Y7 will be reset to off and then the CPU will return to the main body of the program.



Handheld Programmer Keystrokes

$ STR	→	B 1	ENT												
SHFT	L ANDST	D 3	→	SHFT	K JMP	E 4	A 0	ENT							
GX OUT	→	SHFT	V AND	H 7	G 6	D 3	D 3	ENT							
SHFT	L ANDST	D 3	→	SHFT	K JMP	B 1	A 0	E 4	ENT						
GX OUT	→	SHFT	V AND	H 7	G 6	D 3	E 4	ENT							
$ STR	→	E 4	ENT												
SHFT	E 4	N TMR	I 8	ENT											
SP STRN	→	E 4	ENT												
SHFT	D 3	I 8	S RST	I 8	ENT										

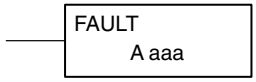
SHFT	E 4	N TMR	D 3	ENT											
SHFT	I 8	N TMR	T MLR	→	A 0	ENT									
$ STR	SHFT	I 8	→	C 2	ENT										
X SET	SHFT	I 8	→	F 5	ENT										
SP STRN	SHFT	I 8	→	D 3	ENT										
X SET	SHFT	I 8	→	A 0	→	H 7	ENT								
SHFT	I 8	R ORN	T MLR	ENT											

Message Instructions

Fault (FAULT)

DS	Used
HPP	Used

The Fault instruction is used to display a message on the handheld programmer, the optional LCD display or in the *Direct*SOFT status bar. The message has a maximum of 23 characters and can be either V-memory data, numerical constant data or ASCII text.



To display the value in a V-memory location, specify the V-memory location in the instruction. To display the data in ACON (ASCII constant) or NCON (Numerical constant) instructions, specify the constant (K) value for the corresponding data label area.

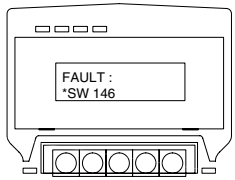
See Appendix G for the ASCII conversion table.

Operand Data Type		DL06 Range
		aaa
V-memory	V	See memory map
Constant	K	1-FFFF

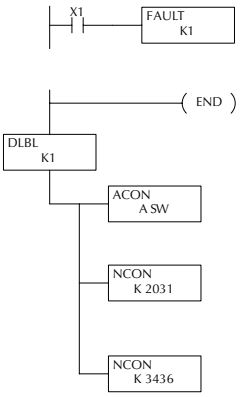
Discrete Bit Flags	Description
SP50	On when the FAULT instruction is executed

Fault Example

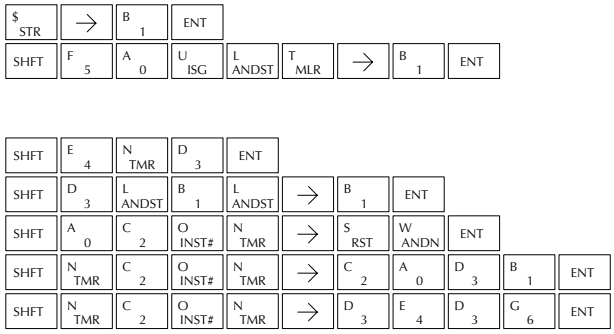
In the following example when X1 is on, the message SW 146 will display on the handheld programmer. The NCONs use the HEX ASCII equivalent of the text to be displayed. (The HEX ASCII for a blank is 20 a 1 is 31, 4 is 34 ...)



*Direct*SOFT



Handheld Programmer Keystrokes



Data Label (DLBL)

DS	Used
HPP	Used

The Data Label instruction marks the beginning of an ASCII/ numeric data area. DLBLs are programmed after the End statement. A maximum of 64 DLBL instructions can be used in a program. Multiple NCONs and ACONs can be used in a DLBL area.

```
DLBL
     K aaa
```

Operand Data Type		DL06 Range
		aaa
Constant	K	1-FFFF

DS	Used
HPP	Used

ASCII Constant (ACON)

```
ACON
     A aaa
```

The ASCII Constant instruction is used with the DLBL instruction to store ASCII text for use with other instructions. Two ASCII characters can be stored in an ACON instruction. If only one character is stored in a ACON a leading space will be inserted.

Operand Data Type		DL06 Range
		aaa
ASCII	A	0-9 A-Z

DS	Used
HPP	Used

Numerical Constant (NCON)

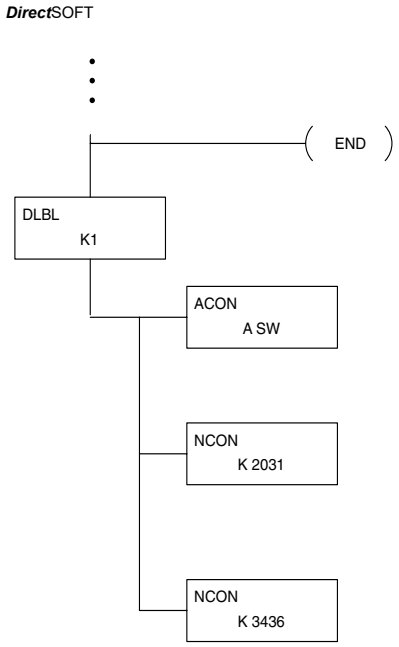
```
NCON
     K aaa
```

The Numerical Constant instruction is used with the DLBL instruction to store the HEX ASCII equivalent of numerical data for use with other instructions. Two digits can be stored in an NCON instruction.

Operand Data Type		DL06 Range
		aaa
Constant	K	1-FFFF

Data Label Example

In the following example, an ACON and two NCON instructions are used within a DLBL instruction to build a text message. See the FAULT instruction for information on displaying messages. The DV-1000 Manual also has information on displaying messages.



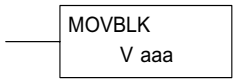
Handheld Programmer Keystrokes

SHFT	E 4	N TMR	D 3	ENT						
SHFT	D 3	L ANDST	B 1	L ANDST	→	B 1	ENT			
SHFT	A 0	C 2	O INST#	N TMR	→	S RST	W ANDN	ENT		
SHFT	N TMR	C 2	O INST#	N TMR	→	C 2	A 0	D 3	B 1	ENT
SHFT	N TMR	C 2	O INST#	N TMR	→	D 3	E 4	D 3	G 6	ENT

Move Block Instruction (MOVBLK)

DS	Used
HPP	Used

The Move Block instruction copies a specified number of words from a Data Label Area of program memory (ACON, NCON) to the specified V-memory location.

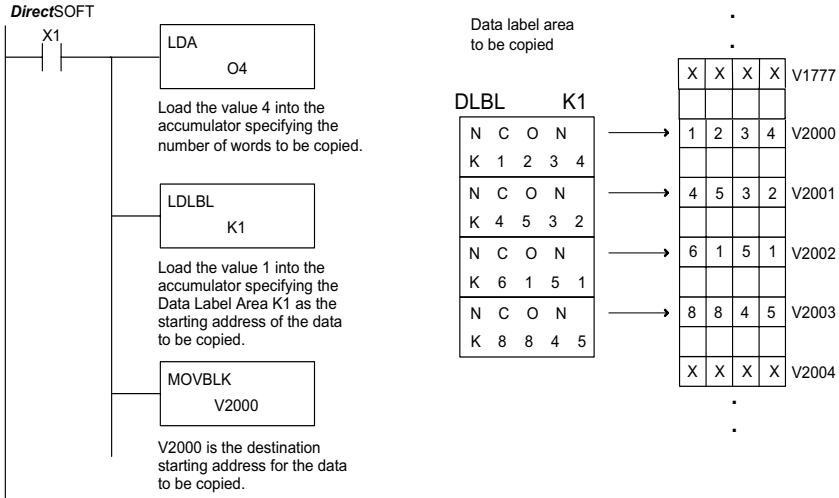


Below are the steps for using the Move Block function:

- Step 1: Load the number of words (octal) to be copied into the 1st level of the accumulator stack.
- Step 2: Load the source data label (LDLBL Kaaa) into the accumulator. This is where the data will be copied from.
- Step 3: Insert the MOVBLK instruction that specifies the V-memory destination. This is where the data will be copied to.

Copy Data From a Data Label Area to V-memory

In the example below, data is copied from a Data Label Area to V-memory. When X1 is on, the octal value (O4) is copied to the first level of the accumulator stack using the Load Address (LDA) instruction. This value specifies the number of words to be copied. Load Label (LDLBL) instruction will load the source data address (K1) into the accumulator. This is where the data will be copied from. The MOVBLK instruction specifies the destination starting location and executes the copying of data from the Data Label Area to V-memory.



Handheld Programmer Keystrokes

$ STR	→	B 1	ENT									
SHFT	L ANDST	D 3	A 0	→	E 4	ENT						
SHFT	L ANDST	D 3	L ANDST	B 1	L ANDST	→	B 1	ENT				
SHFT	M ORST	O INST#	V AND	B 1	L ANDST	K JMP	→	C 2	A 0	A 0	A 0	ENT

Print Message (PRINT)

DS	Used
HPP	N/A

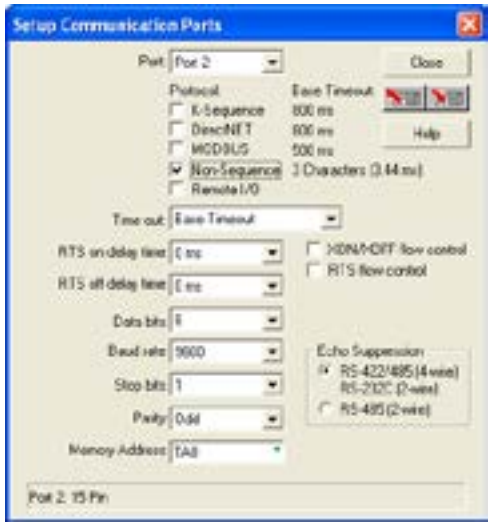
The Print Message instruction prints the embedded text or text/data variable message (maximum 128 characters) to the specified communications port (Port 2 on the DL06 CPU), which must have the communications port configured.

```
PRINT     A aaa
"Hello, this is a PLC message"
```

Operand Data Type		DL06 Range
		aaa
Constant	K	2

You may recall, from the CPU specifications in Chapter 3, that the DL06's ports are capable of several protocols. Port 1 cannot be configured for the non-sequence protocol. To configure port 2 using the Handheld Programmer, use AUX 56 and follow the prompts, making the same choices as indicated below on this page. To configure a port in ***Direct***SOFT, choose the PLC menu, then Setup, then Setup Secondary Comm Port.

- **Port**: From the port number list box at the top, choose **Port 2**.
- **Protocol**: Click the check box to the left of **Non-sequence**, and then you'll see the dialog box shown below.



- **Baud Rate**: Choose the baud rate that matches your printer.
- **Stop Bits, Parity**: Choose number of stop bits and parity setting to match your printer.
- **Memory Address**: Choose a V-memory address for ***Direct***SOFT to use to store the port setup information. You will need to reserve 66 contiguous words in V-memory for this purpose.

Before ending the setup, click the button indicated to send Port 2 configuration to the CPU, and click **Close**. See Chapter 3 for port wiring information, in order to connect your printer to the DL06.

Port 2 on the DL06 has standard RS232 levels, and should work with most printer serial input connections.

Text element – this is used for printing character strings. The character strings are defined as the character (more than 0) ranged by the double quotation marks. Two hex numbers preceded by the dollar sign means an 8-bit ASCII character code. Also, two characters preceded by the dollar sign is interpreted according to the following table:

#	Character code	Description
1	$$	Dollar sign ($)
2	$"	Double quotation (")
3	$L or $l	Line feed (LF)
4	$N or $n	Carriage return line feed (CRLF)
5	$P or $p	Form feed
6	$R or $r	Carriage return (CR)
7	$T or $t	Tab

The following examples show various syntax conventions and the length of the output to the printer.

Example:

" " Length 0 without character

"A" Length 1 with character A

" " Length 1 with blank

" $" " Length 1 with double quotation mark

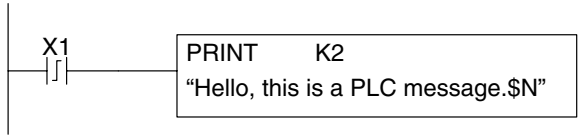
" $ R $ L " Length 2 with one CR and one LF

" $ 0 D $ 0 A " Length 2 with one CR and one LF

" $ $ " Length 1 with one $ mark

In printing an ordinary line of text, you will need to include **double quotation** marks before and after the text string. Error code 499 will occur in the CPU when the print instruction contains invalid text or no quotations. It is important to test your PRINT instruction data during the application development.

The following example prints the message to port 2. We use a PD contact, which causes the message instruction to be active for just one scan. Note the $N at the end of the message, which produces a carriage return / line feed on the printer. This prepares the printer to print the next line, starting from the left margin.



Print the message to Port 2 when X1 makes an off-to-on transition.

V-memory element - this is used for printing V-memory contents in the integer format or real format. Use V-memory number or V-memory number with ":" and data type. The data types are shown in the table below. The Character code must be capital letters.

***NOTE**: There must be a space entered before and after the V-memory address to separate it from the text string. Failure to do this will result in an error code 499.*

#	Character code	Description
1	none	16-bit binary (decimal number)
2	: B	4 digit BCD
3	: D	32-bit binary (decimal number)
4	: D B	8 digit BCD
5	: R	Floating point number (real number)
6	: E	Floating point number (real number with exponent)

Example:

V2000 Print binary data in V2000 for decimal number

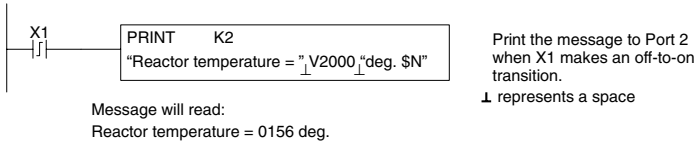
V2000 : B Print BCD data in V2000

V2000 : D Print binary number in V2000 and V2001 for decimal number

V2000 : D B Print BCD data in V2000 and V2001

V2000 : R Print floating point number in V2000/V2001 as real number

V2000 : E Print floating point number in V2000/V2001 as real number with exponent



Example: The following example prints a message containing text and a variable. The "reactor temperature" labels the data, which is at V2000. You can use the : B qualifier after the V2000 if the data is in BCD format, for example. The final string adds the units of degrees to the line of text, and the $N adds a carriage return / line feed.

V-memory text element - This is used for printing text stored in V-memory. Use the % followed by the number of characters after V-memory number for representing the text. If you assign "0" as the number of characters, the print function will read the character count from the first location. Then it will start at the next V-memory location and read that number of ASCII codes for the text from memory.

Example:

V2000 % 16 16 characters in V2000 to V2007 are printed.

V2000 % 0 The characters in V2001 to Vxxxx (determined by the number in V2000) will be printed.

Bit element

This is used for printing the state of the designated bit in V-memory or a relay bit. The bit element can be assigned by the designating point (.) and bit number preceded by the V-memory number or relay number. The output type is described as shown in the table below.

#	Data Format	Description
1	none	Print 1 for an ON state, and 0 for an OFF state
2	:BOOL	Print "TRUE" for an ON state, and "FALSE" for an OFF state
3	:ONOFF	Print "ON" for an ON state, and "OFF" for an OFF state

Example:

V2000 . 15 Prints the status of bit 15 in V2000, in 1/0 format

C100 Prints the status of C100 in 1/0 format

C100 : BOOL Prints the status of C100 in TRUE/FALSE format

C100 : ON/OFF Prints the status of C100 in ON/OFF format

V2000.15 : BOOL Prints the status of bit 15 in V2000 in TRUE/FALSE format

The maximum numbers of characters you can print is 128. The number of characters for each element is listed in the table below:

Element Type	Maximum Characters
Text, 1 character	1
16 bit binary	6
32 bit binary	11
4 digit BCD	4
8 digit BCD	8
Floating point (real number)	12
Floating point (real with exponent)	12
V-memory/text	2
Bit (1/0 format)	1
Bit (TRUE/FALSE format)	5
Bit (ON/OFF format)	3

The handheld programmer's mnemonic is "PRINT" followed by the DEF field.

Special relay flags SP116 and SP117 indicate the status of the DL06 CPU ports (busy, or communications error). See the appendix on special relays for a description.

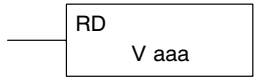
NOTE: *You must use the appropriate special relay in conjunction with the PRINT command to ensure the ladder program does not try to PRINT to a port that is still busy from a previous PRINT or WX or RX instruction.*

Intelligent I/O Instructions

Read from Intelligent Module (RD)

DS32	Used
HPP	Used

The Read from Intelligent Module instruction reads a block of data (1-128 bytes maximum) from an intelligent I/O module into the CPU's V-memory. It loads the function parameters into the first and second level of the accumulator stack and the accumulator by three additional instructions.



Listed below are the steps to program the Read from Intelligent module function.

Step 1: Load the base number (0-3) into the first byte and the slot number (0-7) into the second byte of the second level of the accumulator stack.

Step 2: Load the number of bytes to be transferred into the first level of the accumulator stack (maximum of 128 bytes).

Step 3: Load the address from which the data will be read into the accumulator. This parameter must be a HEX value.

Step 4: Insert the RD instruction which specifies the starting V-memory location (Vaaa) where the data will be read into.

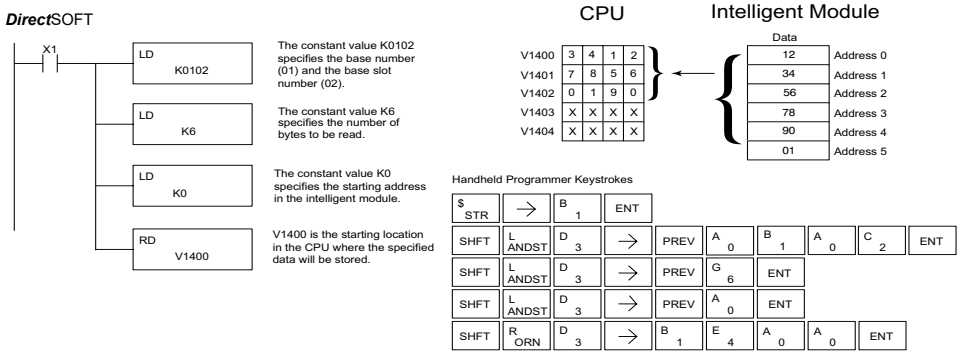
Helpful Hint: – Use the LDA instruction to convert an octal address to its HEX equivalent and load it into the accumulator when the HEX format is required.

Operand Data Type		DL06 Range
		aaa
V-memory	V	See memory map

Discrete Bit Flags	Description
SP54	On when RX, WX RD, WT instructions are executed with the wrong parameters.

NOTE: *Status flags are valid only until another instruction uses the same flag.*

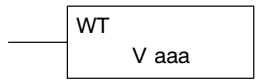
In the following example, when X1 is ON, the RD instruction will read six bytes of data from a intelligent module in base 1, slot 2, starting at address 0 in the intelligent module, and copy the information into V-memory locations V1400-V1402.



Write to Intelligent Module (WT)

DS32	Used
HPP	Used

The Write to Intelligent Module instruction writes a block of data (1-128 bytes maximum) to an intelligent I/O module from a block of V-memory in the CPU. The function parameters are loaded into the first and second level of the accumulator stack and the accumulator by three additional instructions.



Listed below are the steps to program the Read from Intelligent module function.

Step 1: Load the base number (0-3) into the first byte and the slot number (0-7) into the second byte of the second level of the accumulator stack.

Step 2: Load the number of bytes to be transferred into the first level of the accumulator stack (maximum of 128 bytes).

Step 3: Load the intelligent module address which will receive the data into the accumulator. This parameter must be a HEX value.

Step 4: Insert the WT instruction which specifies the starting V-memory location (Vaaa) where the data will be written from in the CPU.

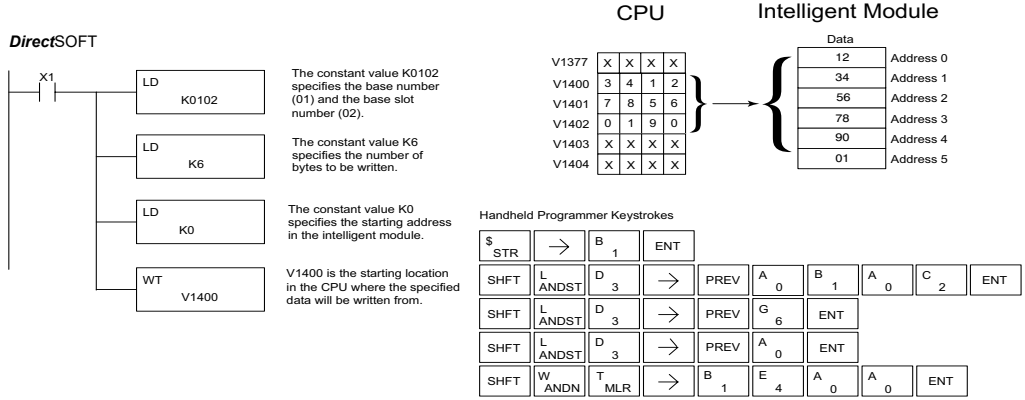
Helpful Hint: – Use the LDA instruction to convert an octal address to its HEX equivalent and load it into the accumulator when the HEX format is required.

Operand Data Type		DL06 Range
		aaa
V-memory	V	See memory map

Discrete Bit Flags	Description
SP54	On when RX, WX RD, WT instructions are executed with the wrong parameters.

***NOTE:** Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the WT instruction will write six bytes of data to an intelligent module in base 1, slot 2,starting at address 0 in the intelligent module, and copy the data from V-memory locations V1400-V1402.

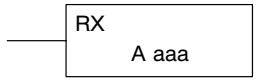


Network Instructions

Read from Network (RX)

DS32	Used
HPP	Used

The Read from Network instruction is used by the master device on a network to read a block of data from a slave device on the same network. The function parameters are loaded into the first and second level of the accumulator stack and the accumulator by three additional instructions.



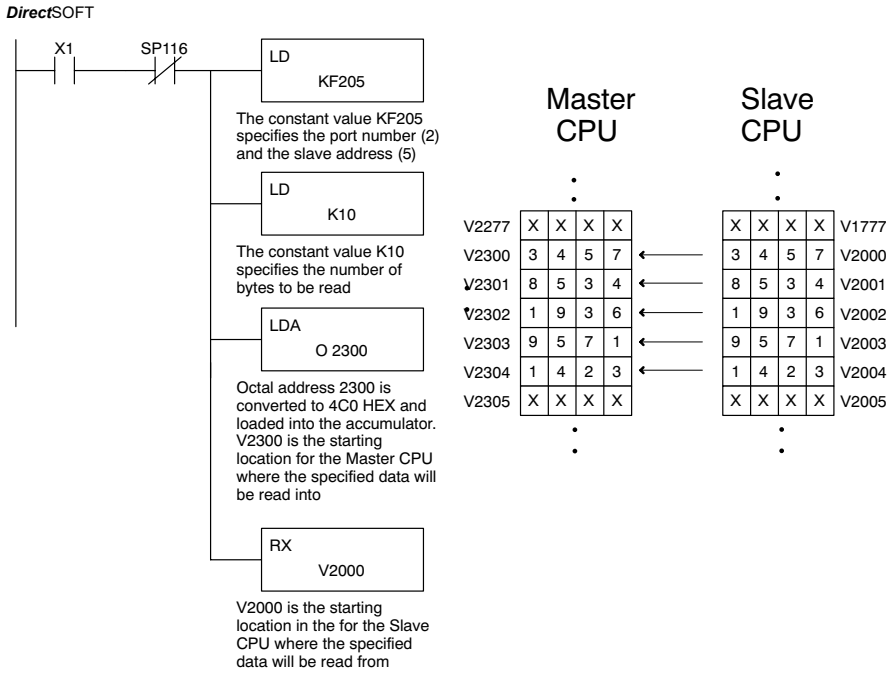
Listed below are the steps necessary to program the Read from Network function.

- Step 1: Load the slave address (0-- 90 BCD) into the first byte and the PLC internal port (KF2) or slot number of the master DCM or ECOM (0-- 7) into the second byte of the second level of the accumulator stack.
- Step 2: Load the number of bytes to be transferred into the first level of the accumulator stack.
- Step 3: Load the address of the data to be read into the accumulator. This parameter requires a HEX value.
- Step 4: Insert the RX instruction which specifies the starting Vmemory location (Aaaa) where the data will be read from in the slave.

Helpful Hint: — For parameters that require HEX values, the LDA instruction can be used to convert an octal address to the HEX equivalent and load the value into the accumulator.

Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Inputs	X	0–777
Outputs	Y	0–777
Control Relays	C	0–1777
Stage	S	0–1777
Timer	T	0–377
Counter	CT	0–177
Special Relay	SP	0–777
Program Memory	$	0–7680 (2K program mem.)

In the following example, when X1 is on and the port busy relay SP116 (see special relays) is not on, the RX instruction will access port 2 operating as a master. Ten consecutive bytes of data (V2000 – V2004) will be read from a CPU at station address 5 and copied into V-memory locations V2300–V2304 in the CPU with the master port.



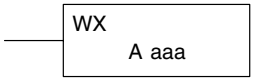
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT									
W ANDN	→	SHFT	SP STRN	B 1	B 1	G 6	ENT					
SHFT	L ANDST	D 3	→	SHFT	K JMP	SHFT	F 5	SHFT	C 2	A 0	F 5	ENT
SHFT	L ANDST	D 3	→	SHFT	K JMP	B 1	A 0	ENT				
SHFT	L ANDST	D 3	A 0	→	C 2	D 3	A 0	A 0	ENT			
SHFT	R ORN	X SET	→	C 2	A 0	A 0	A 0	ENT				

Write to Network (WX)

DS	Used
HPP	Used

The Write to Network instruction is used to write a block of data from the master device to a slave device on the same network. The function parameters are loaded into the accumulator and the first and second levels of the stack.



Listed below are the program steps necessary to execute the Write to Network function.

Step 1: Load the slave address (0–90 BCD) into the low byte and "F2" into the high byte of the accumulator (the next two instructions push this word down to the second layer of the stack).

Step 2: Load the number of bytes to be transferred into the accumulator (the next instruction pushes this word onto the top of the stack).

Step 3: Load the starting Master CPU address into the accumulator. This is the memory location where the data will be written from. This parameter requires a HEX value.

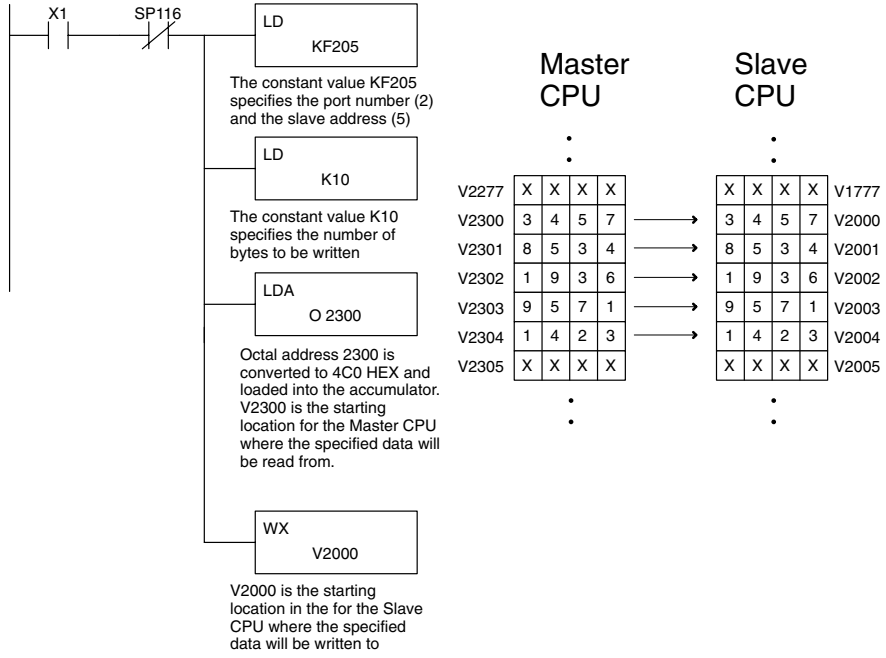
Step 4: Insert the WX instruction which specifies the starting V-memory location (Aaaa) where the data will be written to in the slave.

Helpful Hint: — For parameters that require HEX values, the LDA instruction can be used to convert an octal address to the HEX equivalent and load the value into the accumulator.

Operand Data Type		DL06 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Inputs	X	0–777
Outputs	Y	0–777
Control Relays	C	0–1777
Stage	S	0–1777
Timer	T	0–377
Counter	CT	0–177
Special Relay	SP	0–777
Program Memory	$	0–7680 (2K program mem.)

In the following example, when X1 is on and the module busy relay SP116 (see special relays) is not on, the WX instruction will access port 2 operating as a master. Ten consecutive bytes of data are read from the Master CPU and copied to V-memory locations V2000–V2004 in the slave CPU at station address 5.

DirectSOFT



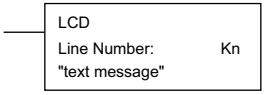
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT														
W ANDN	→	SHFT	SP STRN	B 1	C 1	E 6	ENT										
SHFT	L ANDST	D 3	→	SHFT	K JMP	SHFT	F 5	SHFT	C 2	A 0	F 5	ENT					
SHFT	L ANDST	D 3	→	SHFT	K JMP	B 1	A 0	ENT									
SHFT	L ANDST	D 3	A 0	→	C 2	D 3	A 0	A 0	ENT								
SHFT	W ANDN	X SET	→	C 2	A 0	A 0	A 0	ENT									

LCD

DS	Used
HPP	N/A

When enabled, the LCD instruction causes a user-defined text message to be displayed on the LCD Display Panel. The display is 16 characters wide by 2 rows high so a total of 32 characters can be displayed. Each row is addressed separately; the maximum number of characters the instruction will accept is 16.



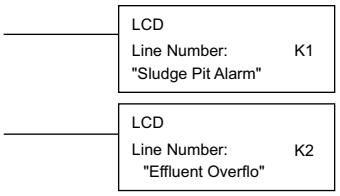
The text message can be entered directly into the message field of the instruction set-up dialog, or it can be located anywhere in user V-memory. If the text is located in V-memory, the LCD instruction is used to point to the memory location where the desired text originates. The length of the text string is also required.

From the *Direct*SOFT project folder, use the Instruction Browser to locate the LCD instruction. When you select the LCD instruction and click OK, the LCD dialog will appear, as shown in the examples. The LCD instruction is inserted into the ladder program via this set-up dialog box.

Display text strings can include embedded variables. Date and time settings and V-memory values can be embedded in the displayed text. Examples of each are shown.

Direct Text Entry

The two dialogs to the right show the selections necessary to create the two ladder instructions below. Double quotation marks are required to delineate the text string. In the first dialog, the text "Sludge Pit Alarm" uses sixteen character spaces and will appear on line 1 when the instruction is enabled. Note, the line number is K1. Clicking the "check" button causes the instruction to be inserted into the ladder program.



By identifying the second Line Number as K2, the text string "Effluent Overflow" will appear on the second line of the display when the second instruction is enabled.

S	l	u	d	g	e		P	i	t		A	l	a	r	m
E	f	f	l	u	e	n	t		O	v	e	r	f	l	o

Embedding date and/or time variables

The date and/or time can be embedded in the displayed text by using the variables listed in the table below. These variables can be included in the **LCD message** field of the LCD dialog. In the example, the time variable (12 hour format) is embedded by adding _time:12. This time format uses a maximum of seven character spaces. The second dialog creates an instruction that prints the date on the second line of the display, when enabled.

Date and Time Variables and Formats		
_date:us	US format	MM/DD/YY
_date:e	European format	DD/MM/YY
_date:a	Asian format	YY/MM/DD
_time:12	12 hour format	HH:MMAM/PM
_time:24	24 hour format	HH:MM:SS


```
LCD
Line Number:          K1
"Alarm 1 "   _time:12
```

```
LCD
Line Number:          K2
  _date:us
```

A	l	a	r	m		1			1	1	:	2	1	P	M
0	5	-	0	8	-	0	2								

Embedding V-memory data

Any V-memory data can be displayed in any one of six available data formats. An example appears to the right. A list of data formats and modifiers is on the next page. Note that different data formats require differing numbers of character positions on the display.


```
LCD
Line Number:          K1
"Count = " V2500:B
```

C	o	u	n	t		=		0	4	1	2				

Data Format Suffixes for Embedded V-memory Data

Several data formats are available for displaying V-memory data on the LCD. The choices are shown in the table below. A colon is used to separate the embedded V-memory location from the data format suffix and modifier. An example appears on the previous page.

Data Format	Modifier	Example	Displayed Characters												
none (16-bit format)		V2000 = 0000 0000 0001 0010	1	2	3	4									
		V2000			1	8									
	[:S]	V2000:S	1	8											
	[:C0]	V2000:C0	0	0	1	8									
	[:0]	V2000:0			1	8									
:B (4 digit BCD)		V2000 = 0000 0000 0001 0010	1	2	3	4									
	[:B]	V2000:B	0	0	1	2									
	[:BS]	V2000:BS	1	2											
	[:BC0]	V2000:BC0	0	0	1	2									
	[:B0]	V2000:B0			1	2									
:D (32-bit decimal)		V2000 = 0000 0000 0000 0000	Double Word												
		V2001 = 0000 0000 0000 0001	1	2	3	4	5	6	7	8	9	10	11		
	[:D]	V2000:D							6	5	5	3	6		
	[:DS]	V2000:DS	6	5	5	3	6								
	[:DC0]	V2000:DC0	0	0	0	0	0	0	6	5	5	3	6		
	[:D0]	V2000:D0							6	5	5	3	6		
:DB (8 digit BCD)		V2000 = 0000 0000 0000 0000	Double Word												
		V2001 = 0000 0000 0000 0011	1	2	3	4	5	6	7	8					
	[:DB]	V2000:DB	0	0	0	3	0	0	0	0					
	[:DBS]	V2000:DBS	3	0	0	0	0								
	[:DBC0]	V2000:DBC0	0	0	0	3	0	0	0	0					
	[:DB0]	V2000:DB0				3	0	0	0	0					
:R (DWord floating point number)		V2001/V2000 = 222.11111	Double Word												
		(real number)	1	2	3	4	5	6	7	8	9	10	11	12	13
	[:R]	V2000:R				f	2	2	2	.	1	1	1	1	1
	[:RS]	V2000:RS	f	2	2	2	.	1	1	1	1	1			
	[:RC0]	V2000:RC0	f	0	0	0	2	2	2	.	1	1	1	1	1
	[:R0]	V2000:R0				f	2	2	2	.	1	1	1	1	1
:E (DWord floating point number with exponent)		V2001/V2000 = 222.1	Double Word												
		(real number)	1	2	3	4	5	6	7	8	9	10	11	12	13
	[:E]	V2000:E		f	2	.	2	2	1	0	0	E	+	0	2
	[:ES]	V2000:ES	f	2	.	2	2	1	0	0	E	+	0	2	
	[:EC0]	V2000:EC0	f	2	.	2	2	1	0	0	E	+	0	2	
	[:E0]	V2000:E0	f	2	.	2	2	1	0	0	E	+	0	2	
f = plus/minus flag (plus = no symbol, minus = -)															

The S, C0, and 0 modifiers alter the presentation of leading zeros and spaces. S removes leading spaces and left justifies the result. C0 replaces leading spaces with leading zeros. 0 is a modification of C0. 0 eliminates any leading zeros in the C0 format version and converts them to spaces.

Text Entry from V-memory

Alternatively, text that resides in V-memory can be displayed on the LCD following the example on this page. The LCD dialog is used twice, once for each line on the display. The dialog requires the address of the first character to be displayed and the number of characters to be displayed.

For example, the two dialogs shown on this page would create the two LCD instructions below. When enabled, these instructions would cause the ASCII characters in V10000 to V10017 to be displayed. The ASCII characters and their corresponding memory locations are shown in the table below.

LCD
Line Number: K1
Starting V Memory Address: V10000
Number of Characters: K16

LCD
Line Number: K2
Starting V Memory Address: V10010
Number of Characters: K16

V10000	d	A
V10001	i	m
V10002		n
V10003	f	O
V10004	i	f
V10005	e	c
V10006		
V10007		
V10010	i	H
V10011	h	g
V10012	T	
V10013	m	e
V10014		p
V10015	l	A
V10016	r	a
V10017		m

A	d	m	i	n		O	f	f	i	c	e			
H	i	g	h		T	e	m	p		A	l	a	r	m

MODBUS RTU Instructions

MODBUS Read from Network (MRX)

DS	Used
HPP	N/A

The MODBUS Read from Network (MRX) instruction is used by the DL06 network master to read a block of data from a connected slave device and to write the data into V–memory addresses within the master. The instruction allows the user to specify the MODBUS Function Code, slave station address, starting master and slave memory addresses, number of elements to transfer, MODBUS data format and the Exception Response Buffer.

CPU/DCM: select either CPU or DCM module for communications

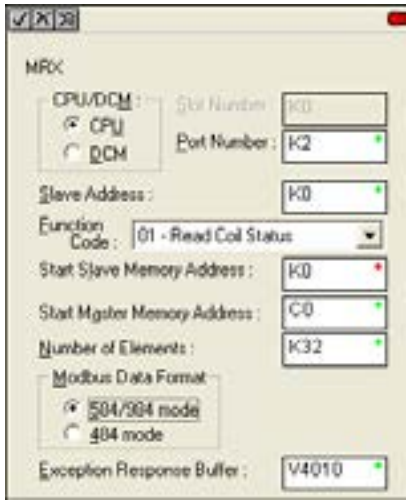
Slot Number: select PLC option slot number if using a DCM module.

Port Number: must be DL06 Port 2 (K2)

Slave Address: specify a slave station address (0–247)

Function Code: The following MODBUS function codes are supported by the MRX instruction:

- 01 – Read a group of coils
- 02 – Read a group of inputs
- 03 – Read holding registers
- 04 – Read input registers
- 07 – Read Exception status



Start Slave Memory Address: specifies the starting slave memory address of the data to be read. See the table on the following page.

Start Master Memory Address: specifies the starting memory address in the master where the data will be placed. See the table on the following page.

Number of Elements: specifies how many coils, inputs, holding registers or input register will be read. See the table on the following page.

MODBUS Data Format: specifies MODBUS 584/984 or 484 data format to be used

Exception Response Buffer: specifies the master memory address where the Exception Response will be placed (6-bytes in length). See the table on the following page. The exception response buffer uses 3 words. These bytes are swapped in the MRX/MWX exception response buffer V-memory so:

- V-Memory 1 Hi Byte = Function Code Byte (Most Significant Bit Set)
- V-Memory 1 Lo Byte = Address Byte
- V-Memory 2 Hi Byte = One of the CRC Bytes
- V-Memory 2 Lo Byte = Exception Code
- V-Memory 3 Hi Byte = 0
- V-Memory 3 Lo Byte = Other CRC Byte

MRX Slave Address Ranges

Function Code	MODBUS Data Format	Slave Address Range(s)
01 – Read Coil	484 Mode	1–999
01 – Read Coil	584/984 Mode	1–65535
02 – Read Input Status	484 Mode	1001–1999
02 – Read Input Status	584/984 Mode	10001–19999 (5 digit) or 100001–165535 (6 digit)
03 – Read Holding Register	484 Mode	4001–4999
03 – Read Holding Register	584/984 Mode	40001–49999 (5 digit) or 4000001–465535 (6 digit)
04 – Read Input Register	484 Mode	3001–3999
04 – Read Input Register	584/984 Mode	30001–39999 (5 digit) or 3000001–365535 (6 digit)
07 – Read Exception Status	484 and 584/984 Mode	N/A

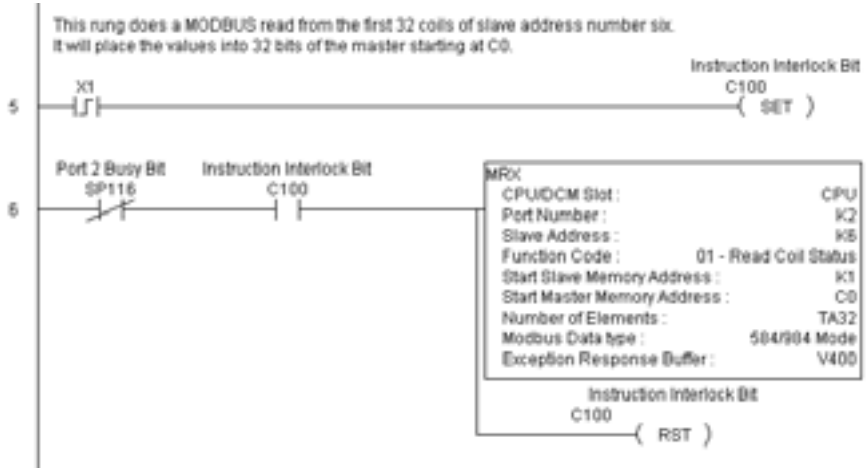
MRX Master Memory Address Ranges		
Operand Data Type		DL06 Range
Inputs	X	0–1777
Outputs	Y	0–1777
Control Relays	C	0–3777
Stage Bits	S	0–1777
Timer Bits	T	0–377
Counter Bits	CT	0–377
Special Relays	SP	0–777
V–memory	V	all
Global Inputs	GX	0–3777
Global Outputs	GY	0–3777

Number of Elements		
Operand Data Type		DL06 Range
V–memory	V	all
Constant	K	Bits: 1–2000 Registers: 1–125

Exception Response Buffer		
Operand Data Type		DL06 Range
V–memory	V	all

MRX Example

DL06 port 2 has two Special Relay contacts associated with it (see Appendix D for comm port special relays). One indicates "Port busy"(SP116), and the other indicates "Port Communication Error"(SP117). The "Port Busy" bit is on while the PLC communicates with the slave. When the bit is off, the program can initiate the next network request. The "Port Communication Error" bit turns on when the PLC has detected an error. Use of this bit is optional. When used, it should be ahead of any network instruction boxes, since the error bit is reset when an MRX or MWX instruction is executed. Typically, network communications will last longer than 1 CPU scan. The program must wait for the communications to finish before starting the next transaction.



***NOTE**: See Chapter 4, page 4-21, for an RLL example using multiple Read and Write interlocks with MRX/MWX instructions.*

MODBUS Write to Network (MWX)

DS	Used
HPP	N/A

The MODBUS Write to Network (MWX) instruction is used to write a block of data from the network masters's (DL06) memory to MODBUS memory addresses within a slave device on the network. The instruction allows the user to specify the MODBUS Function Code, slave station address, starting master and slave memory addresses, number of elements to transfer, MODBUS data format and the Exception Response Buffer.

CPU/DCM: select either CPU or DCM module for communications

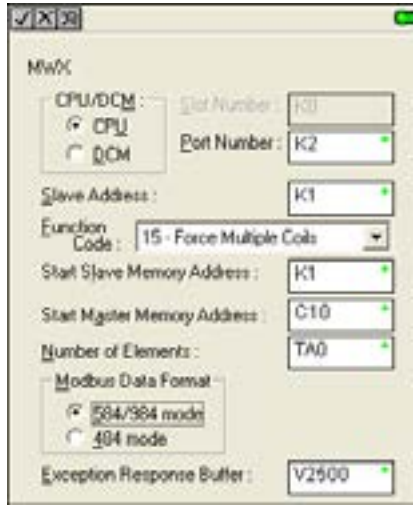
Slot Number: select PLC option slot number if using a DCM module

Port Number: must be DL06 Port 2 (K2)

Slave Address: specify a slave station address (0–247)

Function Code: MODBUS function codes supported by the MWX instruction:

- 05 – Force Single coil
- 06 – Preset Single Register
- 15 – Force Multiple Coils
- 16 – Preset Multiple Registers



Start Slave Memory Address: specifies the starting slave memory address where the data will be written

Start Master Memory Address: specifies the starting address of the data in the master that is to be written to the slave

Number of Elements: specifies how many consecutive coils or registers will be written to. This field is only active when either function code 15 or 16 is selected.

MODBUS Data Format: specifies MODBUS 584/984 or 484 data format to be used

Exception Response Buffer: specifies the master memory address where the Exception Response will be placed (6-bytes in length). See the table on the following page.The exception response buffer uses 3 words. These bytes are swapped in the MRX/MWX exception response buffer V-memory so:

- V-Memory 1 Hi Byte = Function Code Byte (Most Significant Bit Set)
- V-Memory 1 Lo Byte = Address Byte
- V-Memory 2 Hi Byte = One of the CRC Bytes
- V-Memory 2 Lo Byte = Exception Code
- V-Memory 3 Hi Byte = 0
- V-Memory 3 Lo Byte = Other CRC Byte

MWX Slave Address Ranges

MWX Slave Address Ranges		
Function Code	**MODBUS Data Format**	**Slave Address Range(s)**
05 – Force Single Coil	484 Mode	1–999
05 – Force Single Coil	584/984 Mode	1–65535
06 – Preset Single Register	484 Mode	4001–4999
06 – Preset Single Register	584/984 Mode	40001–49999 (5 digit) or 400001–465535 (6 digit)
15 – Force Multiple Coils	484 Mode	1–999
15 – Force Multiple Coils	585/984 Mode	1–65535
16 – Preset Multiple Registers	484 Mode	4001–4999
16 – Preset Multiple Registers	584/984 Mode	40001–49999 (5 digit) or 4000001–465535 (6 digit)

MWX Master Memory Address Ranges

MWX Master Memory Address Ranges		
Operand Data Type		**DL06 Range**
Inputs	X	0–1777
Outputs	Y	0–1777
Control Relays	C	0–3777
Stage Bits	S	0–1777
Timer Bits	T	0–377
Counter Bits	CT	0–377
Special Relays	SP	0–777
V–memory	V	all
Global Inputs	GX	0–3777
Global Outputs	GY	0–3777

MWX Number of Elements

Number of Elements		
Operand Data Type		**DL06 Range**
V–memory	V	all
Constant	K	Bits: 1–2000 Registers: 1–125

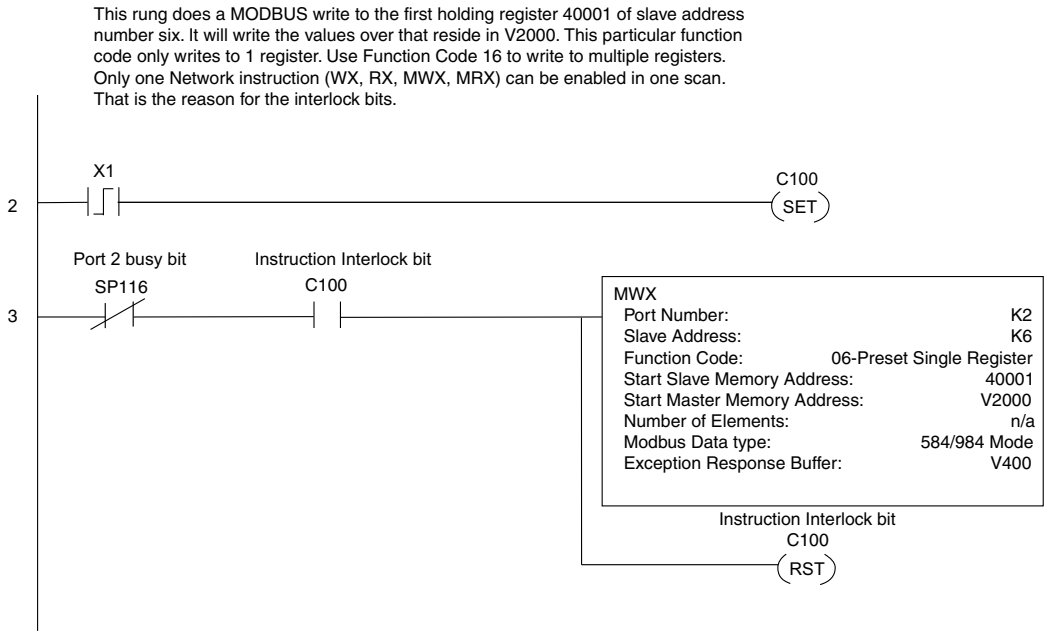
MWX Exception Response Buffer

Number of Elements		
Operand Data Type		**DL06 Range**
V–memory	V	all

MWX Example

DL06 port 2 has two Special Relay contacts associated with it (see Appendix D for comm port special relays). One indicates “Port busy”(SP116), and the other indicates ”Port Communication Error”(SP117). The “Port Busy” bit is on while the PLC communicates with the slave. When the bit is off, the program can initiate the next network request. The “Port Communication Error” bit turns on when the PLC has detected an error. Use of this bit is optional. When used, it should be ahead of any network instruction boxes since the error bit is reset when an MRX or MWX instruction is executed.

Typically, network communications will last longer than 1 CPU scan. The program must wait for the communications to finish before starting the next transaction.



***NOTE**: See Chapter 4, page 4-21, for an RLL example using multiple Read and Write interlocks with MRX/MWX instructions.*

ASCII Instructions

The DL06 CPU supports several instructions and methods that allow ASCII strings to be read into and written from the PLC communications ports. Specifically, port 2 on the DL06 can be used for either reading or writing raw ASCII strings, but cannot be used for both at the same time. The DL06 can also decipher ASCII embedded within a supported protocol (K–Sequence, *Direct*Net, Modbus) via the CPU port.

Reading ASCII Input Strings

There are several methods that the DL06 can use to read ASCII input strings.

1) ASCII IN (AIN) – This instruction configures port 2 for raw ASCII input strings with parameters such as fixed and variable length ASCII strings, termination characters, byte swapping options, and instruction control bits. Use barcode scanners, weight scales, etc. to write raw ASCII input strings into port 2 based on the (AIN) instruction's parameters.
2) Write embedded ASCII strings directly to V–memory from an external HMI or similar master device via a supported communications protocol using the CPU ports. The AIN instruction is not used in this case. 3) If a DL06 PLC is a master on a network, the Network Read instruction (RX) can be used to read embedded ASCII data from a slave device via a supported communications protocol using port 2. The RX instruction places the data directly into V–memory.

Writing ASCII Output Strings

The following instructions can be used to write ASCII output strings:

1) Print from V–memory (PRINTV) – Use this instruction to write raw ASCII strings out of port 2 to a display panel or a serial printer, etc. The instruction features the starting V–memory address, string length, byte swapping options, etc. When the instruction's permissive bit is enabled, the string is written to port 2.
2) Print to V–memory (VPRINT) – Use this instruction to create pre–coded ASCII strings in the PLC (i.e. alarm messages). When the instruction's permissive bit is enabled, the message is loaded into a pre–defined V–memory address location. Then the (PRINTV) instruction may be used to write the pre–coded ASCII string out of port 2. American, European and Asian Time/Date stamps are supported.

Additionally, if a DL06 PLC is a master on a network, the Network Write instruction (WX) can be used to write embedded ASCII data to an HMI or slave device directly from V–memory via a supported communications protocol using port 2.

Managing the ASCII Strings

The following instructions can be helpful in managing the ASCII strings within the CPUs V–memory:

ASCII Find (AFIND) – Finds where a specific portion of the ASCII string is located in continuous V–memory addresses. Forward and reverse searches are supported.

ASCII Extract (AEX) – Extracts a specific portion (usually some data value) from the ASCII find location or other known ASCII data location.

Compare V–memory (CMPV) – This instruction is used to compare two blocks of V–memory addresses and is usually used to detect a change in an ASCII string. Compared data types must be of the same format (i.e., BCD, ASCII, etc.).

Swap Bytes (SWAPB) – usually used to swap V–memory bytes on ASCII data that was written directly to V–memory from an external HMI or similar master device via a communications protocol. The AIN and AEX instructions have a built–in byte swap feature.

ASCII Input (AIN)

DS	Used
HPP	N/A

The ASCII Input instruction allows the CPU to receive ASCII strings through the specified communications port and places the string into a series of specified V–memory registers. The ASCII data can be received as a fixed number of bytes or as a variable length string with specified termination character(s). Other features include, Byte Swap preferences, Character Timeout, and user defined flag bits for Busy, Complete and Timeout Error.

AIN Fixed Length Configuration

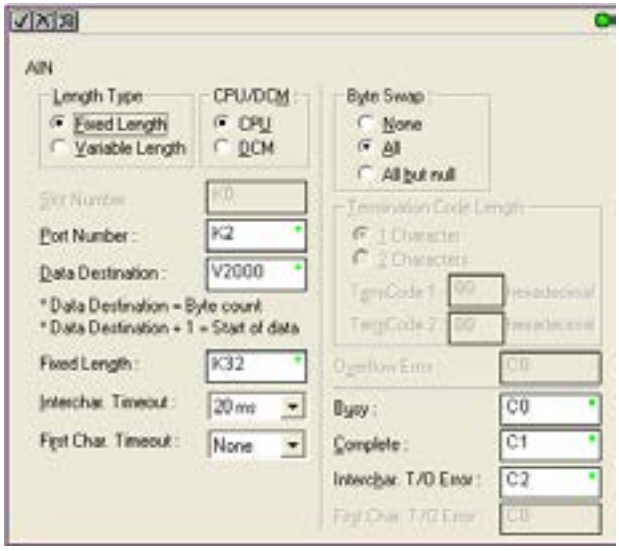
Length Type: select fixed length based on the length of the ASCII string that will be sent to the CPU port

Port Number: must be DL06 port 2 (K2)

Data Destination: specifies where the ASCII string will be placed in V–memory

Fixed Length: specifies the length, in bytes, of the fixed length ASCII string the port will receive

Inter–character Timeout: if the amount of time between incoming ASCII characters exceeds the set time, the specified Timeout Error bit will be set.



No data will be stored at the Data Destination V–memory location. The bit will reset when the AIN instruction permissive bits are disabled. 0ms selection disables this feature.

First Character Timeout: if the amount of time from when the AIN is enabled to the time the first character is received exceeds the set time, the specified First Character Timeout bit will be set. The bit will reset when the AIN instruction permissive bits are disabled. 0ms selection disables this feature.

Byte Swap: swaps the high–byte and low–byte within each V–memory register of the Fixed Length ASCII string. See the SWAPB instruction for details.

Busy Bit: is ON while the AIN instruction is receiving ASCII data

Complete Bit: is set once the ASCII data has been received for the specified fixed length and reset when the AIN instruction permissive bits are disabled.

Inter–character Timeout Error Bit: is set when the Character Timeout is exceeded. See Character Timeout explanation above.

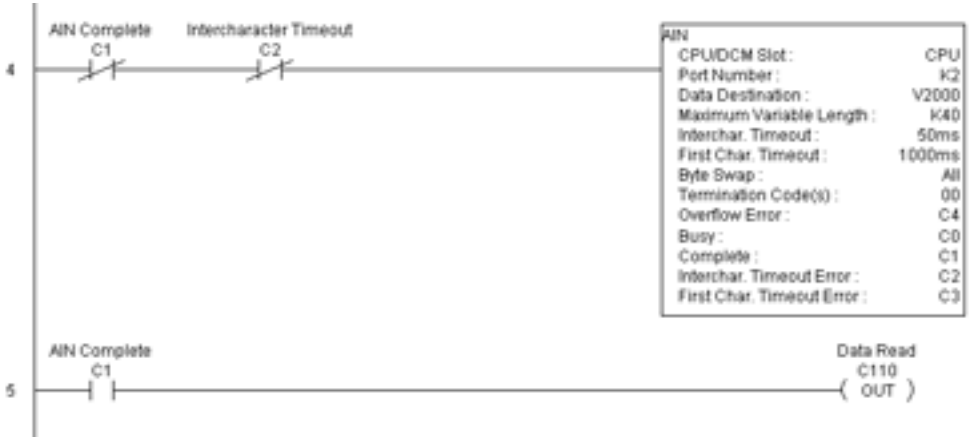
First Character Timeout Error Bit: is set when the First Character Timeout is exceeded. See First Character Timeout explanation above.

Parameter	
Data Destination	All V–memory
Fixed Length	K1–128
Bits: Busy, Complete, Timeout Error, Overflow	C0–3777

AIN Fixed Length Examples

Fixed Length example when the PLC is reading the port continuously and timing is not critical

Fixed Length example when character to character timing is critical



AIN Variable Length Configuration:

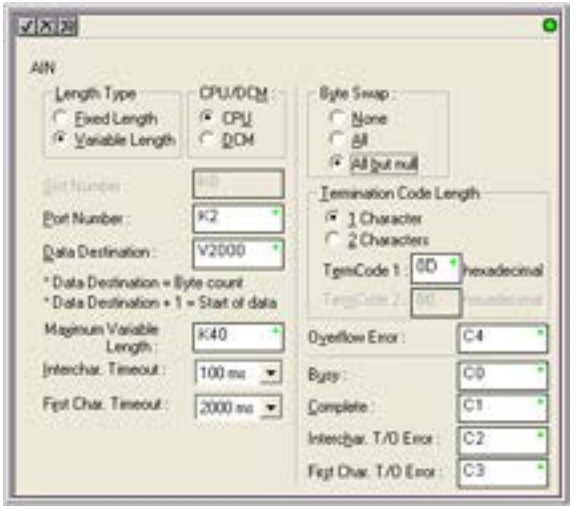
Length Type: select Variable Length if the ASCII string length followed by termination characters will vary in length

Port Number: must be DL06 port 2 (K2)

Data Destination: specifies where the ASCII string will be placed in V–memory

Maximum Variable Length: specifies, in bytes, the maximum length of a Variable Length ASCII string the port will receive

Inter–character Timeout: if the amount of time between incoming ASCII characters exceeds the set time, the Timeout Error bit will be set. No data will be stored at the Data Destination V–memory location.



The Timeout Error bit will reset when the AIN instruction permissive bits are disabled. 0ms selection disables this feature.

First Character Timeout: if the amount of time from when the AIN is enabled to the time the first character is received exceeds the set time, the specified First Character Timeout bit will be set. The bit will reset when the AIN instruction permissive bits are disabled. 0ms selection disables this feature.

Byte Swap: swaps the high–byte and low–byte within each V–memory register of the Variable Length ASCII string. See the SWAPB instruction for details.

Termination Code Length: consists of either 1 or 2 characters. Refer to Appendix G, ASCII Table.

Busy Bit: is ON while the AIN instruction is receiving ASCII data

Complete Bit: is set once the ASCII data has been received up to the termination code characters. It will be reset when the AIN instruction permissive bits are disabled.

Inter–character Timeout Error Bit: is set when the Character Timeout is exceeded. See Character Timeout explanation above.

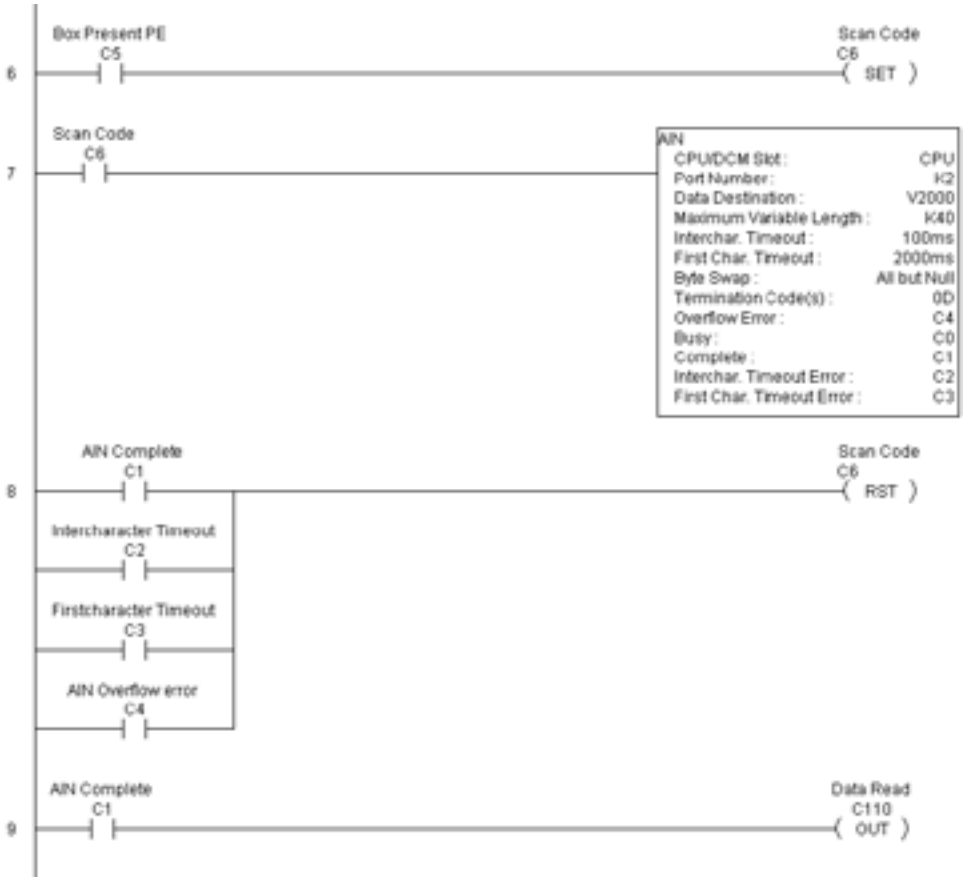
First Character Timeout Error Bit: is set when the First Character Timeout is exceeded. See First Character Timeout explanation above.

Overflow Error Bit: is set when the ASCII data received exceeds the Maximum Variable Length specified.

Parameter	
Data Destination	All V–memory
Fixed Length	K1–128
Bits: Busy, Complete, Timeout Error, Overflow	C0–3777

AIN Variable Length Example

AIN variable length example used to read barcodes on boxes (PE = photoelectric sensor)



ASCII Find (AFIND)

DS	Used
HPP	N/A

The ASCII Find instruction locates a specific ASCII string or portion of an ASCII string within a range of V–memory registers and places the string's Found Index number (byte number where desired string is found), in Hex, into a specified V–memory register. Other features include, Search Starting Index number for skipping over unnecessary bytes before beginning the FIND operation, Forward or Reverse direction search, and From Beginning and From End selections to reference the Found Index Value.

Base Address: specifies the beginning V–memory register where the entire ASCII string is stored in memory

Total Number of Bytes: specifies the total number of bytes to search for the desired ASCII string

Search Starting Index: specifies which byte to skip to (with respect to the Base Address) before beginning the search

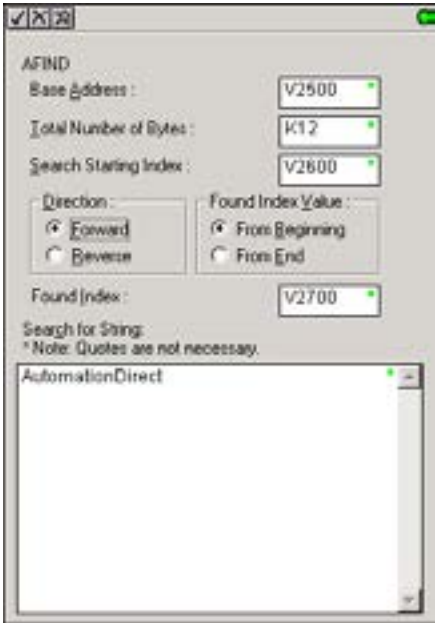
Direction: Forward begins the search from lower numbered V–memory registers to higher numbered V–memory registers. Reverse does the search from higher numbered V–memory registers to lower numbered V–memory registers.

Found Index Value: specifies whether the Beginning or the End byte of the ASCII string found will be loaded into the Found Index register

Found Index: specifies the V–memory register where the Found Index Value will be stored. A value of FFFF will result if the desired string is not located in the memory registers specified.

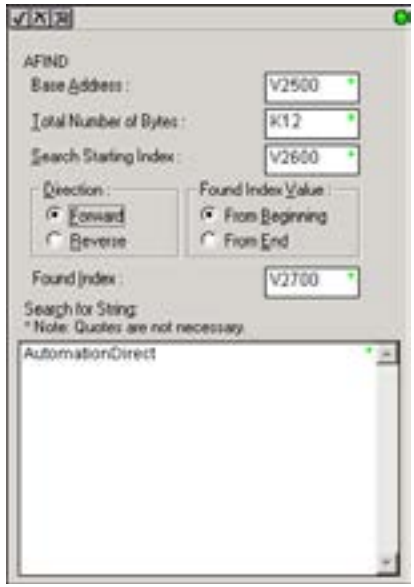
Search for String: up to 128 characters.

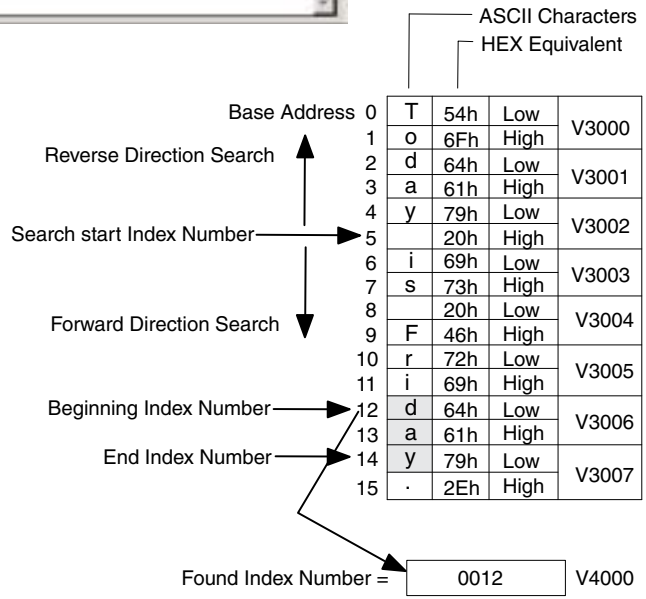
Parameter	DL06 Range
Base Address	All V–memory
Total Number of Bytes	All V–memory or K1–128
Search Starting Index	All V–memory or K0–127
Found Index	All V–memory



AFIND Search Example

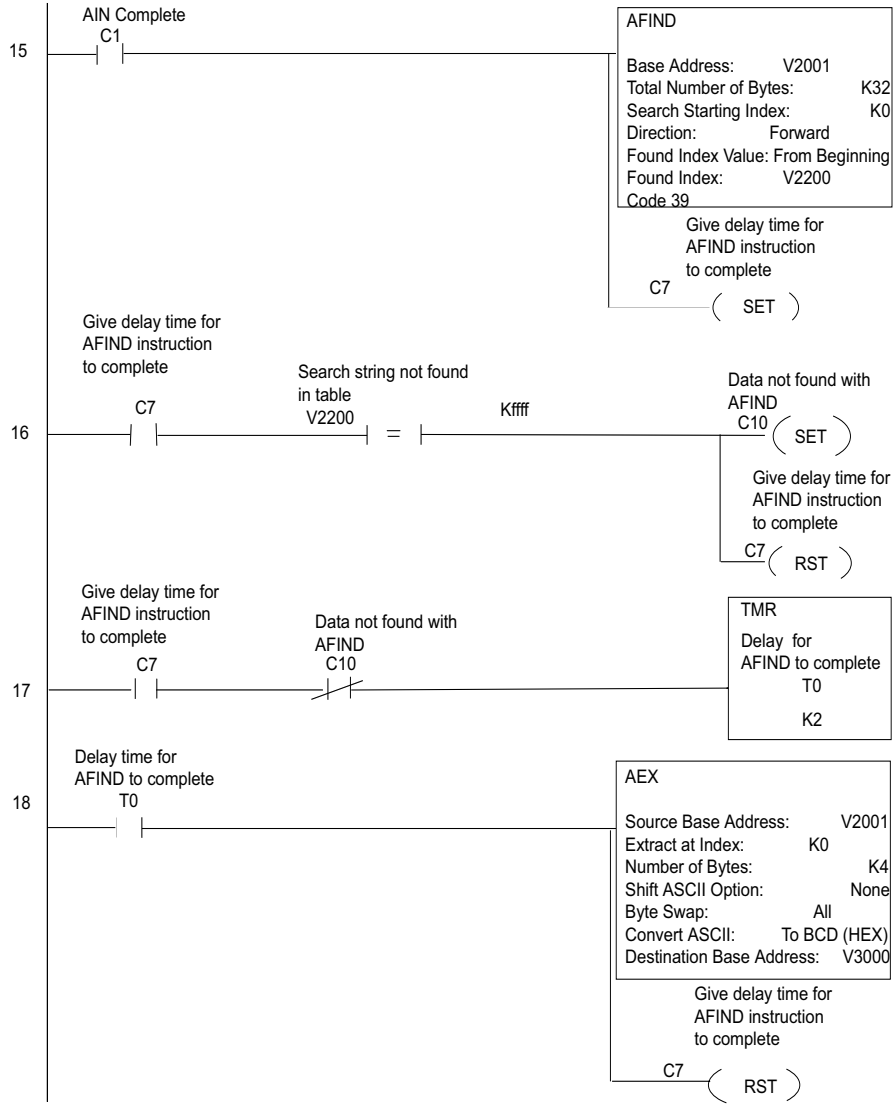
In the following example, the AFIND instruction is used to search for the "day" portion of "Friday" in the ASCII string "Today is Friday.", which had previously been loaded into V–memory. Note that a Search Starting Index of constant (K) 5 combined with a Forward Direction Search is used to prevent finding the "day" portion of the word "Today". The Found Index will be placed into V4000.





AFIND Example Combined with AEX Instruction

When an AIN instruction has executed, its Complete bit can be used to trigger an AFIND instruction to search for a desired portion of the ASCII string. Once the string is found, the AEX instruction can be used to extract the located string.



ASCII Extract (AEX)

DS	Used
HPP	N/A

The ASCII Extract instruction extracts a specified number of bytes of ASCII data from one series of V–memory registers and places it into another series of V–memory registers. Other features include, Extract at Index for skipping over unnecessary bytes before beginning the Extract operation, Shift ASCII Option, for One Byte Left or One Byte Right, Byte Swap and Convert data to a BCD format number.

Source Base Address: specifies the beginning V–memory register where the entire ASCII string is stored in memory

Extract at Index: specifies which byte to skip to (with respect to the Source Base Address) before extracting the data

Number of Bytes: specifies the number of bytes to be extracted

Shift ASCII Option: shifts all extracted data one byte left or one byte right to displace "unwanted" characters if necessary

Byte Swap: swaps the high–byte and the low–byte within each V–memory register of the extracted data. See the SWAPB instruction for details.

Convert BCD(Hex) ASCII to BCD (Hex): if enabled, this will convert ASCII numerical characters to Hexadecimal numerical values

Destination Base Address: specifies the V–memory register where the extracted data will be stored

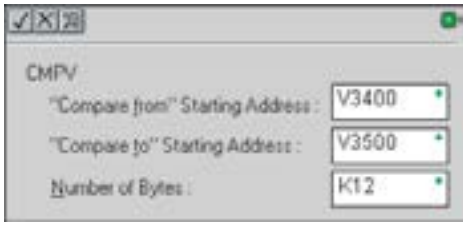
See the previous page for an example using the AEX instruction.

Parameter	DL06 Range	
Source Base Address	All V–memory	
Extract at Index	All V–memory or K0–127	
Number of Bytes "Convert BCD (HEX) ASCII" not checked	Constant range: K1–128	V-memory location containing BCD value: 1–128
Number of Bytes "Convert BCD (HEX) ASCII" checked	Constant range: K1–4	V-memory location containing BCD value: 1–4
Destination Base Address	All V–memory	

ASCII Compare (CMPV)

DS	Used
HPP	N/A

The ASCII Compare instruction compares two groups of V–memory registers. The CMPV will compare any data type (ASCII to ASCII, BCD to BCD, etc.) of one series (group) of V–memory registers to another series of V–memory registers for a specified byte length.



SP61 = 1, the result is equal
SP61 = 0, the result is not equal

"Compare from" Starting Address: specifies the beginning V–memory register of the first group of V–memory registers to be compared from.

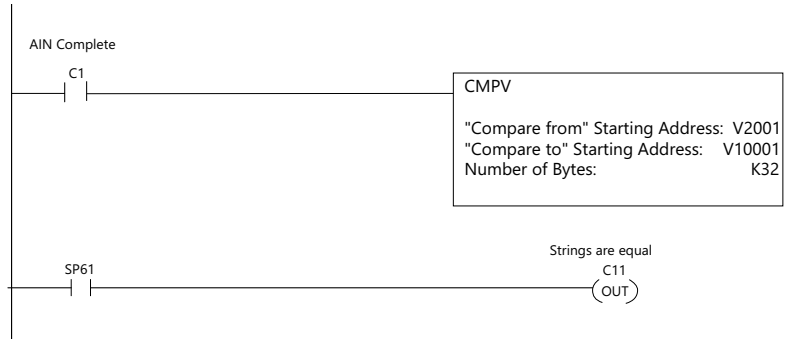
"Compare to" Starting Address: specifies the beginning V–memory register of the second group of V–memory registers to be compared to.

Number of Bytes: specifies the length of each V–memory group to be compared

Parameter	DL06 Range
Compare from Starting Address	All V–memory
Compare to Starting Address	All V–memory
Number of Bytes	K0–127

CMPV Example

The CMPV instruction executes when the AIN instruction is complete. If the compared V–memory tables are equal, SP61 will turn ON.



ASCII Print to V–memory (VPRINT)

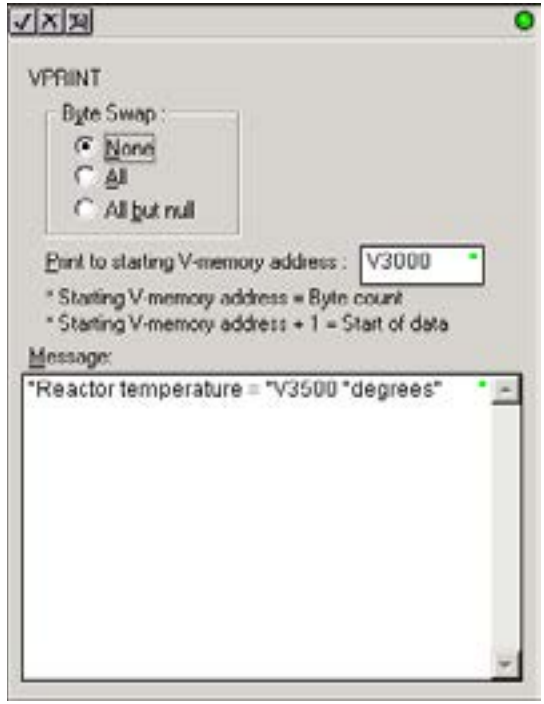
DS	Used
HPP	N/A

The ASCII Print to V–memory instruction will write a specified ASCII string into a series of V–memory registers. Other features include Byte Swap, options to suppress or convert leading zeros or spaces, and _Date and _Time options for U.S., European, and Asian date formats and 12 or 24 hour time formats.

Byte Swap: swaps the high–byte and low–byte within each V–memory register the ASCII string is printed to. See the SWAPB instruction for details.

Print to Starting V–memory Address: specifies the beginning of a series of V–memory addresses where the ASCII string will be placed by the VPRINT instruction.

Starting V–memory Address: the first V–memory register of the series of registers specified will contain the ASCII string's length in bytes.



Starting V–memory Address +1: the 2nd and subsequent registers will contain the ASCII string printed to V–memory.

Parameter	DL06 Range
Print to Starting V–memory Address	All V–memory

VPRINT Time/Date Stamping– the codes in the table below can be used in the VPRINT ASCII string message to "print to V–memory" the current time and/or date.

#	Character code	Date / Time Stamp Options
1	_date:us	American standard (month/day/2 digit year)
2	_date:e	European standard (day/month/2 digit year)
3	_date:a	Asian standard (2 digit year/month/day)
4	_time:12	Standard 12 hour clock (0–12 hour:min am/pm)
5	_time:24	Standard 24 hour clock (0–23 hour:min am/pm)

VPRINT V-memory element – the following modifiers can be used in the VPRINT ASCII string message to "print to V–memory" register contents in integer format or real format. Use V-memory number or V-memory number with ":" and data type. The data types are shown in the table below. The Character code must be capital letters.

***NOTE**: There must be a space entered before and after the V-memory address to separate it from the text string. Failure to do this will result in an error code 499.*

#	Character code	Description
1	none	16-bit binary (decimal number)
2	: B	4 digit BCD
3	: D	32-bit binary (decimal number)
4	: D B	8 digit BCD
5	: R	Floating point number (real number)
6	: E	Floating point number (real number with exponent)

Examples:

V2000 Print binary data in V2000 for decimal number

V2000 : B Print BCD data in V2000

V2000 : D Print binary number in V2000 and V2001 for decimal number

V2000 : D B Print BCD data in V2000 and V2001

V2000 : R Print floating point number in V2000/V2001 as real number

V2000 : E Print floating point number in V2000/V2001 as real number with exponent

The following modifiers can be added to any of the modifies above to suppress or convert leading zeros or spaces. The character code must be capital letters.

#	Character code	Description
1	S	Suppresses leading spaces
2	C0	Converts leading spaces to zeros
3	0	Suppresses leading zeros

Example with V2000 = 0018 (binary format)

V–memory Register with Modifier	Number of Characters			
	1	2	3	4
V2000	0	0	1	8
V2000:B	0	0	1	2
V2000:B0	1	2		

Example with V2000 = sp sp18 (binary format) where sp = space

V–memory Register with Modifier	Number of Characters			
	1	2	3	4
V2000	sp	sp	1	8
V2000:B	sp	sp	1	2
V2000:BS	1	2		
V2000:BC0	0	0	1	2

VPRINT V-memory text element – the following is used for "printing to V–memory" text stored in registers. Use the % followed by the number of characters after V-memory number for representing the text. If you assign "0" as the number of characters, the function will read the character count from the first location. Then it will start at the next V-memory location and read that number of ASCII codes for the text from memory.

Example:

V2000 % 16 16 characters in V2000 to V2007 are printed.

V2000 % 0 The characters in V2001 to Vxxxx (determined by the number in V2000) will be printed.

VPRINT Bit element – the following is used for "printing to V–memory" the state of the designated bit in V-memory or a control relay bit. The bit element can be assigned by the designating point (.) and bit number preceded by the V-memory number or relay number. The output type is described as shown in the table below.

#	Data format	Description
1	none	Print 1 for an ON state, and 0 for an OFF state
2	: BOOL	Print "TRUE" for an ON state, and "FALSE" for an OFF state
3	: ONOFF	Print "ON" for an ON state, and "OFF" for an OFF state

Example:

V2000 . 15 Prints the status of bit 15 in V2000, in 1/0 format

C100 Prints the status of C100 in 1/0 format

C100 : BOOL Prints the status of C100 in TRUE/FALSE format

C100 : ON/OFF Prints the status of C100 in ON/OFF format

V2000.15 : BOOL Prints the status of bit 15 in V2000 in TRUE/FALSE format

The maximum numbers of characters you can VPRINT is 128. The number of characters required for each element, regardless of whether the :S, :C0 or :0 modifiers are used, is listed in the table below.

Element type	Maximum Characters
Text, 1 character	1
16 bit binary	6
32 bit binary	11
4 digit BCD	4
8 digit BCD	8
Floating point (real number)	3
Floating point (real with exponent)	13
V-memory/text	2
Bit (1/0 format)	1
Bit (TRUE/FALSE format)	5
Bit (ON/OFF format)	3

Text element – the following is used for "printing to V–memory" character strings. The character strings are defined as the character (more than 0) ranged by the double quotation marks. Two hex numbers preceded by the dollar sign means an 8-bit ASCII character code. Also, two characters preceded by the dollar sign is interpreted according to the following table:

#	Character code	Description
1	$$	Dollar sign ($)
2	$"	Double quotation (")
3	$Lor $l	Line feed (LF)
4	$N or $n	Carriage return line feed (CRLF)
5	$P or $p	Form feed
6	$R or $r	Carriage return (CR)
7	$T or $t	Tab

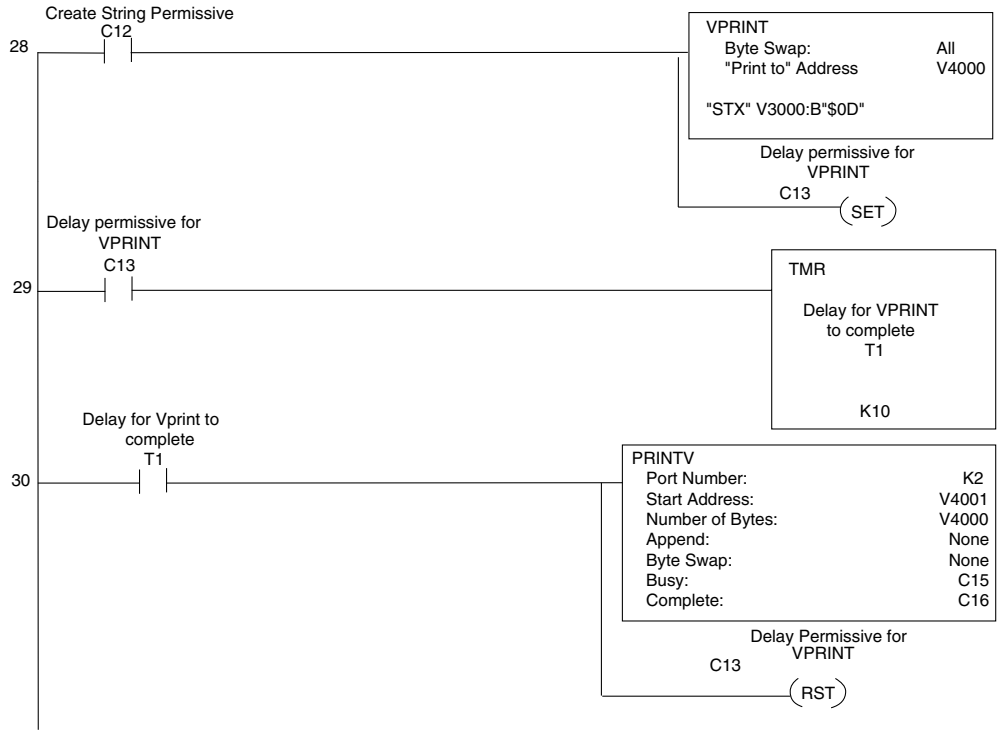
The following examples show various syntax conventions and the length of the output to the printer.

" "	Length 0 without character
"A"	Length 1 with character A
" "	Length 1 with blank
"$" "	Length 1 with double quotation mark
"RL"	Length 2 with one CR and one LF
"$0D$0A"	Length 2 with one CR and one LF
"$$"	Length 1 with one $ mark

In printing an ordinary line of text, you will need to include double quotation marks before and after the text string. Error code 499 will occur in the CPU when the print instruction contains invalid text or no quotations. It is important to test your VPRINT instruction data during the application development.

VPRINT Example Combined with PRINTV Instruction

The VPRINT instruction is used to create a string in V–memory. The PRINTV is used to print the string out of port 2.



ASCII Print from V–memory (PRINTV)

DS	Used
HPP	N/A

The ASCII Print from V–memory instruction will send an ASCII string out of the designated communications port from a specified series of V–memory registers for a specified length in number of bytes. Other features include user specified Append Characters to be placed after the desired data string for devices that require specific termination character(s), Byte Swap options, and user specified flags for Busy and Complete.

Port Number: must be DL06 port 2 (K2)

Start Address: specifies the beginning of series of V–memory registers that contain the ASCII string to print

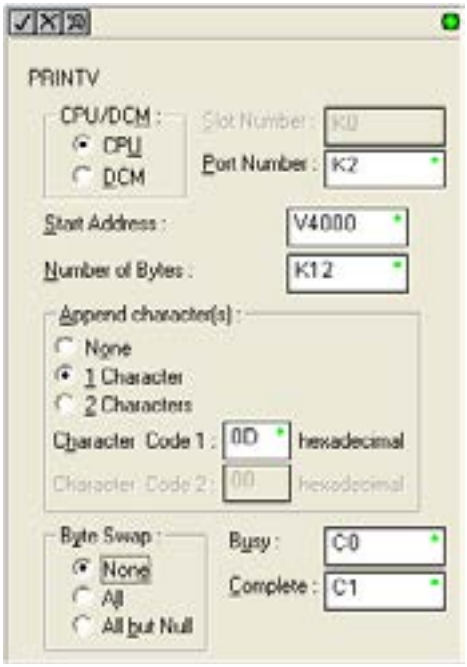
Number of Bytes: specifies the length of the string to print

Append Characters: specifies ASCII characters to be added to the end of the string for devices that require specific termination characters

Byte Swap: swaps the high–byte and low–byte within each V–memory register of the string while printing. See the SWAPB instruction for details.

Busy Bit: will be ON while the instruction is printing ASCII data

Complete Bit: will be set once the ASCII data has been printed and reset when the PRINTV instruction permissive bits are disabled.



See the previous page for an example using the PRINTV instruction.

Parameter	DL06 Range
Port Number	Port 2 (K2)
Start Address	All V–memory
Number of Bytes	All V–memory or k1–128
Bits: Busy, Complete	C0–3777

ASCII Swap Bytes (SWAPB)

DS	Used
HPP	N/A

The ASCII Swap Bytes instruction swaps byte positions (high–byte to low–byte and low–byte to high–byte) within each V–memory register of a series of V–memory registers for a specified number of bytes.

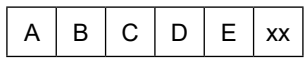
- Starting Address: specifies the beginning of a series of V–memory registers the instruction will use to begin byte swapping
- Number of Bytes: specifies the number of bytes, beginning with the Starting Address, to byte swap.
- Byte Swap: All - swap all bytes specified. All but null - swap all bytes specified except the bytes with a null

Parameter	DL06 Range
Starting Address	All V–memory
Number of Bytes	All V–memory or K1–128

Discrete Bit Flags	Description
SP53	On if the CPU cannot execute the instruction.
SP71	On when a value used by the instruction is invalid.

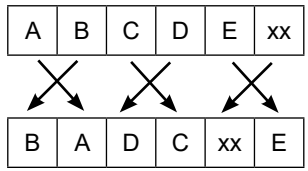
Byte Swap Preferences

No Byte Swapping
(AIN, AEX, PRINTV, VPRINT)



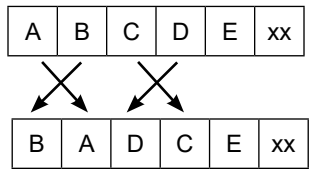
	Byte High	Byte Low
V2477	0005h	
V2500	B	A
V2501	D	C
V2502	xx	E

Byte Swap All



	Byte High	Byte Low
V2477	0005h	
V2500	A	B
V2501	C	D
V2502	E	xx

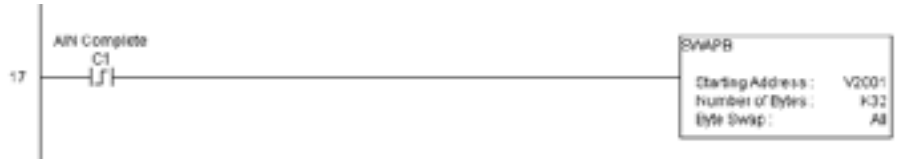
Byte Swap All but Null



	Byte High	Byte Low
V2477	0005h	
V2500	B	A
V2501	D	C
V2502	xx	E

SWAPB Example

The AIN Complete bit is used to trigger the SWAPB instruction. Use a one–shot so the SWAPB only executes once.



ASCII Clear Buffer (ACRB)

DS	Used
HPP	N/A

The ASCII Clear Buffer instruction will clear the ASCII receive buffer of the specified communications port number. Port Number: must be DL06 port 2 (K2)

ACRB Example

The AIN Complete bit or the AIN diagnostic bits are used to clear the ASCII buffer.

This page intentionally left blank.

Intelligent Box (IBox) Instructions

The Intelligent Box Instructions (IBox) listed in this section are additional instructions made available when using *Direct*SOFT to program your DL06 PLC (the DL06 CPU requires firmware version v2.10 or later to use the new features in *Direct*SOFT). For more information on *Direct*SOFT and to download a free demo version, please visit our Web site at: www.automationdirect.com.

Communication IBoxes		
Instruction	**Ibox #**	**Page**
ECOM100 Configuration (ECOM100)	IB-710	5-272
ECOM100 Disable DHCP (ECDHCPD)	IB-736	5-274
ECOM100 Enable DHCP (ECDHCPE)	IB-735	5-276
ECOM100 Query DHCP Setting (ECDHCPQ)	IB-734	5-278
ECOM100 Send E-mail (ECEMAIL)	IB-711	5-280
ECOM100 Restore Default E-mail Setup (ECEMRDS)	IB-713	5-283
ECOM100 E-mail Setup (ECEMSUP)	IB-712	5-286
ECOM100 IP Setup (ECIPSUP)	IB-717	5-290
ECOM100 Read Description (ECRDDES)	IB-726	5-292
ECOM100 Read Gateway Address (ECRDGWA)	IB-730	5-294
ECOM100 Read IP Address (ECRDIP)	IB-722	5-296
ECOM100 Read Module ID (ECRDMID)	IB-720	5-298
ECOM100 Read Module Name (ECRDNAM)	IB-724	5-300
ECOM100 Read Subnet Mask (ECRDSNM)	IB-732	5-302
ECOM100 Write Description (ECWRDES)	IB-727	5-304
ECOM100 Write Gateway Address (ECWRGWA)	IB-731	5-306
ECOM100 Write IP Address (ECWRIP)	IB-723	5-308
ECOM100 Write Module ID (ECWRMID)	IB-721	5-310
ECOM100 Write Name (ECWRNAM)	IB-725	5-312
ECOM100 Write Subnet Mask (ECWRSNM)	IB-733	5-314
ECOM100 RX Network Read (ECRX)	IB-740	5-316
ECOM100 WX Network Write(ECWX)	IB-741	5-319
NETCFG Network Configuration (NETCFG)	IB-700	5-322
Network RX Read (NETRX)	IB-701	5-324
Network WX Write (NETWX)	IB-702	5-327

Counter I/O IBoxes (Works with H0-CTRIO and H0-CTRIO2)		
Instruction	**Ibox #**	**Page**
CTRIO Configuration (CTRIO)	IB-1000	5-330
CTRIO Add Entry to End of Preset Table (CTRADPT)	IB-1005	5-332
CTRIO Clear Preset Table (CTRCLRT)	IB-1007	5-335
CTRIO Edit Preset Table Entry (CTREDPT)	IB-1003	5-338
CTRIO Edit Preset Table Entry and Reload (CTREDRL)	IB-1002	5-342
CTRIO Initialize Preset Table (CTRINPT)	IB-1004	5-346
CTRIO Initialize Preset Table (CTRINTR)	IB-1010	5-350
CTRIO Load Profile (CTRLDPR)	IB-1001	5-354
CTRIO Read Error (CTRRDER)	IB-1014	5-357
CTRIO Run to Limit Mode (CTRRTLM)	IB-1011	5-359
CTRIO Run to Position Mode (CTRRTPM)	IB-1012	5-362
CTRIO Velocity Mode (CTRVELO)	IB-1013	5-365
CTRIO Write File to ROM (CTRWFTR)	IB-1006	5-368

Analog Input/Output Combo Module Pointer Setup (ANLGCMB) (IB-462)

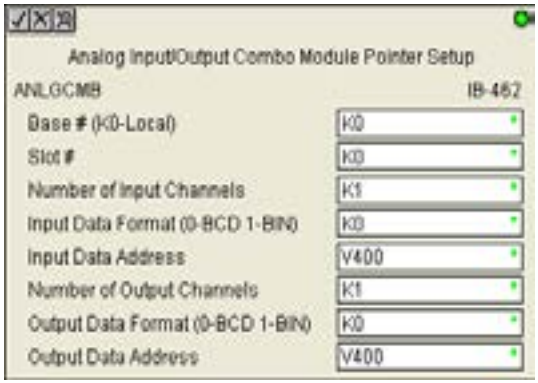
DS	Used
HPP	N/A

The Analog Input/Output Combo Module Pointer Setup instruction generates the logic to configure the pointer method for an analog input/output combination module on the first PLC scan following a Program to Run transition.

The ANLGCMB IBox instruction determines the data format and Pointer addresses based on the CPU type, the Base# and the module Slot#.

The Input Data Address is the starting location in user V-memory where the analog input data values will be stored, one location for each input channel enabled.

The Output Data Address is the starting location in user V-memory where the analog output data values will be placed by ladder code or external device, one location for each output channel enabled



Since the IBox logic only executes on the first scan, the instruction cannot have any input logic.

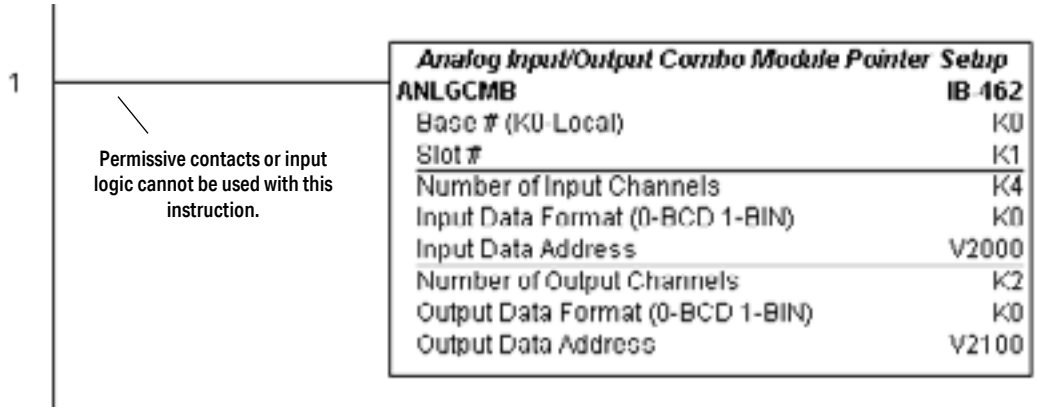
ANLGCMB Parameters

- Base # (K0-Local): must be 0 for DL06 PLC
- Slot #: specifies which PLC option slot is occupied by the analog module (1–4)
- Number of Input Channels: specifies the number of analog input channels to scan
- Input Data Format (0-BCD 1-BIN): specifies the analog input data format (BCD or Binary) - the binary format may be used for displaying data on some OI panels
- Input Data Address: specifies the starting V-memory location that will be used to store the analog input data
- Number of Output Channels: specifies the number of analog output channels that will be used
- Output Data Format (0-BCD 1-BIN): specifies the format of the analog output data (BCD or Binary)
- Output Data Address: specifies the starting V-memory location that will be used to source the analog output data

Parameter		DL06 Range
Base # (K0-Local)	K	K0 (local base only)
Slot #	K	K1-4
Number of Input Channels	K	K1-8
Input Data Format (0-BCD 1-BIN)	K	BCD: K0; Binary: K1
Input Data Address	V	See DL06 V-memory map - Data Words
Number of Output Channels	K	K1-8
Output Data Format (0-BCD 1-BIN)	K	BCD: K0; Binary: K1
Output Data Address	V	See DL06 V-memory map - Data Words

ANLGCMB Example

In the following example, the ANLGCMB instruction is used to setup the pointer method for an analog I/O combination module that is installed in option slot 2. Four input channels are enabled and the analog data will be written to V2000 - V2003 in BCD format. Two output channels are enabled and the analog values will be read from V2100 - V2101 in BCD format.



Analog Input Module Pointer Setup (ANLGIN) (IB-460)

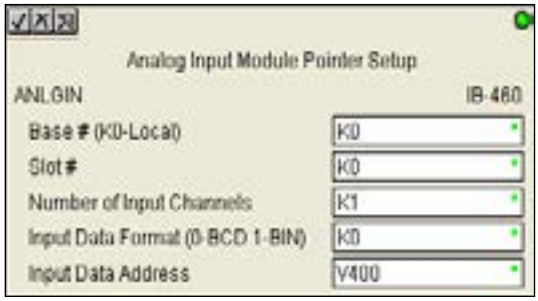
DS	Used
HPP	N/A

Analog Input Module Pointer Setup generates the logic to configure the pointer method for one analog input module on the first PLC scan following a Program to Run transition.

This IBox determines the data format and Pointer addresses based on the CPU type, the Base#, and the Slot#.

The Input Data Address is the starting location in user V-memory where the analog input data values will be stored, one location for each input channel enabled.

Since this logic only executes on the first scan, this IBox cannot have any input logic.



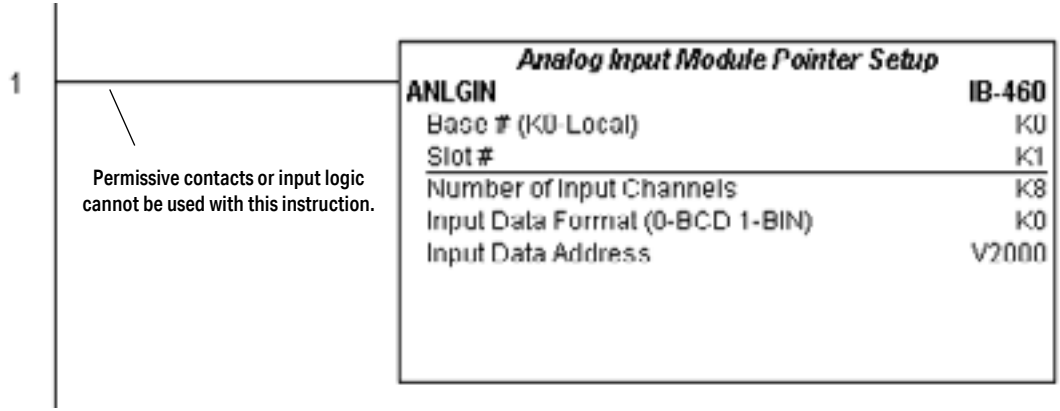
ANLGIN Parameters

- Base # (K0-Local): must be 0 for DL06 PLC
- Slot #: specifies which PLC option slot is occupied by the analog module (1–4)
- Number of Input Channels: specifies the number of input channels to scan
- Input Data Format (0-BCD 1-BIN): specifies the analog input data format (BCD or Binary) - the binary format may be used for displaying data on some OI panels
- Input Data Address: specifies the starting V-memory location that will be used to store the analog input data

Parameter		DL06 Range
Base # (K0-Local)	K	K0 (local base only)
Slot #	K	K1-4
Number of Input Channels	K	K1-8
Input Data Format (0-BCD 1-BIN)	K	BCD: K0; Binary: K1
Input Data Address	V	See DL06 V-memory map - Data Words

ANLGIN Example

In the following example, the ANLGIN instruction is used to setup the pointer method for an analog input module that is installed in option slot 1. Eight input channels are enabled and the analog data will be written to V2000 - V2007 in BCD format.



Analog Output Module Pointer Setup (ANLGOUT) (IB-461)

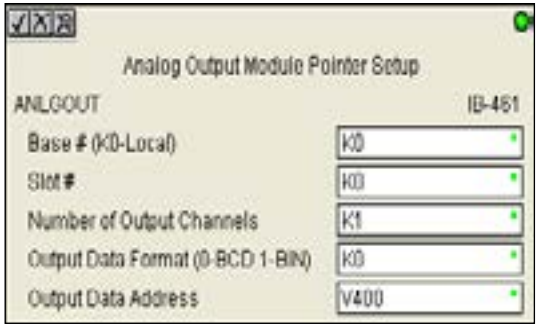
DS	Used
HPP	N/A

Analog Output Module Pointer Setup generates the logic to configure the pointer method for one analog output module on the first PLC scan following a Program to Run transition.

This IBox determines the data format and Pointer addresses based on the CPU type, the Base#, and the Slot#.

The Output Data Address is the starting location in user V-memory where the analog output data values will be placed by ladder code or external device, one location for each output channel enabled.

Since this logic only executes on the first scan, this IBox cannot have any input logic.



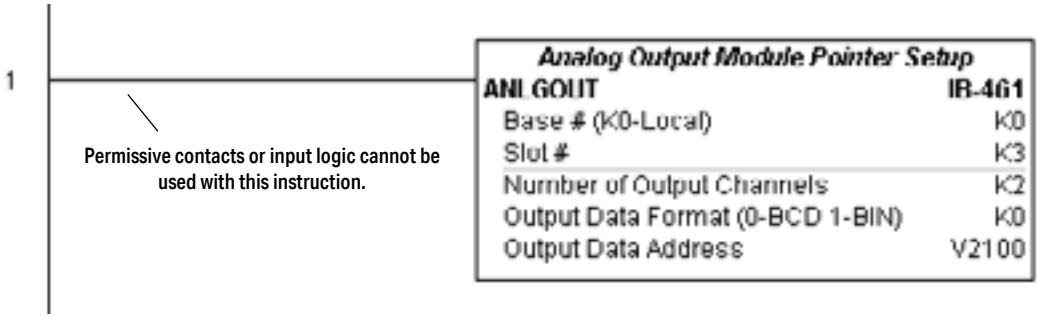
ANLGOUT Parameters

- Base # (K0-Local): must be 0 for DL06 PLC
- Slot #: specifies which PLC option slot is occupied by the analog module (1–4)
- Number of Output Channels: specifies the number of analog output channels that will be used
- Output Data Format (0-BCD 1-BIN): specifies the format of the analog output data (BCD or Binary)
- Output Data Address: specifies the starting V-memory location that will be used to source the analog output data

Parameter		DL06 Range
Base # (K0-Local)	K	K0 (local base only)
Slot #	K	K1-4
Number of Output Channels	K	K1-8
Output Data Format (0-BCD 1-BIN)	K	BCD: K0; Binary: K1
Output Data Address	V	See DL06 V-memory map - Data Words

ANLGOUT Example

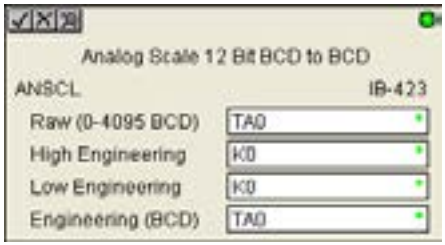
In the following example, the ANLGOUT instruction is used to setup the pointer method for an analog output module that is installed in option slot 3. Two output channels are enabled and the analog data will be read from V2100 - V2101 in BCD format.



Analog Scale 12 Bit BCD to BCD (ANSCL) (IB-423)

DS	Used
HPP	N/A

Analog Scale 12 Bit BCD to BCD scales a 12 bit BCD analog value (0-4095 BCD) into BCD engineering units. You specify the engineering unit high value (when raw is 4095), and the engineering low value (when raw is 0), and the output V-Memory address you want the to place the scaled engineering unit value. The engineering units are generated as BCD and can be the full range of 0 to 9999 (see ANSCLB - Analog Scale 12 Bit Binary to Binary if your raw units are in Binary format).



NOTE: *This IBox only works with unipolar unsigned raw values. It does NOT work with bipolar or sign plus magnitude raw values.*

ANSCL Parameters

- Raw (0-4095 BCD): specifies the V-memory location of the unipolar unsigned raw 0-4095 unscaled value
- High Engineering: specifies the high engineering value when the raw input is 4095
- Low Engineering: specifies the low engineering value when the raw input is 0
- Engineering (BCD): specifies the V-memory location where the scaled engineering BCD value will be placed

Parameter		DL06 Range
Raw (0-4095 BCD)	V,P	See DL06 V-memory map - Data Words
High Engineering	K	K0-9999
Low Engineering	K	K0-9999
Engineering (BCD)	V,P	See DL06 V-memory map - Data Words

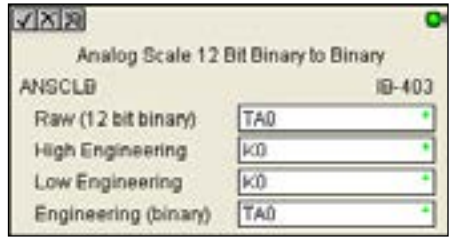
ANSCL Example

In the following example, the ANSCL instruction is used to scale a raw value (0-4095 BCD) that is in V2000. The engineering scaling range is set 0-100 (low engineering value - high engineering value). The scaled value will be placed in V2100 in BCD format.

Analog Scale 12 Bit Binary to Binary (ANSCLB) (IB-403)

DS	Used
HPP	N/A

Analog Scale 12 Bit Binary to Binary scales a 12 bit binary analog value (0-4095 decimal) into binary (decimal) engineering units. You specify the engineering unit high value (when raw is 4095), and the engineering low value (when raw is 0), and the output V-Memory address you want to place the scaled engineering unit value. The engineering units are generated as binary and can be the full range of 0 to 65535 (see ANSCL - Analog Scale 12 Bit BCD to BCD if your raw units are in BCD format).



***NOTE**: This IBox only works with unipolar unsigned raw values. It does NOT work with bipolar, sign plus magnitude, or signed 2's complement raw values.*

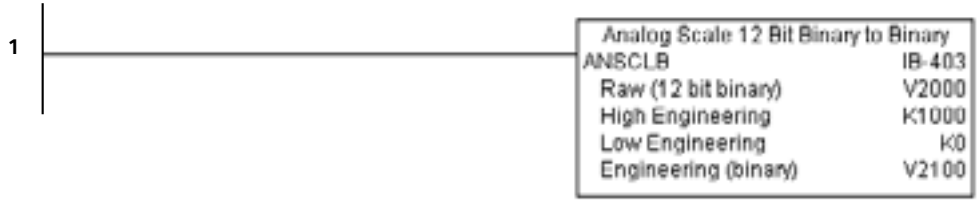
ANSCLB Parameters

- Raw (12 bit binary): specifies the V-memory location of the unipolar unsigned raw decimal unscaled value (12 bit binary = 0-4095 decimal)
- High Engineering: specifies the high engineering value when the raw input is 4095 decimal
- Low Engineering: specifies the low engineering value when the raw input is 0 decimal
- Engineering (binary): specifies the V-memory location where the scaled engineering decimal value will be placed

Parameter		DL06 Range
Raw (12 bit binary)	V,P	See DL06 V-memory map - Data Words
High Engineering	K	K0-65535
Low Engineering	K	K0-65535
Engineering (binary)	V,P	See DL06 V-memory map - Data Words

ANSCLB Example

In the following example, the ANSCLB instruction is used to scale a raw value (0-4095 binary) that is in V2000. The engineering scaling range is set 0-1000 (low engineering value - high engineering value). The scaled value will be placed in V2100 in binary format.



Filter Over Time - BCD (FILTER) (IB-422)

DS	Used
HPP	N/A

Filter Over Time BCD will perform a first-order filter on the Raw Data on a defined time interval. The equation is:

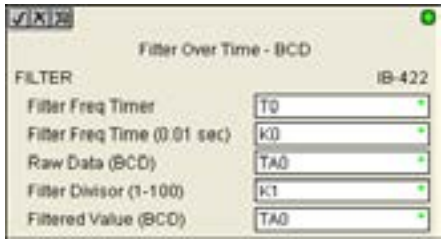
New = Old + [(Raw - Old) / FDC] where,

New: New Filtered Value

Old: Old Filtered Value

FDC: Filter Divisor Constant

Raw: Raw Data



The Filter Divisor Constant is an integer in the range K1 to K100, such that if it equaled K1 then no filtering would be done.

The rate at which the calculation is performed is specified by time in hundredths of a second (0.01 seconds) as the Filter Freq Time parameter. Note that this Timer instruction is embedded in the IBox and must NOT be used anywhere else in your program. Power flow controls whether the calculation is enabled. If it is disabled, the Filter Value is not updated. On the first scan from Program to Run mode, the Filter Value is initialized to 0 to give the calculation a consistent starting point.

FILTER Parameters

- Filter Frequency Timer: specifies the Timer (T) number which is used by the Filter instruction
- Filter Frequency Time (0.01sec): specifies the rate at which the calculation is performed
- Raw Data (BCD): specifies the V-memory location of the raw unfiltered BCD value
- Filter Divisor (1-100): this constant used to control the filtering effect. A larger value will increase the smoothing effect of the filter. A value of 1 results with no filtering.
- Filtered Value (BCD): specifies the V-memory location where the filtered BCD value will be placed

Parameter		DL06 Range
Filter Frequency Timer	T	T0-377
Filter Frequency Time (0.01 sec)	K	K0-9999
Raw Data (BCD)	V	See DL06 V-memory map - Data Words
Filter Divisor (1-100)	K	K1-100
Filtered Value (BCD)	V	See DL06 V-memory map - Data Words

FILTER Example

In the following example, the Filter instruction is used to filter a BCD value that is in V2000. Timer(T0) is set to 0.5 sec, the rate at which the filter calculation will be performed. The filter constant is set to 2. A larger value will increase the smoothing effect of the filter. A value of 1 results with no filtering. The filtered value will be placed in V2100.

Filter Over Time - Binary (FILTERB) (IB-402)

DS	Used
HPP	N/A

Filter Over Time in Binary (decimal) will perform a first-order filter on the Raw Data on a defined time interval. The equation is

New = Old + [(Raw - Old) / FDC] where

New: New Filtered Value

Old: Old Filtered Value

FDC: Filter Divisor Constant

Raw: Raw Data

The Filter Divisor Constant is an integer in the range K1 to K100, such that if it equaled K1 then no filtering would be done.

The rate at which the calculation is performed is specified by time in hundredths of a second (0.01 seconds) as the Filter Freq Time parameter. Note that this Timer instruction is embedded in the IBox and must NOT be used any other place in your program. Power flow controls whether the calculation is enabled. If it is disabled, the Filter Value is not updated. On the first scan from Program to Run mode, the Filter Value is initialized to 0 to give the calculation a consistent starting point.

FILTERB Parameters

- Filter Frequency Timer: specifies the Timer (T) number which is used by the Filter instruction
- Filter Frequency Time (0.01sec): specifies the rate at which the calculation is performed
- Raw Data (Binary): specifies the V-memory location of the raw unfiltered binary (decimal) value
- Filter Divisor (1-100): this constant used to control the filtering effect. A larger value will increase the smoothing effect of the filter. A value of 1 results with no filtering.
- Filtered Value (Binary): specifies the V-memory location where the filtered binary (decimal) value will be placed

Parameter		DL06 Range
Filter Frequency Timer	T	T0-377
Filter Frequency Time (0.01 sec)	K	K0-9999
Raw Data (Binary)	V	See DL06 V-memory map - Data Words
Filter Divisor (1-100)	K	K1-100
Filtered Value (Binary)	V	See DL06 V-memory map - Data Words

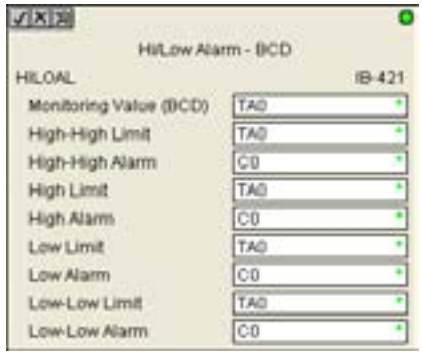
FILTERB Example

In the following example, the FILTERB instruction is used to filter a binary value that is in V2000. Timer (T1) is set to 0.5 sec, the rate at which the filter calculation will be performed. The filter constant is set to 3.0. A larger value will increase the smoothing effect of the filter. A value of 1 results with no filtering. The filtered value will be placed in V2100

Hi/Low Alarm - BCD (HILOAL) (IB-421)

DS	Used
HPP	N/A

Hi/Low Alarm - BCD monitors a BCD value V-Memory location and sets four possible alarm states, High-High, High, Low, and Low-Low whenever the IBox has power flow. You enter the alarm thresholds as constant K BCD values (K0-K9999) and/or BCD value V-Memory locations.



You must ensure that threshold limits are valid, that is HH >= H > L >= LL. Note that when the High-High or Low-Low alarm condition is true, that the High and Low alarms will also be set, respectively. This means you may use the same threshold limit and same alarm bit for the High-High and the High alarms in case you only need one "High" alarm. Also note that the boundary conditions are inclusive. That is, if the Low boundary is K50, and the Low-Low boundary is K10, and if the Monitoring Value equals 10, then the Low Alarm AND the Low-Low alarm will both be ON. If there is no power flow to the IBox, then all alarm bits will be turned off regardless of the value of the Monitoring Value parameter.

HILOAL Parameters

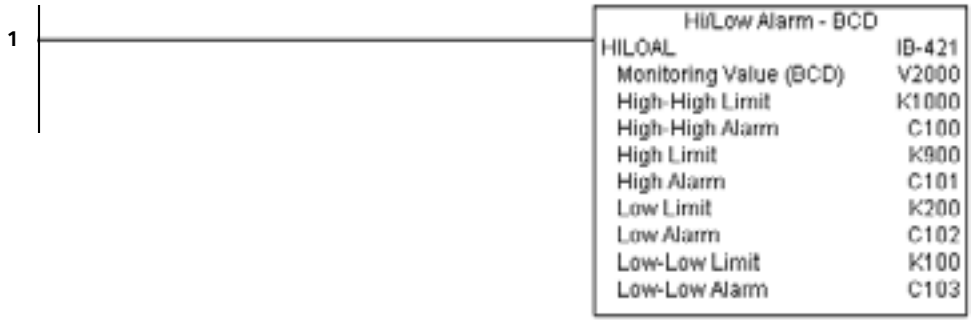
- Monitoring Value (BCD): specifies the V-memory location of the BCD value to be monitored
- High-High Limit: V-memory location or constant specifies the high-high alarm limit
- High-High Alarm: On when the high-high limit is reached
- High Limit: V-memory location or constant specifies the high alarm limit
- High Alarm: On when the high limit is reached
- Low Limit: V-memory location or constant specifies the low alarm limit
- Low Alarm: On when the low limit is reached
- Low-Low Limit: V-memory location or constant specifies the low-low alarm limit
- Low-Low Alarm: On when the low-low limit is reached

Parameter		DL06 Range
Monitoring Value (BCD)	V	See DL06 V-memory map - Data Words
High-High Limit	V, K	K0-9999; or see DL06 V-memory map - Data Words
High-High Alarm	X, Y, C, GX,GY, B	See DL06 V-memory map
High Limit	V, K	K0-9999; or see DL06 V-memory map - Data Words
High Alarm	X, Y, C, GX,GY, B	See DL06 V-memory map
Low Limit	V, K	K0-9999; or see DL06 V-memory map - Data Words
Low Alarm	X, Y, C, GX,GY,B	See DL06 V-memory map
Low-Low Limit	V, K	K0-9999; or see DL06 V-memory map - Data Words
Low-Low Alarm	X, Y, C, GX,GY, B	See DL06 V-memory map

HILOAL Example

In the following example, the HILOAL instruction is used to monitor a BCD value that is in V2000. If the value in V2000 meets/exceeds the high limit of K900, C101 will turn on. If the value continues to increase to meet/exceed the high-high limit, C100 will turn on. Both bits would be on in this case. The high and high-high limits and alarms can be set to the same value if one "high" limit or alarm is desired to be used.

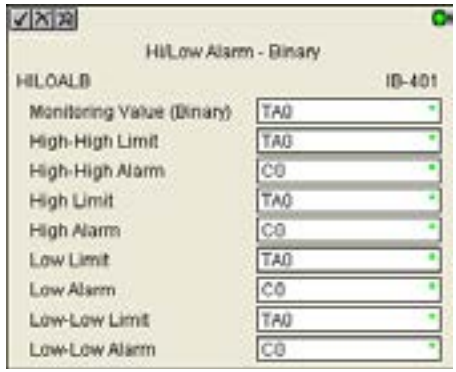
If the value in V2000 meets or falls below the low limit of K200, C102 will turn on. If the value continues to decrease to meet or fall below the low-low limit of K100, C103 will turn on. Both bits would be on in this case. The low and low-low limits and alarms can be set to the same value if one "low" limit or alarm is desired to be used.



Hi/Low Alarm - Binary (HILOALB) (IB-401)

DS	Used
HPP	N/A

Hi/Low Alarm - Binary monitors a binary (decimal) V-Memory location and sets four possible alarm states, High-High, High, Low, and Low-Low whenever the IBox has power flow. You enter the alarm thresholds as constant K decimal values (K0-K65535) and/or binary (decimal) V-Memory locations.



You must ensure that threshold limits are valid, that is HH >= H > L >= LL. Note that when the High-High or Low-Low alarm condition is true, that the High and Low alarms will also be set, respectively. This means you may use the same threshold limit and same alarm bit for the High-High and the High alarms in case you only need one "High" alarm. Also note that the boundary conditions are inclusive. That is, if the Low boundary is K50, and the Low-Low boundary is K10, and if the Monitoring Value equals 10, then the Low Alarm AND the Low-Low alarm will both be ON. If there is no power flow to the IBox, then all alarm bits will be turned off regardless of the value of the Monitoring Value parameter.

HILOALB Parameters

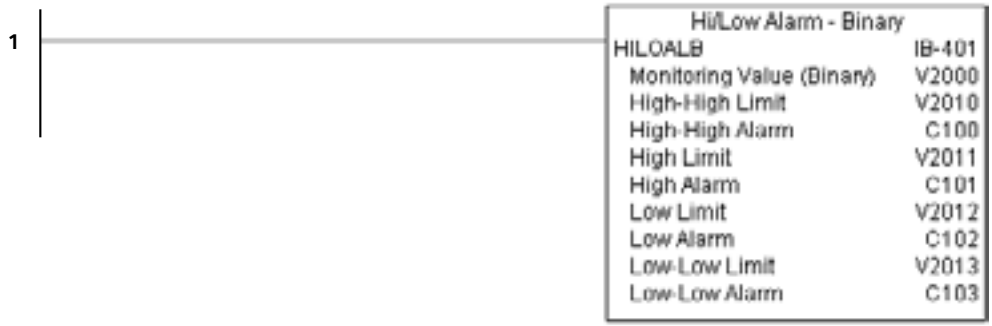
- Monitoring Value (Binary): specifies the V-memory location of the Binary value to be monitored
- High-High Limit: V-memory location or constant specifies the high-high alarm limit
- High-High Alarm: On when the high-high limit is reached
- High Limit: V-memory location or constant specifies the high alarm limit
- High Alarm: On when the high limit is reached
- Low Limit: V-memory location or constant specifies the low alarm limit
- Low Alarm: On when the low limit is reached
- Low-Low Limit: V-memory location or constant specifies the low-low alarm limit
- Low-Low Alarm: On when the low-low limit is reached

Parameter		DL06 Range
Monitoring Value (Binary)	V	See DL06 V-memory map - Data Words
High-High Limit	V, K	K0-65535; or see DL06 V-memory map - Data Words
High-High Alarm	X, Y, C, GX,GY, B	See DL06 V-memory map
High Limit	V, K	K0-65535; or see DL06 V-memory map - Data Words
High Alarm	X, Y, C, GX,GY, B	See DL06 V-memory map
Low Limit	V, K	K0-65535; or see DL06 V-memory map - Data Words
Low Alarm	X, Y, C, GX,GY,B	See DL06 V-memory map
Low-Low Limit	V, K	K0-65535; or see DL06 V-memory map - Data Words
Low-Low Alarm	X, Y, C, GX,GY, B	See DL06 V-memory map

HILOALB Example

In the following example, the HILOALB instruction is used to monitor a binary value that is in V2000. If the value in V2000 meets/exceeds the high limit of the binary value in V2011, C101 will turn on. If the value continues to increase to meet/exceed the high-high limit value in V2010, C100 will turn on. Both bits would be on in this case. The high and high-high limits and alarms can be set to the same V-memory location/value if one "high" limit or alarm is desired to be used.

If the value in V2000 meets or falls below the low limit of the binary value in V2012, C102 will turn on. If the value continues to decrease to meet or fall below the low-low limit in V2013, C103 will turn on. Both bits would be on in this case. The low and low-low limits and alarms can be set to the same V-memory location/value if one "low" limit or alarm is desired to be used.



Off Delay Timer (OFFDTMR) (IB-302)

DS	Used
HPP	N/A

Off Delay Timer will delay the "turning off" of the Output parameter by the specified Off Delay Time (in hundredths of a second) based on the power flow into the IBox. Once the IBox receives power, the Output bit will turn on immediately. When the power flow to the IBox turns off, the Output bit WILL REMAIN ON for the specified amount of time (in hundredths of a second).

Once the Off Delay Time has expired, the output will turn Off. If the power flow to the IBox comes back on BEFORE the Off Delay Time, then the timer is RESET and the Output will remain On - so you must continuously have NO power flow to the IBox for AT LEAST the specified Off Delay Time before the Output will turn Off.

This IBox utilizes a Timer resource (TMRF), which cannot be used anywhere else in your program.

OFFDTMR Parameters

- Timer Number: specifies the Timer(TMRF) number which is used by the OFFDTMR instruction
- Off Delay Time (0.01 sec): specifies how long the Output will remain on once power flow to the Ibox is removed
- Output: specifies the output that will be delayed "turning off" by the Off Delay Time.

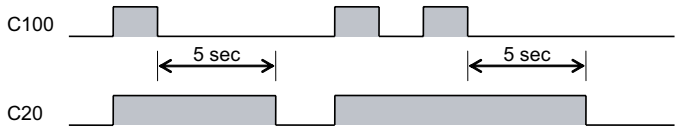
Parameter		DL06 Range
Timer Number	T	T0-377
Off Delay Time	K,V	K0-9999; See DL06 V-memory map - Data Words
Output	X, Y, C, GX,GY, B	See DL06 V-memory map

OFFDTMR Example

In the following example, the OFFDTMR instruction is used to delay the "turning off" of output C20. Timer 2 (T2) is set to 5 seconds, the "off-delay" period.

When C100 turns on, C20 turns on and will remain on while C100 is on. When C100 turns off, C20 will remain for the specified Off Delay Time (5s), and then turn off.

Example timing diagram



On Delay Timer (ONDTMR) (IB-301)

DS	Used
HPP	N/A

On Delay Timer will delay the "turning on" of the Output parameter by the specified amount of time (in hundredths of a second) based on the power flow into the IBox. Once the IBox loses power, the Output is turned off immediately. If the power flow turns off BEFORE the On Delay Time, then the timer is RESET and the Output is never turned on, so you must have continuous power flow to the IBox for at least the specified On Delay Time before the Output turns On.

This IBox utilizes a Timer resource (TMRF), which cannot be used anywhere else in your program.

ONDTMR Parameters

- Timer Number: specifies the Timer(TMRF) number which is used by the ONDTMR instruction
- On Delay Time (0.01sec): specifies how long the Output will remain off once power flow to the Ibox is applied.
- Output: specifies the output that will be delayed "turning on" by the On Delay Time.

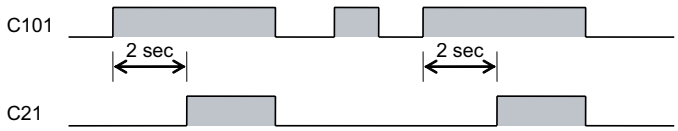
Parameter		DL06 Range
Timer Number	T	T0-377
On Delay Time	K,V	K0-9999; See DL06 V-memory map - Data Words
Output	X, Y, C, GX,GY, B	See DL06 V-memory map

ONDTMR Example

In the following example, the ONDTMR instruction is used to delay the "turning on" of output C21. Timer 1 (T1) is set to 2 seconds, the "on-delay" period.

When C101 turns on, C21 is delayed turning on by 2 seconds. When C101 turns off, C21 turns off immediately.

Example timing diagram



One Shot (ONESHOT) (IB-303)

DS	Used
HPP	N/A

One Shot will turn on the given bit output parameter for one scan on an OFF to ON transition of the power flow into the IBox. This IBox is simply a different name for the PD Coil (Positive Differential).

ONESHOT Parameters

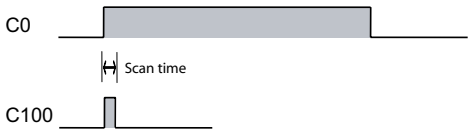
- Discrete Output: specifies the output that will be on for one scan

Parameter		DL06 Range
Discrete Output	X, Y, C	See DL06 V-memory map

ONESHOT Example

In the following example, the ONESHOT instruction is used to turn C100 on for one PLC scan after C0 goes from an off to on transition. The input logic must produce an off to on transition to execute the One Shot instruction.

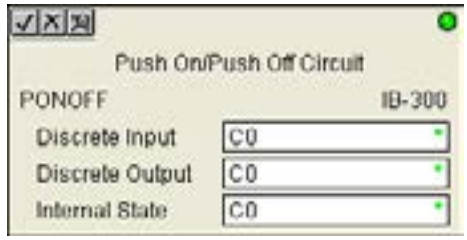
Example timing diagram



Push On / Push Off Circuit (PONOFF) (IB-300)

DS	Used
HPP	N/A

Push On/Push Off Circuit toggles an output state whenever its input power flow transitions from off to on. Requires an extra bit parameter for scan-to-scan state information. This extra bit must NOT be used anywhere else in the program. This is also known as a "flip-flop circuit". The PONOFF IBox cannot have any input logic.



PONOFF Parameters

- Discrete Input: specifies the input that will toggle the specified output
- Discrete Output: specifies the output that will be "turned on/off" or toggled
- Internal State: specifies a work bit that is used by the instruction

Parameter		DL06 Range
Discrete Input	X,Y,C,S,T,CT,GX,GY,SP,B,PB	See DL06 V-memory map
Discrete Output	X,Y,C,GX,GY,B	See DL06 V-memory map
Internal State	X, Y, C	See DL06 V-memory map

PONOFF Example

In the following example, the PONOFF instruction is used to control the on and off states of the output C20 with a single input C10. When C10 is pressed once, C20 turns on. When C10 is pressed again, C20 turns off. C100 is an internal bit used by the instruction.

Move Single Word (MOVEW) (IB-200)

DS	Used
HPP	N/A

Move Single Word moves (copies) a word to a memory location directly or indirectly via a pointer, either as a HEX constant, from a memory location, or indirectly through a pointer

MOVEW Parameters

- From WORD: specifies the word that will be moved to another location
- To WORD: specifies the location where the "From WORD" will be move to

Parameter		DL06 Range
From WORD	V,P,K	K0-FFFF; See DL06 V-memory map - Data Words
To WORD	V,P	See DL06 V-memory map - Data Words

MOVEW Example

In the following example, the MOVEW instruction is used to move 16-bits of data from V2000 to V3000 when C100 turns on.

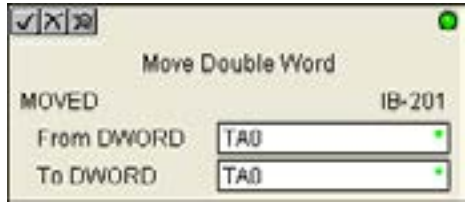
Move Double Word (MOVED) (IB-201)

DS	Used
HPP	N/A

Move Double Word moves (copies) a double word to two consecutive memory locations directly or indirectly via a pointer, either as a double HEX constant, from a double memory location, or indirectly through a pointer to a double memory location.

MOVED Parameters

- From DWORD: specifies the double word that will be moved to another location
- To DWORD: specifies the location where the "From DWORD" will be move to



Parameter		DL06 Range
From DWORD	V,P,K	K0-FFFFFFFF; See DL06 V-memory map - Data Words
To DWORD	V,P	See DL06 V-memory map - Data Words

MOVED Example

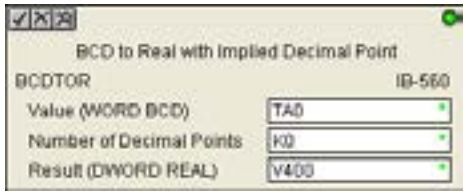
In the following example, the MOVED instruction is used to move 32-bits of data from V2000 and V2001 to V3000 and V3001 when C100 turns on.

BCD to Real with Implied Decimal Point (BCDTOR) (IB-560)

DS	Used
HPP	N/A

BCD to Real with Implied Decimal Point converts the given 4 digit WORD BCD value to a Real number, with the implied number of decimal points (K0-K4).

For example, BCDTOR K1234 with an implied number of decimal points equal to K1, would yield R123.4



BCDTOR Parameters

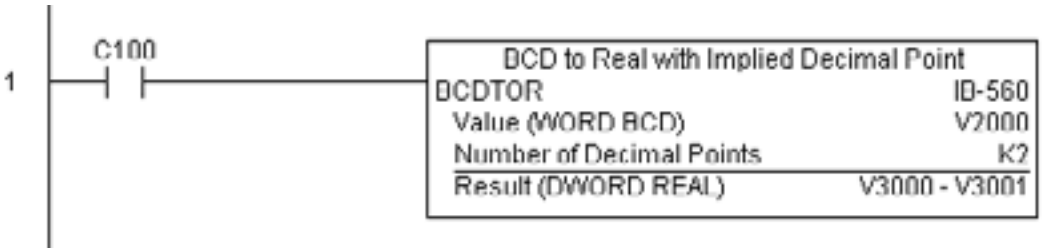
- Value (WORD BCD): specifies the word or constant that will be converted to a Real number
- Number of Decimal Points: specifies the number of implied decimal points in the Result DWORD
- Result (DWORD REAL): specifies the location where the Real number will be placed

Parameter		DL06 Range
Value (WORD BCD)	V,P,K	K0-9999; See DL06 V-memory map - Data Words
Number of Decimal Points	K	K0-4
Result (DWORD REAL)	V	See DL06 V-memory map - Data Words

BCDTOR Example

In the following example, the BCDTOR instruction is used to convert the 16-bit data in V2000 from a 4-digit BCD data format to a 32-bit REAL (floating point) data format and stored into V3000 and V3001 when C100 turns on.

K2 in the Number of Decimal Points implies the data will have two digits to the right of the decimal point.

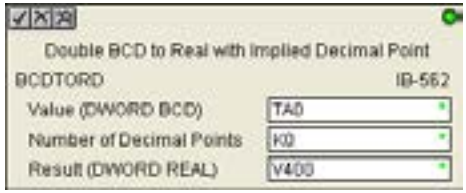


Double BCD to Real with Implied Decimal Point (BCDTORD) (IB-562)

DS	Used
HPP	N/A

Double BCD to Real with Implied Decimal Point converts the given 8 digit DWORD BCD value to a Real number, given an implied number of decimal points (K0-K8).

For example, BCDTORD K12345678 with an implied number of decimal points equal to K5, would yield R123.45678



BCDTORD Parameters

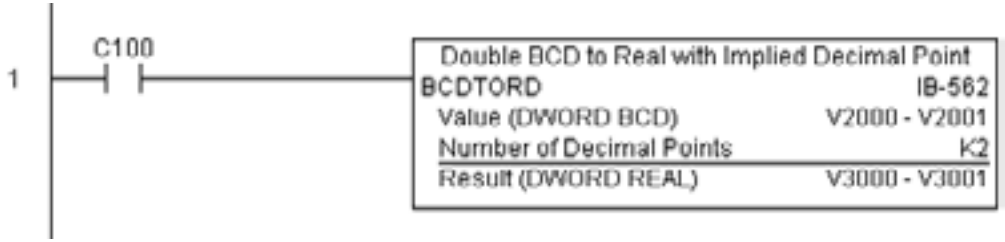
- Value (DWORD BCD): specifies the Dword or constant that will be converted to a Real number
- Number of Decimal Points: specifies the number of implied decimal points in the Result DWORD
- Result (DWORD REAL): specifies the location where the Real number will be placed

Parameter		DL06 Range
Value (DWORD BCD)	V,P,K	K0-99999999; See DL06 V-memory map - Data Words
Number of Decimal Points	K	K0-8
Result (DWORD REAL)	V	See DL06 V-memory map - Data Words

BCDTORD Example

In the following example, the BCDTORD instruction is used to convert the 32-bit data in V2000 from an 8-digit BCD data format to a 32-bit REAL (floating point) data format and stored into V3000 and V3001 when C100 turns on.

K2 in the Number of Decimal Points implies the data will have two digits to the right of the decimal point.

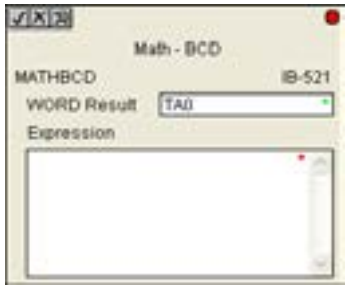


Math - BCD (MATHBCD) (IB-521)

DS	Used
HPP	N/A

Math - BCD Format lets you enter complex mathematical expressions like you would in Visual Basic, Excel, or C++ to do complex calculations, nesting parentheses up to 4 levels deep. In addition to + - * /, you can do Modulo (% aka Remainder), Bit-wise And (&) Or (|) Xor (^), and some BCD functions - Convert to BCD (BCD), Convert to Binary (BIN), BCD Complement (BCDCPL), Convert from Gray Code (GRAY), Invert Bits (INV), and BCD/HEX to Seven Segment Display (SEG).

Example: ((V2000 + V2001) / (V2003 - K100)) * GRAY(V3000 & K001F)



Every V-memory reference MUST be to a single word BCD formatted value. Intermediate results can go up to 32 bit values, but as long as the final result fits in a 16 bit BCD word, the calculation is valid. Typical example of this is scaling using multiply then divide, (V2000 * K1000) / K4095. The multiply term most likely will exceed 9999 but fits within 32 bits. The divide operation will divide 4095 into the 32-bit accumulator, yielding a result that will always fit in 16 bits.

You can reference binary V-memory values by using the BCD conversion function on a V-Memory location but NOT an expression. That is BCD(V2000) is okay and will convert V2000 from Binary to BCD, but BCD(V2000 + V3000) will add V2000 as BCD, to V3000 as BCD, then interpret the result as Binary and convert it to BCD - NOT GOOD.

Also, the final result is a 16 bit BCD number and so you could do BIN around the entire operation to store the result as Binary.

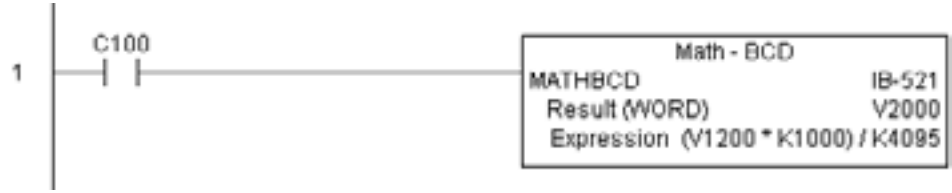
MATHBCD Parameters

- WORD Result: specifies the location where the BCD result of the mathematical expression will be placed (result must fit into 16 bit single V-memory location)
- Expression: specifies the mathematical expression to be executed and the result is stored in specified WORD Result. Each V-memory location used in the expression must be in BCD format.

Parameter		DL06 Range
WORD Result	V	See DL06 V-memory map - Data Words
Expression		Text

MATHBCD Example

In the following example, the MATHBCD instruction is used to calculate the math expression which multiplies the BCD value in V1200 by 1000 then divides by 4095 and loads the resulting value in V2000 when C100 turns on.

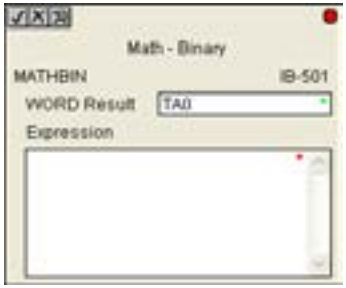


Math - Binary (MATHBIN) (IB-501)

DS	Used
HPP	N/A

Math - Binary Format lets you enter complex mathematical expressions like you would in Visual Basic, Excel, or C++ to do complex calculations, nesting parentheses up to 4 levels deep. In addition to + - * /, you can do Modulo (% aka Remainder), Shift Right (>>) and Shift Left (<<), Bit-wise And (&) Or (|) Xor (^), and some binary functions - Convert to BCD (BCD), Convert to Binary (BIN), Decode Bits (DECO), Encode Bits (ENCO), Invert Bits (INV), HEX to Seven Segment Display (SEG), and Sum Bits (SUM).

Example: ((V2000 + V2001) / (V2003 - K10)) SUM(V3000 & K001F)



Every V-memory reference MUST be to a single word binary formatted value. Intermediate results can go up to 32 bit values, but as long as the final result fits in a 16 bit binary word, the calculation is valid. Typical example of this is scaling using multiply then divide, (V2000 * K1000) / K4095. The multiply term most likely will exceed 65535 but fits within 32 bits. The divide operation will divide 4095 into the 32-bit accumulator, yielding a result that will always fit in 16 bits.

You can reference BCD V-Memory values by using the BIN conversion function on a V-memory location but NOT an expression. That is, BIN(V2000) is okay and will convert V2000 from BCD to Binary, but BIN(V2000 + V3000) will add V2000 as Binary, to V3000 as Binary, then interpret the result as BCD and convert it to Binary - NOT GOOD.

Also, the final result is a 16 bit binary number and so you could do BCD around the entire operation to store the result as BCD.

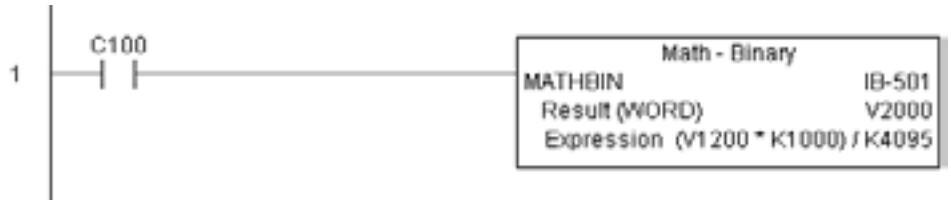
MATHBIN Parameters

- WORD Result: specifies the location where the binary result of the mathematical expression will be placed (result must fit into 16 bit single V-memory location)
- Expression: specifies the mathematical expression to be executed and the result is stored in specified WORD Result. Each V-memory location used in the expression must be in binary format.

Parameter		DL06 Range
WORD Result	V	See DL06 V-memory map - Data Words
Expression		Text

MATHBIN Example

In the following example, the MATHBIN instruction is used to calculate the math expression which multiplies the Binary value in V1200 by 1000 then divides by 4095 and loads the resulting value in V2000 when C100 turns on.



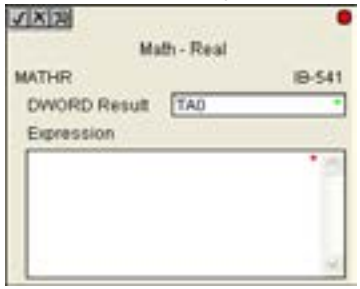
Math - Real (MATHR) (IB-541)

DS	Used
HPP	N/A

Math - Real Format lets you enter complex mathematical expressions like you would in Visual Basic, Excel, or C++ to do complex calculations, nesting parentheses up to 4 levels deep. In addition to + - * /, you can do Bit-wise And (&) Or (|) Xor (^), and many Real functions - Arc Cosine (ACOSR), Arc Sine (ASINR), Arc Tangent (ATANR), Cosine (COSR), Convert Radians to Degrees (DEGR), Invert Bits (INV), Convert Degrees to Radians (RADR), HEX to Seven Segment Display (SEG), Sine (SINR), Square Root (SQRTR), Tangent (TANR).

Example: ((V2000 + V2002) / (V2004 - R2.5)) SINR(RADR(V3000 / R10.0))

Every V-memory reference MUST be able to fit into double word Real formatted value.



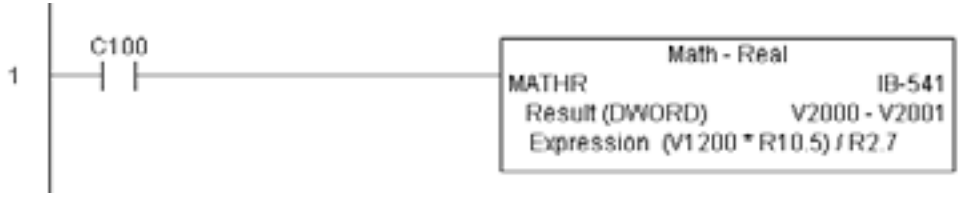
MATHR Parameters

- DWORD Result: specifies the location where the Real result of the mathematical expression will be placed (result must fit into a double word Real formatted location)
- Expression: specifies the mathematical expression to be executed and the result is stored in specified DWORD Result location. Each V-memory location used in the expression must be in Real format.

Parameter		DL06 Range
DWORD Result	V	See DL06 V-memory map - Data Words
Expression		Text

MATHR Example

In the following example, the MATHR instruction is used to calculate the math expression which multiplies the REAL (floating point) value in V1200 by 10.5 then divides by 2.7 and loads the resulting 32-bit value in V2000 and V2001 when C100 turns on.



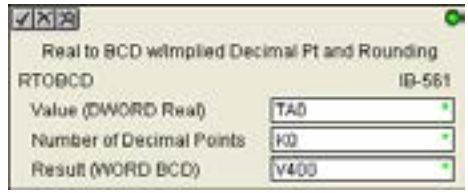
Real to BCD with Implied Decimal Point and Rounding (RTOBCD) (IB-561)

DS	Used
HPP	N/A

Real to BCD with Implied Decimal Point and Rounding converts the absolute value of the given Real number to a 4 digit BCD number, compensating for an implied number of decimal points (K0-K4) and performs rounding.

For example, RTOBCD R56.74 with an implied number of decimal points equal to K1, would yield 567 BCD. If the implied number of decimal points was 0, then the function would yield 57 BCD (note that it rounded up).

If the Real number is negative, the Result will equal its positive, absolute value.



RTOBCD Parameters

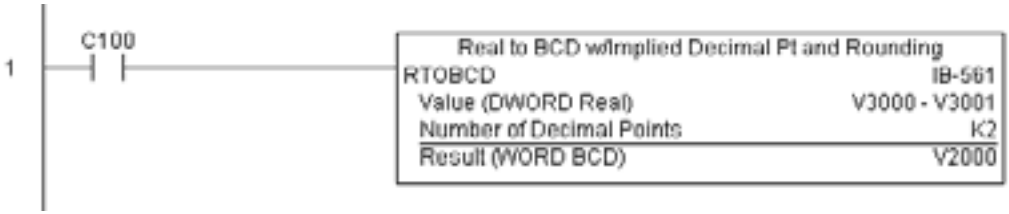
- Value (DWORD Real): specifies the Real Dword location or number that will be converted and rounded to a BCD number with decimal points
- Number of Decimal Points: specifies the number of implied decimal points in the Result WORD
- Result (WORD BCD): specifies the location where the rounded/implied decimal points BCD value will be placed

Parameter		DL06 Range
Value (DWORD Real)	V,P,R	R ; See DL06 V-memory map - Data Words
Number of Decimal Points	K	K0-4
Result (WORD BCD)	V	See DL06 V-memory map - Data Words

RTOBCD Example

In the following example, the RTOBCD instruction is used to convert the 32-bit REAL (floating point) data format in V3000 and V3001 to the 4-digit BCD data format and stored in V2000 when C100 turns on.

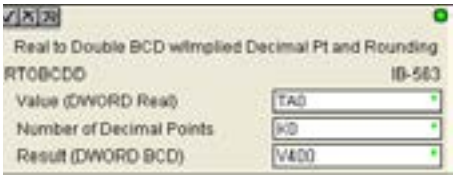
K2 in the Number of Decimal Points implies the data will have two implied decimal points.



Real to Double BCD with Implied Decimal Point and Rounding (RTOBCDD) (IB-563)

DS	Used
HPP	N/A

Real to Double BCD with Implied Decimal Point and Rounding converts the absolute value of the given Real number to an 8 digit DWORD BCD number, compensating for an implied number of decimal points (K0-K8) and performs rounding.



For example, RTOBCDD R38156.74 with an implied number of decimal points equal to K1, would yield 381567 BCD. If the implied number of decimal points was 0, then the function would yield 38157 BCD (note that it rounded up).

If the Real number is negative, the Result will equal its positive, absolute value.

RTOBCDD Parameters

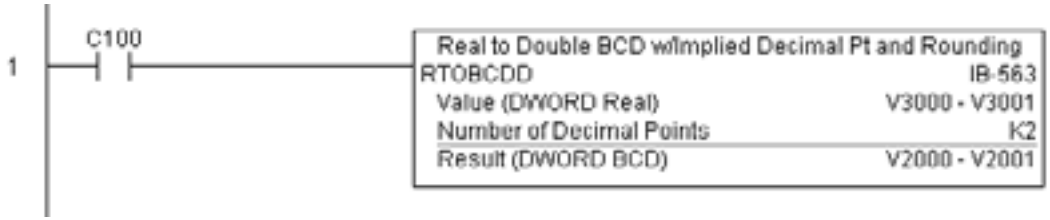
- Value (DWORD Real): specifies the Dword Real number that will be converted and rounded to a BCD number with decimal points
- Number of Decimal Points: specifies the number of implied decimal points in the Result DWORD
- Result (DWORD BCD): specifies the location where the rounded/implied decimal points DWORD BCD value will be placed

Parameter		DL06 Range
Value (DWORD Real)	V,P,R	R ; See DL06 V-memory map - Data Words
Number of Decimal Points	K	K0-8
Result (DWORD BCD)	V	See DL06 V-memory map - Data Words

RTOBCDD Example

In the following example, the RTOBCDD instruction is used to convert the 32-bit REAL (floating point) data format in V3000 and V3001 to the 8-digit BCD data format and stored in V2000 and V2001 when C100 turns on.

K2 in the Number of Decimal Points implies the data will have two implied decimal points.



Square BCD (SQUARE) (IB-523)

DS	Used
HPP	N/A

Square BCD squares the given 4-digit WORD BCD number and writes it in as an 8-digit DWORD BCD result.

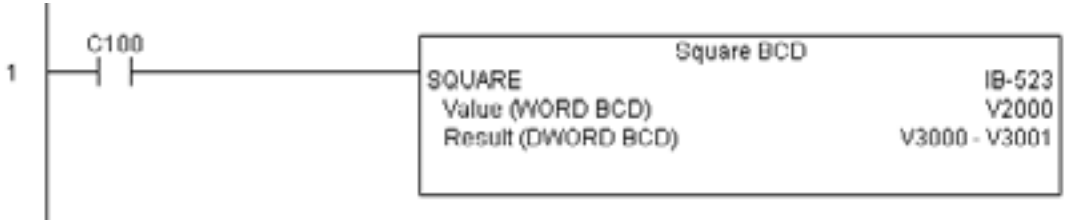
SQUARE Parameters

- Value (WORD BCD): specifies the BCD Word or constant that will be squared
- Result (DWORD BCD): specifies the location where the squared DWORD BCD value will be placed

Parameter		DL06 Range
Value (WORD BCD)	V,P,K	K0-9999 ; See DL06 V-memory map - Data Words
Result (DWORD BCD)	V	See DL06 V-memory map - Data Words

SQUARE Example

In the following example, the SQUARE instruction is used to square the 4-digit BCD value in V2000 and store the 8-digit double word BCD result in V3000 and V3001 when C100 turns on.



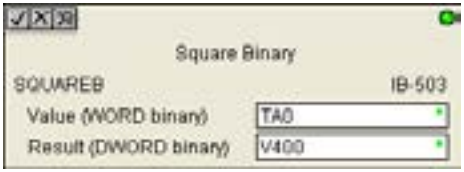
Square Binary (SQUAREB) (IB-503)

DS	Used
HPP	N/A

Square Binary squares the given 16-bit WORD Binary number and writes it as a 32-bit DWORD Binary result.

SQUAREB Parameters

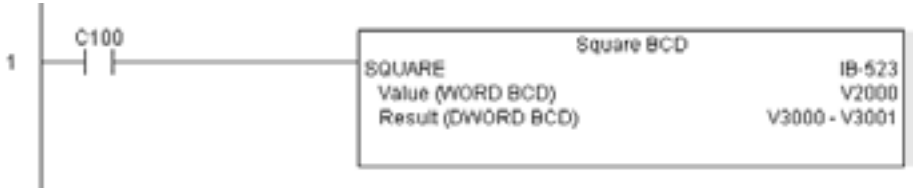
- Value (WORD Binary): specifies the binary Word or constant that will be squared
- Result (DWORD Binary): specifies the location where the squared DWORD binary value will be placed



Parameter		DL06 Range
Value (WORD Binary)	V,P,K	K0-65535; See DL06 V-memory map - Data Words
Result (DWORD Binary)	V	See DL06 V-memory map - Data Words

SQUAREB Example

In the following example, the SQUAREB instruction is used to square the single word Binary value in V2000 and store the 8-digit double word Binary result in V3000 and V3001 when C100 turns on.



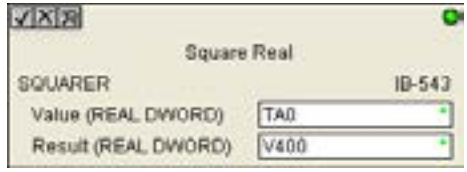
Square Real (SQUARER) (IB-543)

DS	Used
HPP	N/A

Square Real squares the given REAL DWORD number and writes it to a REAL DWORD result.

SQUARER Parameters

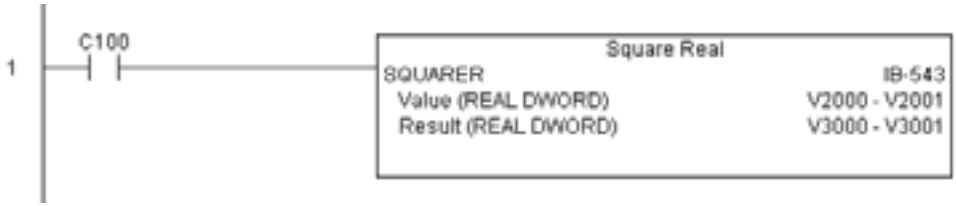
- Value (REAL DWORD): specifies the Real DWORD location or number that will be squared
- Result (REAL DWORD): specifies the location where the squared Real DWORD value will be placed



Parameter		DL06 Range
Value (REAL DWORD)	V,P,R	R ; See DL06 V-memory map - Data Words
Result (REAL DWORD)	V	See DL06 V-memory map - Data Words

SQUARER Example

In the following example, the SQUARER instruction is used to square the 32-bit floating point REAL value in V2000 and V2001 and store the REAL value result in V3000 and V3001 when C100 turns on.



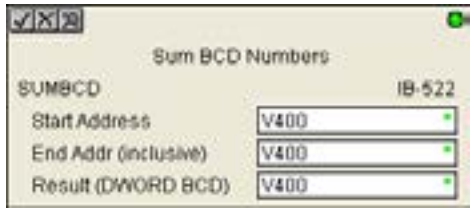
Sum BCD Numbers (SUMBCD) (IB-522)

DS	Used
HPP	N/A

Sum BCD Numbers sums up a list of consecutive 4-digit WORD BCD numbers into an 8-digit DWORD BCD result.

You specify the group's starting and ending V-memory addresses (inclusive). When enabled, this instruction will add up all the numbers in the group (so you may want to place a differential contact driving the enable).

SUMBCD could be used as the first part of calculating an average.



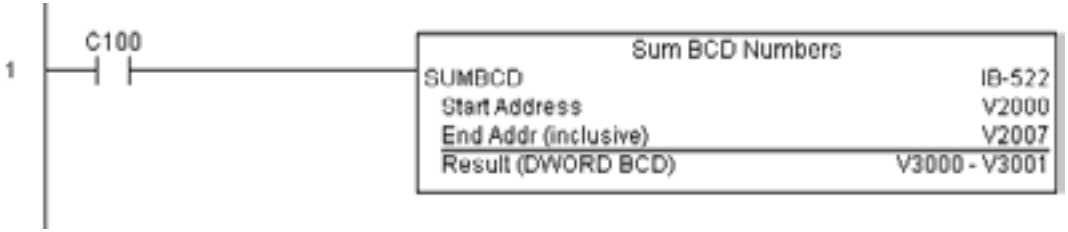
SUMBCD Parameters

- Start Address: specifies the starting address of a block of V-memory location values to be added together (BCD)
- End Addr (inclusive): specifies the ending address of a block of V-memory location values to be added together (BCD)
- Result (DWORD BCD): specifies the location where the sum of the block of V-memory BCD values will be placed

Parameter		DL06 Range
Start Address	V	See DL06 V-memory map - Data Words
End Address (inclusive)	V	See DL06 V-memory map - Data Words
Result (DWORD BCD)	V	See DL06 V-memory map - Data Words

SUMBCD Example

In the following example, the SUMBCD instruction is used to total the sum of all BCD values in words V2000 thru V2007 and store the resulting 8-digit double word BCD value in V3000 and V3001 when C100 turns on.



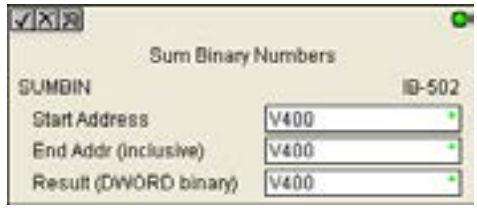
Sum Binary Numbers (SUMBIN) (IB-502)

DS	Used
HPP	N/A

Sum Binary Numbers sums up a list of consecutive 16-bit WORD Binary numbers into a 32-bit DWORD binary result.

You specify the group's starting and ending V-memory addresses (inclusive). When enabled, this instruction will add up all the numbers in the group (so you may want to place a differential contact driving the enable).

SUMBIN could be used as the first part of calculating an average.



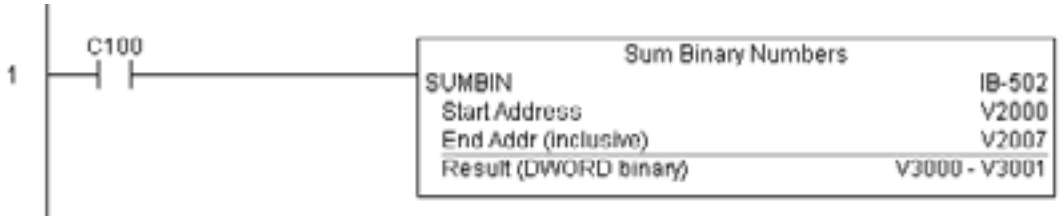
SUMBIN Parameters

- Start Address: specifies the starting address of a block of V-memory location values to be added together (Binary)
- End Addr (inclusive): specifies the ending address of a block of V-memory location values to be added together (Binary)
- Result (DWORD Binary): specifies the location where the sum of the block of V-memory binary values will be placed

Parameter		DL06 Range
Start Address	V	See DL06 V-memory map - Data Words
End Address (inclusive)	V	See DL06 V-memory map - Data Words
Result (DWORD Binary)	V	See DL06 V-memory map - Data Words

SUMBIN Example

In the following example, the SUMBIN instruction is used to total the sum of all Binary values in words V2000 thru V2007 and store the resulting 8-digit double word Binary value in V3000 and V3001 when C100 turns on.



Sum Real Numbers (SUMR) (IB-542)

DS	Used
HPP	N/A

Sum Real Numbers sums up a list of consecutive REAL DWORD numbers into a REAL DWORD result.

You specify the group's starting and ending V- memory addresses (inclusive). Remember that Real numbers are DWORDs and occupy 2 words of V-Memory each, so the number of Real values summed up is equal to half the number of memory locations. Note that the End Address can be EITHER word of the 2 word ending address, for example, if you wanted to add the 4 Real numbers stored in V2000 thru V2007 (V2000, V2002, V2004, and V2006), you can specify V2006 OR V2007 for the ending address and you will get the same result.

When enabled, this instruction will add up all the numbers in the group (so you may want to place a differential contact driving the enable).

SUMR could be used as the first part of calculating an average.

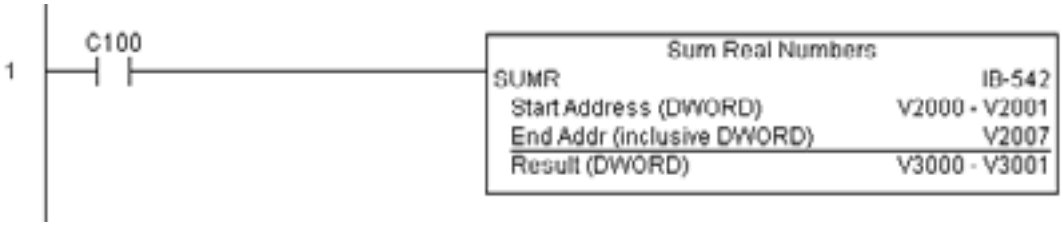
SUMR Parameters

- Start Address (DWORD): specifies the starting address of a block of V-memory location values to be added together (Real)
- End Addr (inclusive) (DWORD): specifies the ending address of a block of V-memory location values to be added together (Real)
- Result (DWORD): specifies the location where the sum of the block of V-memory Real values will be placed

Parameter		DL06 Range
Start Address (inclusive DWORD)	V	See DL06 V-memory map - Data Words
End Address (inclusive DWORD)	V	See DL06 V-memory map - Data Words
Result (DWORD)	V	See DL06 V-memory map - Data Words

SUMR Example

In the following example, the SUMR instruction is used to total the sum of all floating point REAL number values in words V2000 thru V2007 and store the resulting 32-bit floating point REAL number value in V3000 and V3001 when C100 turns on.



ECOM100 Configuration (ECOM100) (IB-710)

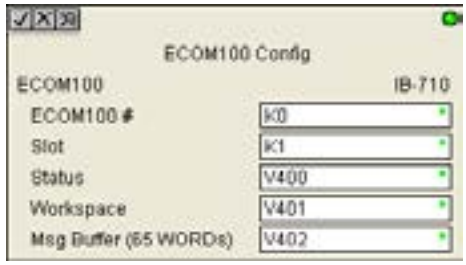
DS	Used
HPP	N/A

ECOM100 Configuration defines the parameters other ECOM100 IBoxes will use when working with this specific ECOM100 module. Each ECOM100 module that will be used with IBox instructions will require a unique ECOM1000 Configuration instruction. The addresses used become workspaces for the IBox instruction to use.

The addresses used in this instruction must not be used elsewhere in the program.

The instructions must be placed at the top of ladder, without a contact. The instruction will inherently run only once, on the first scan.

IBoxes ECEMAIL, ECRX, ECIPSUP and others require an ECOM100 Configuration before they will operate properly.



In order for MOST ECOM100 IBoxes to function, DIP switch 7 on the ECOM100 circuit board must be ON DIP switch 7 can remainOFF if ECOM100 Network Read and Write IBoxes (ECRX, ECWX) are the only IBoxes that will be used.

ECOM100 Configuration Parameters

- ECOM100#: specify a logical number to be associated with this particular ECOM100 module. All other ECxxxx IBoxes that need to reference this ECOM1000 module must reference this logical number
- Slot: specifies the option slot the module occupies
- Status: specifies a V-memory location that will be used by the instruction
- Workspace: specifies a V-memory location that will be used by the instruction
- Msg Buffer: specifies the starting address of a 65 word buffer that will be used by the module for configuration

Parameter		DL06 Range
ECOM100#	K	K0-255
Slot	K	K1-4
Status	V	See DL06 V-memory map - Data Words
Workspace	V	See DL06 V-memory map - Data Words
Msg Buffer (65 words used)	V	See DL06 V-memory map - Data Words

ECOM100 Example

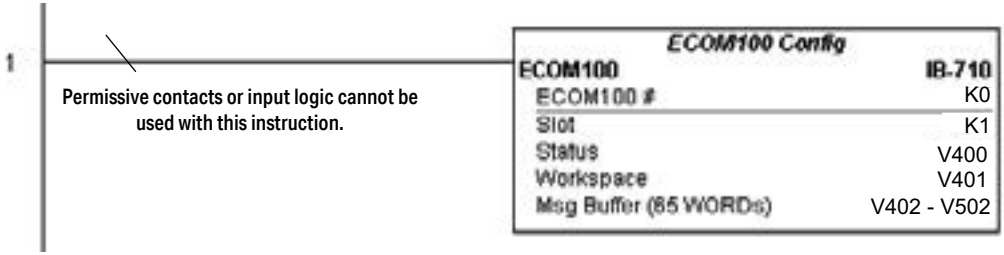
The ECOM100 Config IBox coordinates all of the interaction with other ECOM100 based IBoxes (ECxxxx). You must have an ECOM100 Config IBox for each ECOM100 module in your system. Configuration IBoxes must be at the top of your program and must execute every scan.

This IBox defines ECOM100# K0 to be in slot 3. Any ECOM100 IBoxes that need to reference this specific module (such as ECEMAIL, ECRX, ...) would enter K0 for their ECOM100# parameter.

The Status register is for reporting any completion or error information to other ECOM100 IBoxes. This V-Memory register must not be used anywhere else in the entire program.

The Workspace register is used to maintain state information about the ECOM100, along with proper sharing and interlocking with the other ECOM100 IBoxes in the program. This V-Memory register must not be used anywhere else in the entire program.

The Message Buffer of 65 words (130 bytes) is a common pool of memory that is used by other ECOM100 IBoxes (such as ECEMAIL). This way, you can have a bunch of ECEMAIL IBoxes, but only need 1 common buffer for generating and sending each EMail. These V-Memory registers must not be used anywhere else in your entire program.



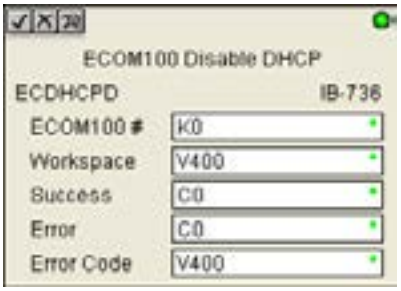
ECOM100 Disable DHCP (ECDHCPD) (IB-736)

DS	Used
HPP	N/A

ECOM100 Disable DHCP will setup the ECOM100 to use its internal TCP/IP settings on a leading edge transition to the IBox. To configure the ECOM100's TCP/IP settings manually, use the NetEdit3 utility, or you can do it programmatically from your PLC program using the ECOM100 IP Setup (ECIPSUP), or the individual ECOM100 IBoxes: ECOM Write IP Address (ECWRIP), ECOM Write Gateway Address (ECWRGWA), and ECOM100 Write Subnet Mask (ECWRSNM).

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Either the Success or Error bit parameter will turn on once the command is complete. If there is an error, the Error Code parameter will report an ECOM100 error code (less than 100), or a PLC logic error (greater than 1000).



The "Disable DHCP" setting is stored in Flash-ROM in the ECOM100 and the execution of this IBox will disable the ECOM100 module for at least a half second until it writes the Flash-ROM. Therefore, it is HIGHLY RECOMMENDED that you only execute this IBox ONCE, on first scan. Since it requires a LEADING edge to execute, use a NORMALLY CLOSED SP0 (STR NOT First Scan) to drive the power flow to the IBox.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.

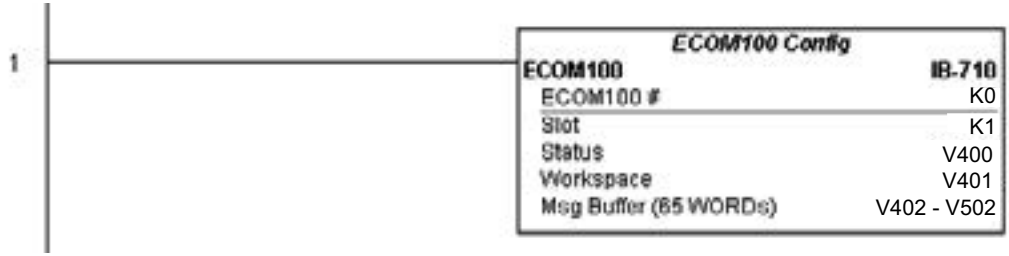
ECDHCPD Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not successfully completed
- Error Code: specifies the location where the Error Code will be written

Parameter		DL06 Range
ECOM100#	K	K0-255
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map
Error Code	V	See DL06 V-memory map - Data Words

ECDHCPD Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module. V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130 byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.



Rung 2: On the 2nd scan, disable DHCP in the ECOM100. DHCP is the same protocol used by PCs for using a DHCP Server to automatically assign the ECOM100's IP Address, Gateway Address, and Subnet Mask. Typically disabling DHCP is done by assigning a hard-coded IP Address either in NetEdit or using one of the ECOM100 IP Setup IBoxes, but this IBox allows you to disable DHCP in the ECOM100 using your ladder program. The ECDHCPD is leading edge triggered, not power-flow driven (similar to a counter input leg). The command to disable DHCP will be sent to the ECOM100 whenever the power flow into the IBox goes from OFF to ON. If successful, turn on C100. If there is a failure, turn on C101. If it fails, you can look at V2000 for the specific error code.

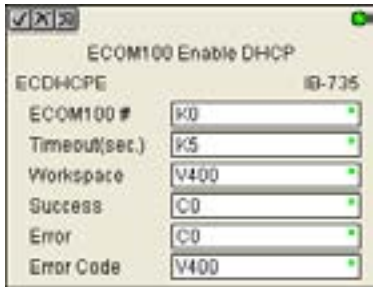
ECOM100 Enable DHCP (ECDHCPE) (IB-735)

DS	Used
HPP	N/A

ECOM100 Enable DHCP will tell the ECOM100 to obtain its TCP/IP setup from a DHCP Server on a leading edge transition to the IBox.

The IBox will be successful once the ECOM100 has received its TCP/IP settings from the DHCP server. Since it is possible for the DHCP server to be unavailable, a Timeout parameter is provided so that the IBox can complete, but with an Error (Error Code = 1004 decimal).

See also the ECOM100 IP Setup (ECIPSUP) IBox 717 to directly setup ALL of the TCP/IP parameters in a single instruction - IP Address, Subnet Mask, and Gateway Address.



The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Either the Success or Error bit parameter will turn on once the command is complete. If there is an error, the Error Code parameter will report an ECOM100 error code (less than 100), or a PLC logic error (greater than 1000).

The "Enable DHCP" setting is stored in Flash-ROM in the ECOM100 and the execution of this IBox will disable the ECOM100 module for at least a half second until it writes the Flash-ROM. Therefore, it is HIGHLY RECOMMENDED that you only execute this IBox ONCE, on first scan. Since it requires a LEADING edge to execute, use a NORMALLY CLOSED SP0 (STR NOT First Scan) to drive the power flow to the IBox.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.

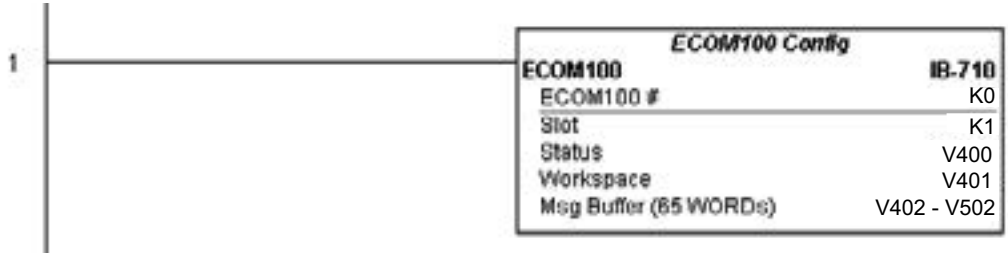
ECDHCPE Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Timeout(sec): specifies a timeout period so that the instruction may have time to complete
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not successfully completed
- Error Code: specifies the location where the Error Code will be written

Parameter		DL06 Range
ECOM100#	K	K0-255
Timeout (sec)	K	K5-127
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map
Error Code	V	See DL06 V-memory map - Data Words

ECDHCPE Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module.V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130 byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.



Rung 2: On the 2nd scan, enable DHCP in the ECOM100. DHCP is the same protocol used by PCs for using a DHCP Server to automatically assign the ECOM100's IP Address, Gateway Address, and Subnet Mask. Typically this is done using NetEdit, but this IBox allows you to enable DHCP in the ECOM100 using your ladder program. The ECDHCPE is leading edge triggered, not power-flow driven (similar to a counter input leg). The commands to enable DHCP will be sent to the ECOM100 whenever the power flow into the IBox goes from OFF to ON. The ECDHCPE does more than just set the bit to enable DHCP in the ECOM100, but it then polls the ECOM100 once every second to see if the ECOM100 has found a DHCP server and has a valid IP Address. Therefore, a timeout parameter is needed in case the ECOM100 cannot find a DHCP server. If a timeout does occur, the Error bit will turn on and the error code will be 1005 decimal. The Success bit will turn on only if the ECOM100 finds a DHCP Server and is assigned a valid IP Address. If successful, turn on C100. If there is a failure, turn on C101. If it fails, you can look at V2000 for the specific error code.

ECOM100 Query DHCP Setting (ECDHCPQ) (IB-734)

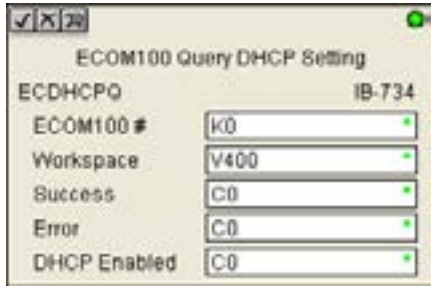
DS	Used
HPP	N/A

ECOM100 Query DHCP Setting will determine if DHCP is enabled in the ECOM100 on a leading edge transition to the IBox. The DHCP Enabled bit parameter will be ON if DHCP is enabled, OFF if disabled.

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Either the Success or Error bit parameter will turn on once the command is complete.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.



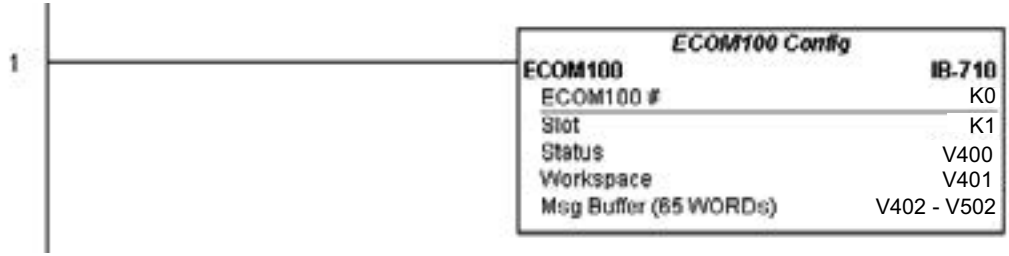
ECDHCPQ Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the instruction is completed successfully
- Error: specifies a bit that will turn on if the instruction is not successfully completed
- DHCP Enabled: specifies a bit that will turn on if the ECOM100's DHCP is enabled or remain off if disabled - after instruction query, be sure to check the state of the Success/ Error bit state along with DHCP Enabled bit state to confirm a successful module query

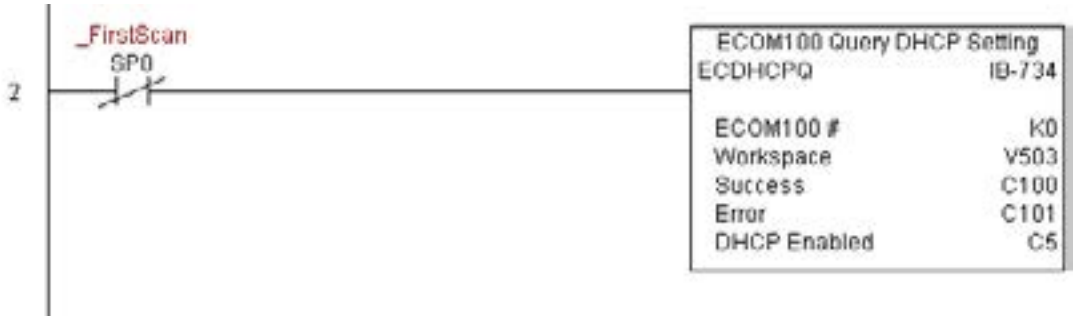
Parameter		DL06 Range
ECOM100#	K	K0-255
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map
DHCP Enabled	X,Y,C,GX,GY,B	See DL06 V-memory map

ECDHCPQ Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module.V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130 byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.



Rung 2: On the 2nd scan, read whether DHCP is enabled or disabled in the ECOM100 and store it in C5. DHCP is the same protocol used by PCs for using a DHCP Server to automatically assign the ECOM100's IP Address, Gateway Address, and Subnet Mask. The ECDHCPQ is leading edge triggered, not power-flow driven (similar to a counter input leg). The command to read (Query) whether DHCP is enabled or not will be sent to the ECOM100 whenever the power flow into the IBox goes from OFF to ON. If successful, turn on C100. If there is a failure, turn on C101.



ECOM100 Send E-mail (ECEMAIL) (IB-711)

DS	Used
HPP	N/A

ECOM100 Send EMail, on a leading edge transition, will behave as an EMail client and send an SMTP request to your SMTP Server to send the EMail message to the EMail addresses in the To: field and also to those listed in the Cc: list hard coded in the ECOM100. It will send the SMTP request based on the specified ECOM100#, which corresponds to a specific unique ECOM100 Configuration (ECOM100) at the top of your program.

The Body: field supports what the PRINT and VPRINT instructions support for text and embedded variables, allowing you to embed real-time data in your EMail (e.g. "V2000 = " V2000:B).

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Either the Success or Error bit parameter will turn on once the request is complete. If there is an error, the Error Code parameter will report an ECOM100 error code (less than 100), an SMPT protocol error (between 100 and 999), or a PLC logic error (greater than 1000).

Since the ECOM100 is only an EMail Client and requires access to an SMTP Server, you MUST have the SMTP parameters configured properly in the ECOM100 via the ECOM100's Home Page and/or the EMail Setup instruction (ECEMSUP). To get to the ECOM100's Home Page, use your favorite Internet browser and browse to the ECOM100's IP Address, e.g. http://192.168.12.86

You are limited to approximately 100 characters of message data for the entire instruction, including the To: Subject: and Body: fields. To save space, the ECOM100 supports a hard coded list of EMail addresses for the Carbon Copy field (cc:) so that you can configure those in the ECOM100, and keep the To: field small (or even empty), to leave more room for the Subject: and Body: fields.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.

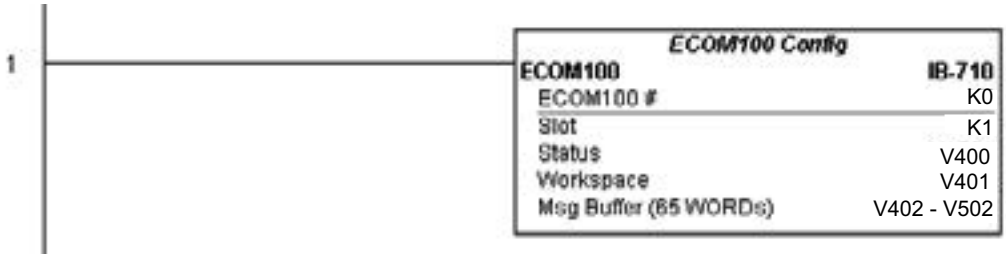
ECEMAIL Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not successfully completed
- Error Code: specifies the location where the Error Code will be written
- To: specifies an E-mail address that the message will be sent to
- Subject: subject of the e-mail message
- Body: supports what the PRINT and VPRINT instructions support for text and embedded variables, allowing you to embed real-time data in the EMail message

Parameter		DL06 Range
ECOM100#	K	K0-255
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map
Error Code	V	See DL06 V-memory map
To:		Text
Subject:		Text
Body:		See PRINT and VPRINT instructions

ECEMAIL Example

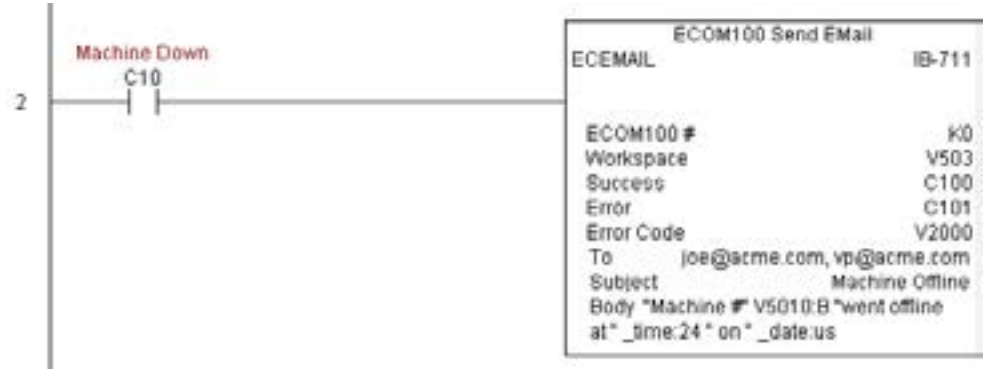
Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module.V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130 byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.



Rung 2: When a machine goes down, send an email to Joe in maintenance and to the VP over production showing what machine is down along with the date/time stamp of when it went down.

The ECEMAIL is leading edge triggered, not power-flow driven (similar to a counter input leg). An email will be sent whenever the power flow into the IBox goes from OFF to ON. This helps prevent self inflicted spamming.

If the EMail is sent, turn on C100. If there is a failure, turn on C101. If it fails, you can look at V2000 for the SMTP error code or other possible error codes.



ECEMAIL Decimal Status Codes

This list of status codes is based on the list in the *ECOM100 Mock Slave Address 89 Command Specification*.

ECOM100 Status codes can be classified into four different areas based on its **decimal** value.

ECOM100 Status Code Areas	
0-1	Normal Status - no error
2-99	Internal ECOM100 errors
100-999	Standard TCP/IP protocol errors (SMTP, HTTP, etc)
1000+	IBox ladder logic assigned errors (SP Slot Error, etc)

For the ECOM100 Send EMail IBOX, the status codes below are specific to this IBox.

Normal Status 0 - 1

ECOMM100 Send EMAIL IBOX Status Codes	
0-1	Success - ECEMAIL completed successfully.
1	Busy - ECEMAIL IBOX logic sets the Error register to this value when the ECEMAIL starts a new request.

Internal ECOM100 Errors (2-99)

Internal ECOM 100 Errors (2-99)	
10-19	Timeout Errors - last digit shows where in ECOM100's SMTP state logic the timeout occurred; *regardless of the last digit*, the SMTP conversation with the SMTP Server timed out.
	SMTP Internal Errors (20-29)
20	TCP Write Error
21	No Sendee
22	Invalid State
23	Invalid Data
24	Invalid SMTP Configuration
25	Memory Allocation Error

ECEMAIL IBox Ladder Logic Assigned Errors (1000+)

ECEMAIL IBox Ladder Logic Assigned Errors (1000+)	
101	SP Slot Error - the SP error bit for the ECOM100's slot turned on. Possibly using RX or WX instructions on the ECOM100 and walking on the ECEMAIL execution. Use should use ECRX and ECX IBoxes,

ECEMAIL Decimal Status Codes

SMTP Protocol Errors - SMTP (100-999)

SMTP Protocol Errors - SMTP (100-999)	
Error	**Description**
1xx	Informational replies
2xx	Success replies
200	(Non-standard success response)
211	System status or system help reply
214	Help message
220	<domain> Service ready. Ready to start TLS
221	<domain> Service closing transmission channel
250	Ok, queuing for node <node> started. Requested mail action okay, completed
251	Ok, no messages waiting for node <node>. User not local; will forward to <forward-path>
252	OK, pending messages for node <node> started. Cannot VRFY user (e.g., info is not local), but will take message for this user and attempt delivery.
253	OK, messages pending messages for node <node> started
3xx	(re) direction replies
354	Start mail input; end with <CRLF>.<CRLF>
355	Octet-offset is the transaction offset.
4xx	client/request error replies
421	<domain> Service not available, closing transmission channel
432	A password transition is needed
450	Requested mail action not taken: mailbox unavailable. ATRN request refused.
451	Requested action aborted; local error in processing. Unable to process ATRN request now.
452	Requested action not taken: insufficient system storage
453	You have no mail
454	TLS is not available due to temporary reason. Encryption required for requested authentication mechanism.
458	Unable to queue messages for node <node>
459	Node <node> not allowed: <reason>
5xx	Server/process error replies
500	Syntax error, command unrecognized. Syntax error.
501	Syntax error in parameters or arguments
502	Command not implemented
503	Bad sequence of commands
504	Command parameter not implemented
521	<domain> does not accept mail
530	Access denied. Must issue a STARTTLS command first. Encryption required for requested authentication mechanism.

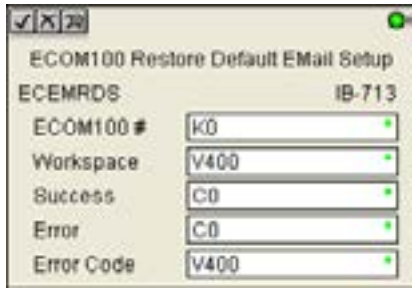
ECOM100 Restore Default E-mail Setup (ECEMRDS) (IB-713)

DS	Used
HPP	N/A

ECOM100 Restore Default EMail Setup, on a leading edge transition, will restore the original EMail Setup data stored in the ECOM100 back to the working copy based on the specified ECOM100#, which corresponds to a specific unique ECOM100 Configuration (ECOM100) at the top of your program.

When the ECOM100 is first powered up, it copies the EMail setup data stored in ROM to the working copy in RAM. You can then modify this working copy from your program using the ECOM100 EMail Setup (ECEMSUP) IBox. After modifying the working copy, you can later restore the original setup data via your program by using this IBox.

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.



Either the Success or Error bit parameter will turn on once the command is complete. If there is an error, the Error Code parameter will report an ECOM100 error code (less than 100), or a PLC logic error (greater than 1000).

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.

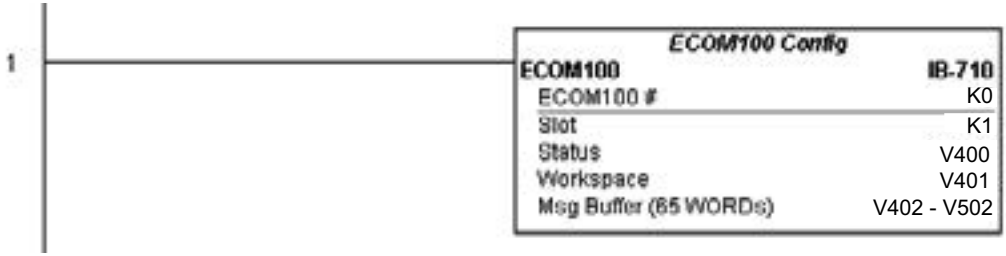
ECEMRDS Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not successfully completed
- Error Code: specifies the location where the Error Code will be written

Parameter		DL06 Range
ECOM100#	K	K0-255
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map
Error Code	V	See DL06 V-memory map - Data Words

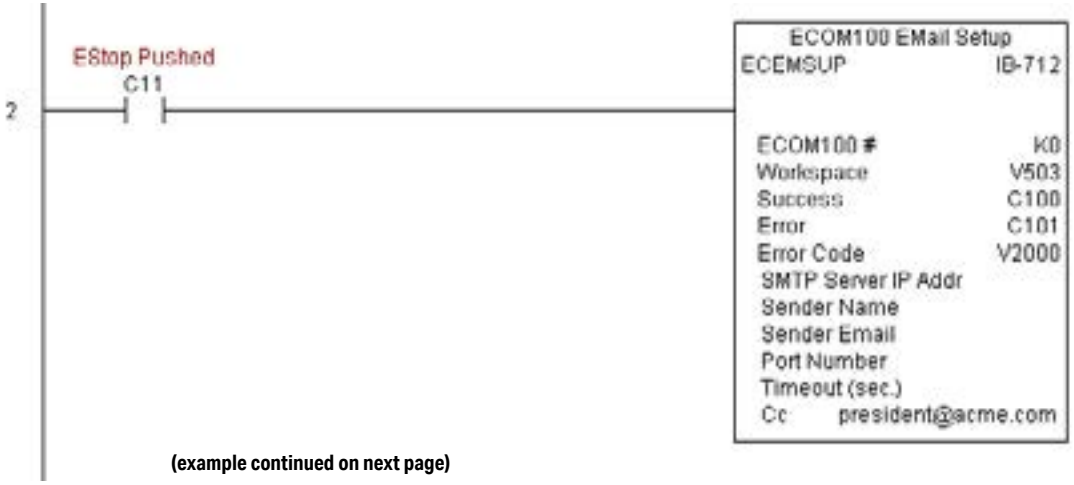
ECEMRDS Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module. V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130 byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.



Rung 2: Whenever an EStop is pushed, ensure that president of the company gets copies of all EMails being sent.

The ECOM100 EMail Setup IBox allows you to set/change the SMTP EMail settings stored in the ECOM100.



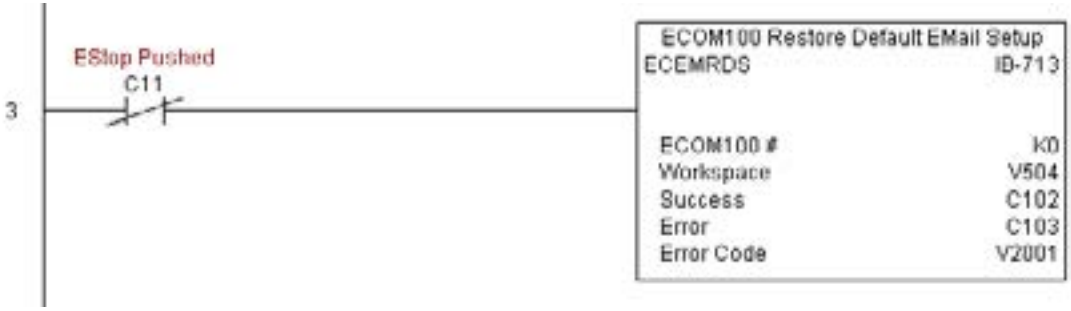
(example continued on next page)

ECEMRDS Example (cont'd)

Rung 3: Once the EStop is pulled out, take the president off the cc: list by restoring the default EMail setup in the ECOM100.

The ECEMRDS is leading edge triggered, not power-flow driven (similar to a counter input leg). The ROM based EMail configuration stored in the ECOM100 will be copied over the "working copy" whenever the power flow into the IBox goes from OFF to ON (the working copy can be changed by using the ECEMSUP IBox).

If successful, turn on C102. If there is a failure, turn on C103. If it fails, you can look at V2001 for the specific error code.



ECOM100 E-mail Setup (ECEMSUP) (IB-712)

DS	Used
HPP	N/A

ECOM100 E-Mail Setup, on a leading edge transition, will modify the working copy of the E-Mail setup currently in the ECOM100 based on the specified ECOM100#, which corresponds to a specific unique ECOM100 Configuration (ECOM100) at the top of your program.

You may pick and choose any or all fields to be modified using this instruction. Note that these changes are cumulative: if you execute multiple ECOM100 EMail Setup IBoxes, then all of the changes are made in the order they are executed. Also note that you can restore the original ECOM100 EMail Setup that is stored in the ECOM100 to the working copy by using the ECOM100 Restore Default EMail Setup (ECEMRDS) IBox.

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Either the Success or Error bit parameter will turn on once the command is complete. If there is an error, the Error Code parameter will report an ECOM100 error code (less than 100), or a PLC logic error (greater than 1000).

You are limited to approximately 100 characters/bytes of setup data for the entire instruction. So if needed, you could divide the entire setup across multiple ECEMSUP IBoxes on a field-by-field basis, for example do the Carbon Copy (cc:) field in one ECEMSUP IBox and the remaining setup parameters in another.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.

ECEMSUP Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not successfully completed
- Error Code: specifies the location where the Error Code will be written
- SMTP Server IP Addr: optional parameter that specifies the IP Address of the SMTP Server on the ECOM100's network
- Sender Name: optional parameter that specifies the sender name that will appear in the "From:" field to those who receive the e-mail
- Sender EMail: optional parameter that specifies the sender EMail address that will appear in the "From:" field to those who receive the e-mail

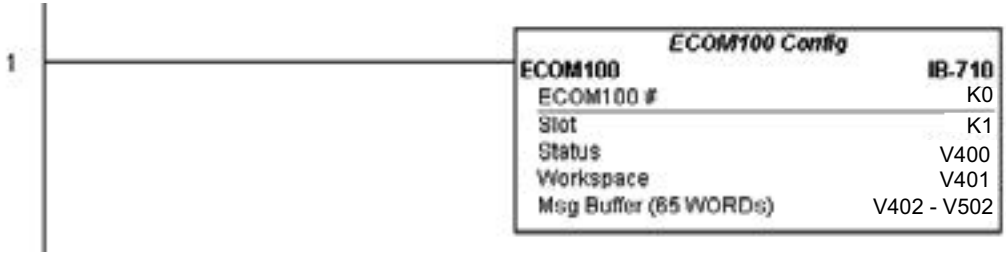
ECEMSUP Parameters (cont'd)

- Port Number: optional parameter that specifies the TCP/IP Port Number to send SMTP requests; usually this does not to be configured (see your network administrator for information on this setting)
- Timeout (sec): optional parameter that specifies the number of seconds to wait for the SMTP Server to send the EMail to all the recipients
- Cc: optional parameter that specifies a list of "carbon copy" Email addresses to send all EMails to

Parameter		DL06 Range
ECOM100#	K	K0-255
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map
Error Code	V	See DL06 V-memory map - Data Words

ECEMSUP Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module.V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130 byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.

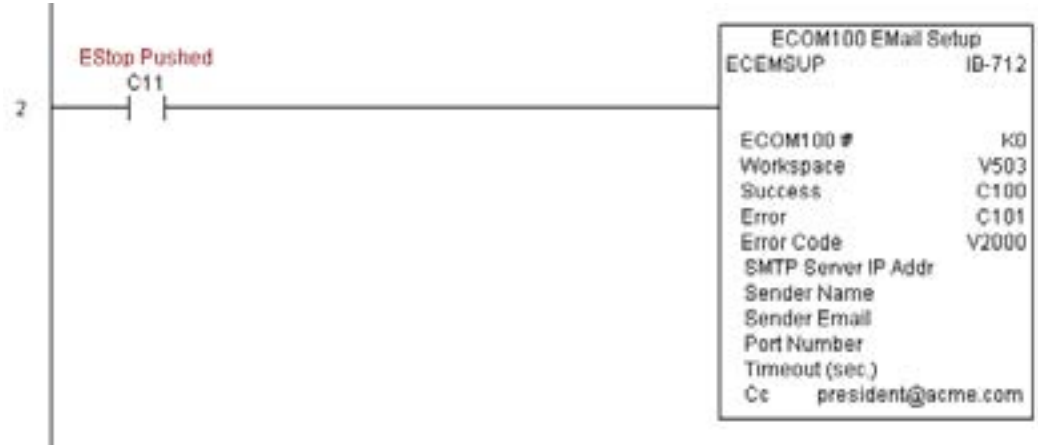


(example continued on next page)

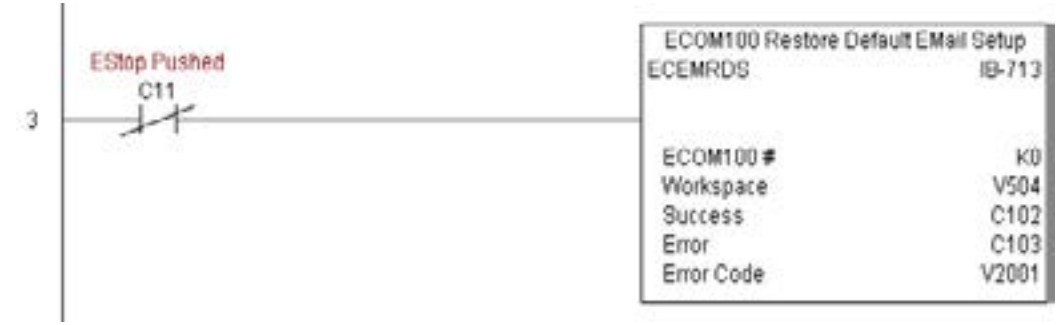
ECEMSUP Example (cont'd)

Rung 2: Whenever an EStop is pushed, ensure that president of the company gets copies of all EMails being sent.The ECOM100 EMail Setup IBox allows you to set/change the SMTP EMail settings stored in the ECOM100. The ECEMSUP is leading edge triggered, not power-flow driven (similar to a counter input leg). At power-up, the ROM based EMail configuration stored in the ECOM100 is copied to a RAM based "working copy". You can change this working copy by using the ECEMSUP IBox. To restore the original ROM based configuration, use the Restore Default EMail Setup ECEMRDS IBox.

If successful, turn on C100. If there is a failure, turn on C101. If it fails, you can look at V2000 for the specific error code.



Rung 3: Once the EStop is pulled out, take the president off the cc: list by restoring the default EMail setup in the ECOM100.



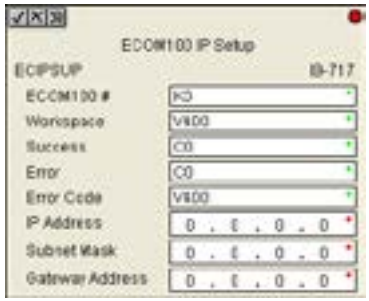
ECOM100 IP Setup (ECIPSUP) (IB-717)

DS	Used
HPP	N/A

ECOM100 IP Setup will configure the three TCP/IP parameters in the ECOM100: IP Address, Subnet Mask, and Gateway Address, on a leading edge transition to the IBox. The ECOM100 is specified by the ECOM100#, which corresponds to a specific unique ECOM100 Configuration (ECOM100) IBox at the top of your program.

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Either the Success or Error bit parameter will turn on once the command is complete. If there is an error, the Error Code parameter will report an ECOM100 error code (less than 100), or a PLC logic error (greater than 1000).



This setup data is stored in Flash-ROM in the ECOM100 and will disable the ECOM100 module for at least a half second until it writes the Flash-ROM. Therefore, it is HIGHLY RECOMMENDED that you only execute this IBox ONCE on first scan. Since it requires a LEADING edge to execute, use a NORMALLY CLOSED SP0 (NOT First Scan) to drive the power flow to the IBox.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.

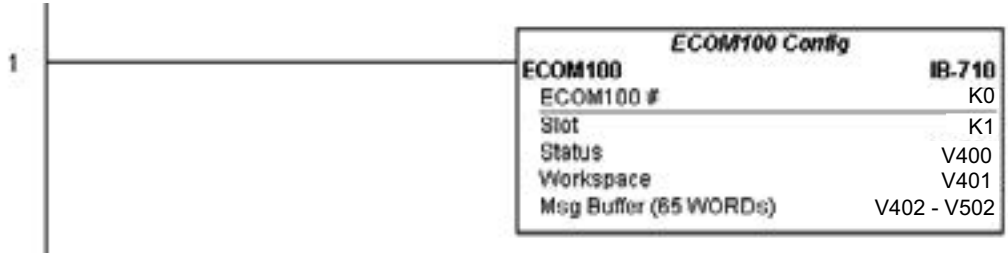
ECIPSUP Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not successfully completed
- Error Code: specifies the location where the Error Code will be written
- IP Address: specifies the module's IP Address
- Subnet Mask: specifies the Subnet Mask for the module to use
- Gateway Address: specifies the Gateway Address for the module to use

Parameter		DL06 Range
ECOM100#	K	K0-255
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map
Error Code	V	See DL06 V-memory map - Data Words
IP Address	IP Address	0.0.0.1. to 255.255.255.254
Subnet Mask Address	IP Address Mask	0.0.0.1. to 255.255.255.254
Gateway Address	IP Address	0.0.0.1. to 255.255.255.254

ECIPSUP Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module.V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130 byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.



Rung 2: On the 2nd scan, configure all of the TCP/IP parameters in the ECOM100:

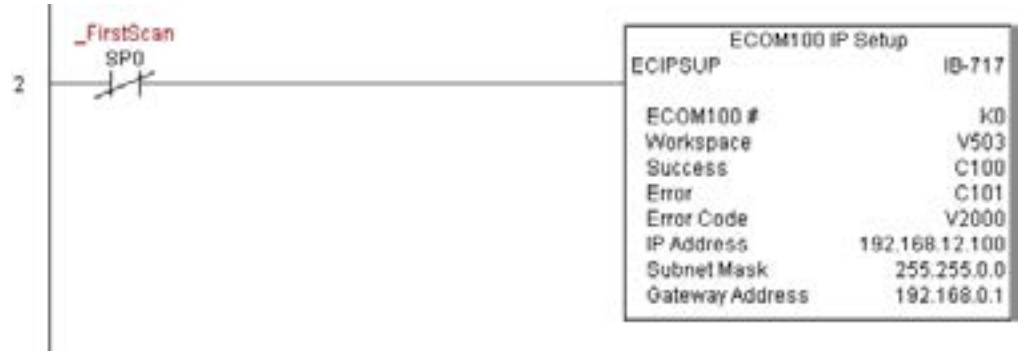
IP Address: 192.168. 12.100

Subnet Mask: 255.255. 0. 0

Gateway Address: 192.168. 0. 1

The ECIPSUP is leading edge triggered, not power-flow driven (similar to a counter input leg). The command to write the TCP/IP configuration parameters will be sent to the ECOM100 whenever the power flow into the IBox goes from OFF to ON.

If successful, turn on C100. If there is a failure, turn on C101. If it fails, you can look at V2000 for the specific error code.



ECOM100 Read Description (ECRDDES) (IB-726)

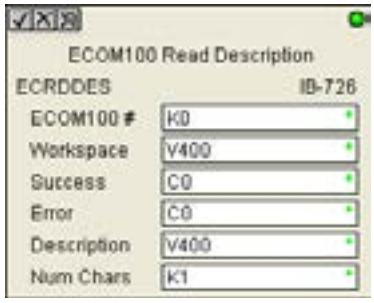
DS	Used
HPP	N/A

ECOM100 Read Description will read the ECOM100's Description field up to the number of specified characters on a leading edge transition to the IBox.

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Either the Success or Error bit parameter will turn on once the command is complete.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.



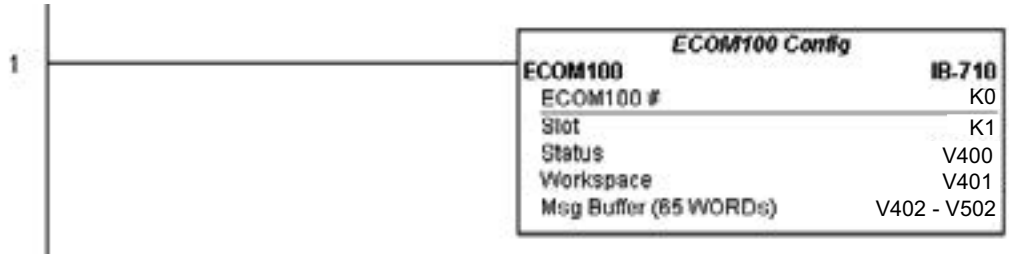
ECRDDES Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not successfully completed
- Description: specifies the starting buffer location where the ECOM100's Module Name will be placed
- Num Char: specifies the number of characters (bytes) to read from the ECOM100's Description field

Parameter		DL06 Range
ECOM100#	K	K0-255
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map
Description	V	See DL06 V-memory map - Data Words
Num Chars	K	K1-128

ECRDDES Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module.V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130 byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.



Rung 2: On the 2nd scan, read the Module Description of the ECOM100 and store it in V3000 thru V3007 (16 characters). This text can be displayed by an HMI.

The ECRDDES is leading edge triggered, not power-flow driven (similar to a counter input leg). The command to read the module description will be sent to the ECOM100 whenever the power flow into the IBox goes from OFF to ON.

If successful, turn on C100. If there is a failure, turn on C101.

ECOM100 Read Gateway Address (ECRDGWA) (IB-730)

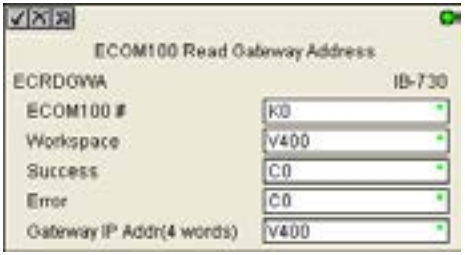
DS	Used
HPP	N/A

ECOM100 Read Gateway Address will read the 4 parts of the Gateway IP address and store them in 4 consecutive V-Memory locations in decimal format, on a leading edge transition to the IBox.

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Either the Success or Error bit parameter will turn on once the command is complete.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.



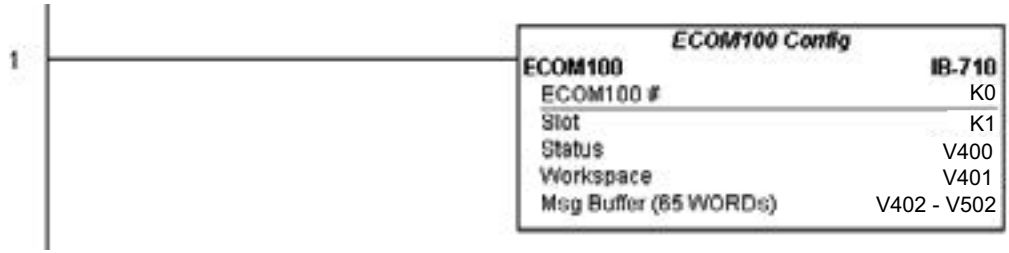
ECRDGWA Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not successfully completed
- Gateway IP Addr: specifies the starting address where the ECOM100's Gateway Address will be placed in 4 consecutive V-memory locations

Parameter		DL06 Range
ECOM100#	K	K0-255
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map
Gateway IP Address (4 Words)	V	See DL06 V-memory map - Data Words

ECRDGWA Example

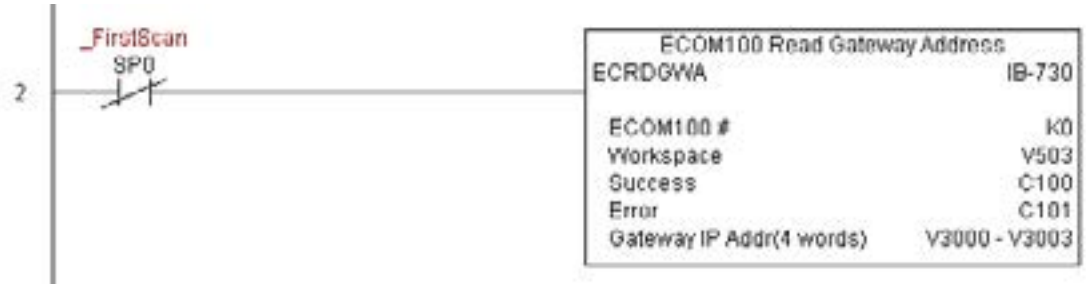
Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module.V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130 byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.



Rung 2: On the 2nd scan, read the Gateway Address of the ECOM100 and store it in V3000 thru V3003 (4 decimal numbers). The ECOM100's Gateway Address could be displayed by an HMI.

The ECRDGWA is leading edge triggered, not power-flow driven (similar to a counter input leg). The command to read the Gateway Address will be sent to the ECOM100 whenever the power flow into the IBox goes from OFF to ON.

If successful, turn on C100. If there is a failure, turn on C101.



ECOM100 Read IP Address (ECRDIP) (IB-722)

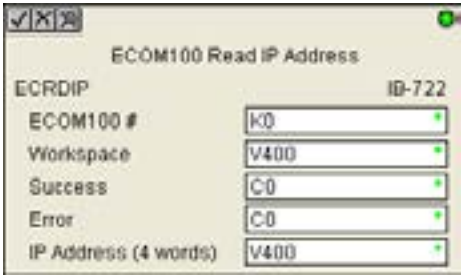
DS	Used
HPP	N/A

ECOM100 Read IP Address will read the 4 parts of the IP address and store them in 4 consecutive V-Memory locations in decimal format, on a leading edge transition to the IBox.

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Either the Success or Error bit parameter will turn on once the command is complete.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.



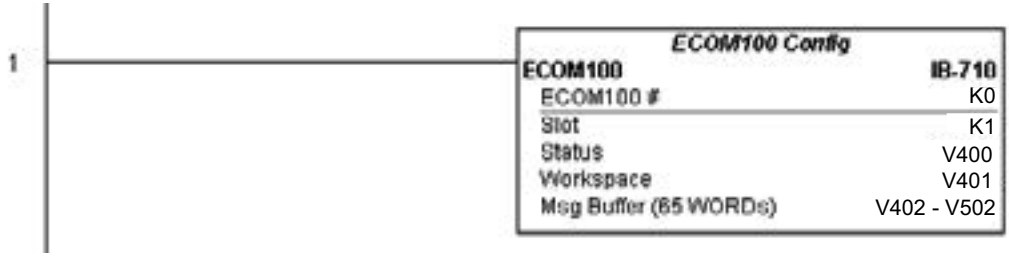
ECRDIP Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not successfully completed
- IP Address: specifies the starting address where the ECOM100's IP Address will be placed in 4 consecutive V-memory locations

Parameter		DL06 Range
ECOM100#	K	K0-255
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map
IP Address (4 Words)	V	See DL06 V-memory map - Data Words

ECRDIP Example

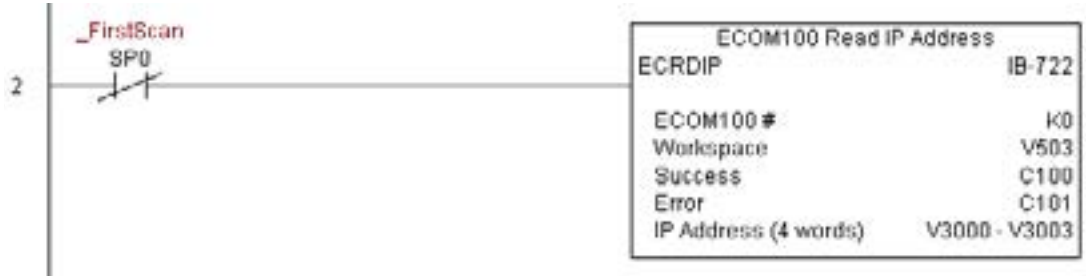
Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module.V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130 byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.



Rung 2: On the 2nd scan, read the IP Address of the ECOM100 and store it in V3000 thru V3003 (4 decimal numbers). The ECOM100's IP Address could be displayed by an HMI.

The ECRDIP is leading edge triggered, not power-flow driven (similar to a counter input leg). The command to read the IP Address will be sent to the ECOM100 whenever the power flow into the IBox goes from OFF to ON.

If successful, turn on C100. If there is a failure, turn on C101.



ECOM100 Read Module ID (ECRDMID) (IB-720)

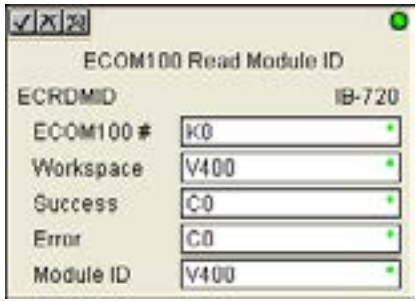
DS	Used
HPP	N/A

ECOM100 Read Module ID will read the binary (decimal) WORD sized Module ID on a leading edge transition to the IBox.

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Either the Success or Error bit parameter will turn on once the command is complete.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.



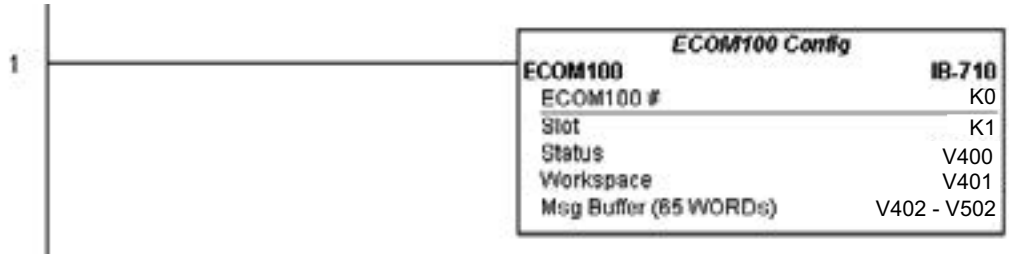
ECRDMID Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not successfully completed
- Module ID: specifies the location where the ECOM100's Module ID (decimal) will be placed

Parameter		DL06 Range
ECOM100#	K	K0-255
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map
Module ID	V	See DL06 V-memory map - Data Words

ECRDMID Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module. V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130 byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.



Rung 2: On the 2nd scan, read the Module ID of the ECOM100 and store it in V2000.

The ECRDMID is leading edge triggered, not power-flow driven (similar to a counter input leg). The command to read the module ID will be sent to the ECOM100 whenever the power flow into the IBox goes from OFF to ON.

If successful, turn on C100. If there is a failure, turn on C101.

ECOM100 Read Module Name (ECRDNAM) (IB-724)

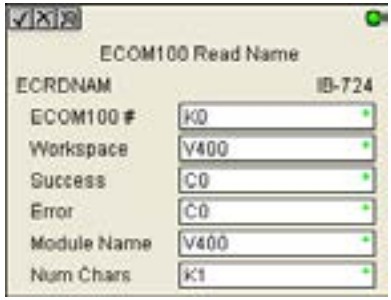
DS	Used
HPP	N/A

ECOM100 Read Name will read the Module Name up to the number of specified characters on a leading edge transition to the IBox.

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Either the Success or Error bit parameter will turn on once the command is complete.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.



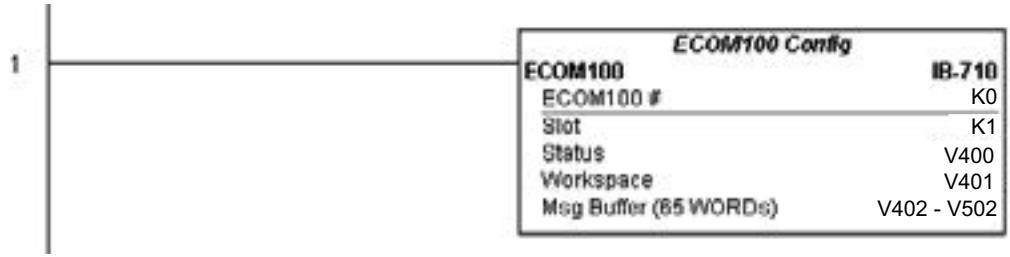
ECRDNAM Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not successfully completed
- Module Name: specifies the starting buffer location where the ECOM100's Module Name will be placed
- Num Chars: specifies the number of characters (bytes) to read from the ECOM100's Name field

Parameter		DL06 Range
ECOM100#	K	K0-255
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map
Module Name	V	See DL06 V-memory map - Data Words
Num Chars	K	K1-128

ECRDNAM Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module. V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130 byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.



Rung 2: On the 2nd scan, read the Module Name of the ECOM100 and store it in V3000 thru V3003 (8 characters). This text can be displayed by an HMI.

The ECRDNAM is leading edge triggered, not power-flow driven (similar to a counter input leg). The command to read the module name will be sent to the ECOM100 whenever the power flow into the IBox goes from OFF to ON.

If successful, turn on C100. If there is a failure, turn on C101.

ECOM100 Read Subnet Mask (ECRDSNM) (IB-732)

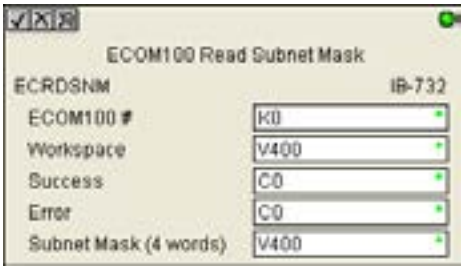
DS	Used
HPP	N/A

ECOM100 Read Subnet Mask will read the 4 parts of the Subnet Mask and store them in 4 consecutive V-Memory locations in decimal format, on a leading edge transition to the IBox.

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Either the Success or Error bit parameter will turn on once the command is complete.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.



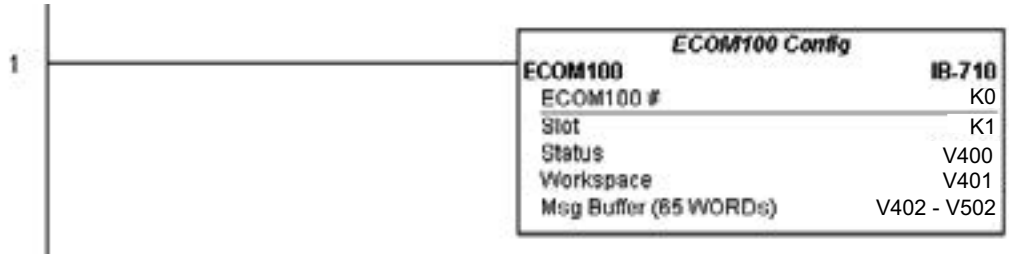
ECRDSNM Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not successfully completed
- Subnet Mask: specifies the starting address where the ECOM100's Subnet Mask will be placed in 4 consecutive V-memory locations

Parameter		DL06 Range
ECOM100#	K	K0-255
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map
Subnet Mask (4 Words)	V	See DL06 V-memory map - Data Words

ECRDSNM Example

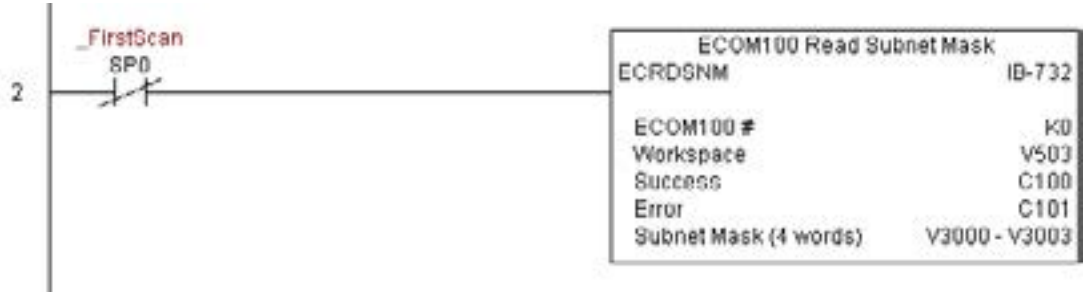
Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module.V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130 byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.



Rung 2: On the 2nd scan, read the Subnet Mask of the ECOM100 and store it in V3000 thru V3003 (4 decimal numbers). The ECOM100's Subnet Mask could be displayed by an HMI.

The ECRDSNM is leading edge triggered, not power-flow driven (similar to a counter input leg). The command to read the Subnet Mask will be sent to the ECOM100 whenever the power flow into the IBox goes from OFF to ON.

If successful, turn on C100. If there is a failure, turn on C101.



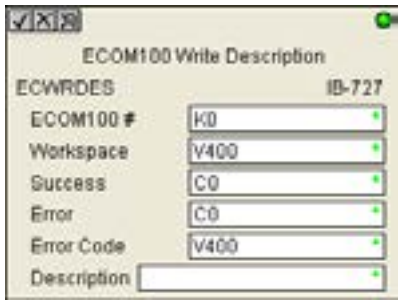
ECOM100 Write Description (ECWRDES) (IB-727)

DS	Used
HPP	N/A

ECOM100 Write Description will write the given Description to the ECOM100 module on a leading edge transition to the IBox. If you use a dollar sign ($) or double quote (“), use the PRINT/VPRINT escape sequence of TWO dollar signs ($$) for a single dollar sign or dollar sign-double quote ($”) for a double quote character.

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Either the Success or Error bit parameter will turn on once the command is complete. If there is an error, the Error Code parameter will report an ECOM100 error code (less than 100), or a PLC logic error (greater than 1000).



The Description is stored in Flash-ROM in the ECOM100 and the execution of this IBox will disable the ECOM100 module for at least a half second until it writes the Flash-ROM. Therefore, it is HIGHLY RECOMMENDED that you only execute this IBox ONCE on first scan. Since it requires a LEADING edge to execute, use a NORMALLY CLOSED SP0 (STR NOT First Scan) to drive the power flow to the IBox.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.

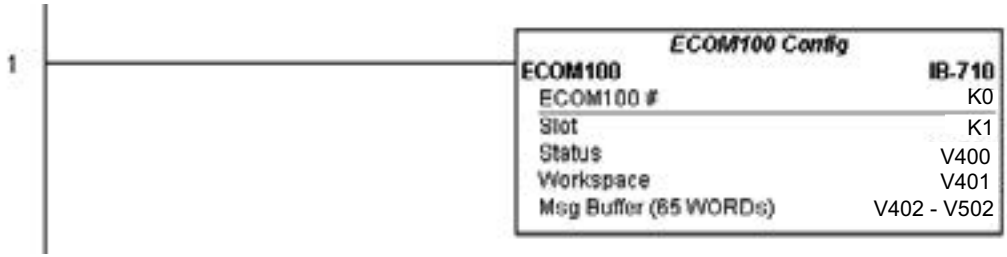
ECWRDES Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not successfully completed
- Error Code: specifies the location where the Error Code will be written
- Description: specifies the Description that will be written to the module

Parameter		DL06 Range
ECOM100#	K	K0-255
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map
Error Code	V	See DL06 V-memory map - Data Words
Description		Text

ECWRDES Example

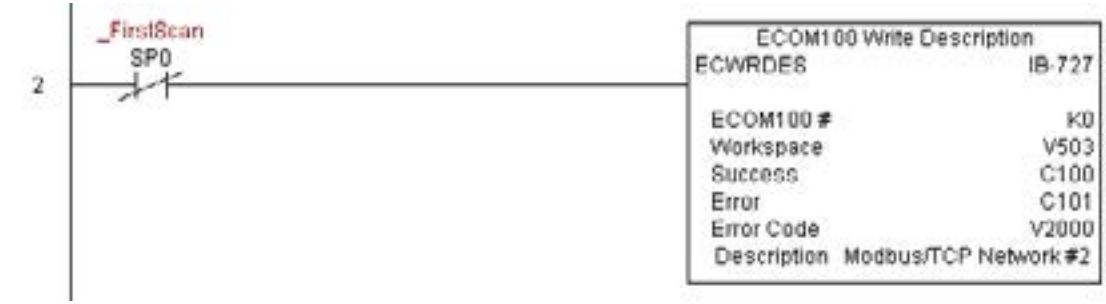
Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module.V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130 byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.



Rung 2: On the 2nd scan, set the Module Description of the ECOM100. Typically this is done using NetEdit, but this IBox allows you to configure the module description in the ECOM100 using your ladder program.

The EWRDES is leading edge triggered, not power-flow driven (similar to a counter input leg). The command to write the module description will be sent to the ECOM100 whenever the power flow into the IBox goes from OFF to ON.

If successful, turn on C100. If there is a failure, turn on C101. If it fails, you can look at V2000 for the specific error code.



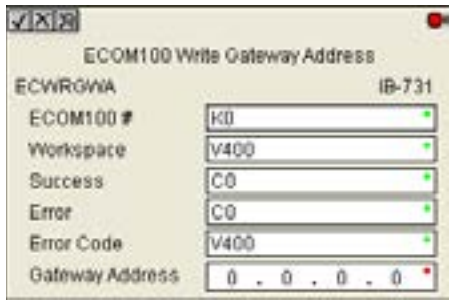
ECOM100 Write Gateway Address (ECWRGWA) (IB-731)

DS	Used
HPP	N/A

ECOM100 Write Gateway Address will write the given Gateway IP Address to the ECOM100 module on a leading edge transition to the IBox. See also ECOM100 IP Setup (ECIPSUP) IBox 717 to setup ALL of the TCP/IP parameters in a single instruction - IP Address, Subnet Mask, and Gateway Address.

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Either the Success or Error bit parameter will turn on once the command is complete. If there is an error, the Error Code parameter will report an ECOM100 error code (less than 100), or a PLC logic error (greater than 1000).



The Gateway Address is stored in Flash-ROM in the ECOM100 and the execution of this IBox will disable the ECOM100 module for at least a half second until it writes the Flash-ROM. Therefore, it is HIGHLY RECOMMENDED that you only execute this IBox ONCE, on first scan. Since it requires a LEADING edge to execute, use a NORMALLY CLOSED SP0 (STR NOT First Scan) to drive the power flow to the IBox.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.

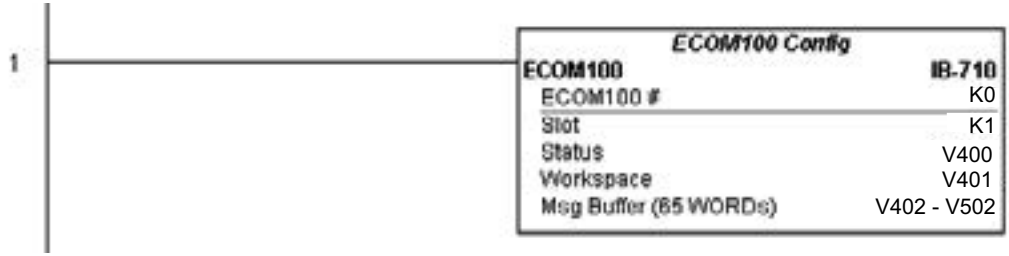
ECWRGWA Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not successfully completed
- Error Code: specifies the location where the Error Code will be written
- Gateway Address: specifies the Gateway IP Address that will be written to the module

Parameter		DL06 Range
ECOM100#	K	K0-255
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map
Error Code	V	See DL06 V-memory map - Data Words
Gateway Address		0.0.0.1. to 255.255.255.254

ECWRGWA Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module.V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130 byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.

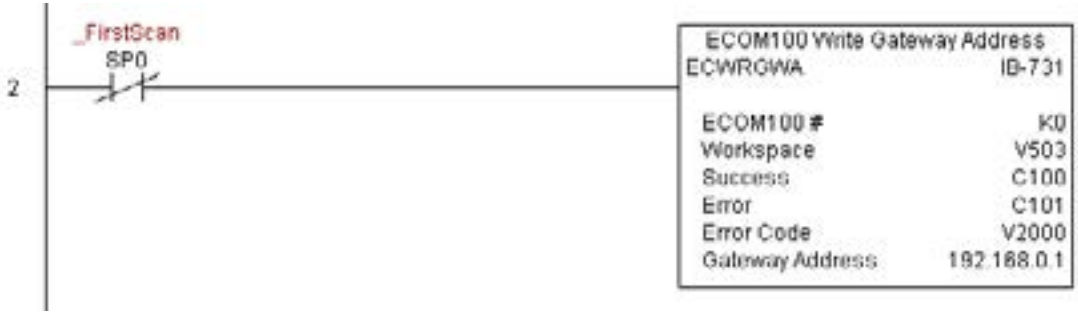


Rung 2: On the 2nd scan, assign the Gateway Address of the ECOM100 to 192.168.0.1

The ECWRGWA is leading edge triggered, not power-flow driven (similar to a counter input leg). The command to write the Gateway Address will be sent to the ECOM100 whenever the power flow into the IBox goes from OFF to ON.

If successful, turn on C100. If there is a failure, turn on C101. If it fails, you can look at V2000 for the specific error code.

To configure all of the ECOM100 TCP/IP parameters in one IBox, see the ECOM100 IP Setup (ECIPSUP) IBox.



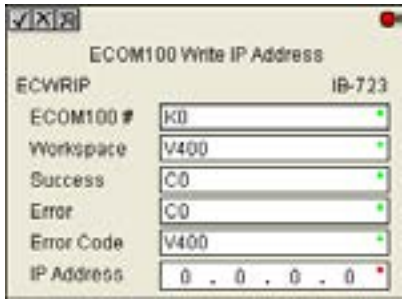
ECOM100 Write IP Address (ECWRIP) (IB-723)

DS	Used
HPP	N/A

ECOM100 Write IP Address will write the given IP Address to the ECOM100 module on a leading edge transition to the IBox. See also ECOM100 IP Setup (ECIPSUP) IBox 717 to setup ALL of the TCP/IP parameters in a single instruction - IP Address, Subnet Mask, and Gateway Address.

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Either the Success or Error bit parameter will turn on once the command is complete. If there is an error, the Error Code parameter will report an ECOM100 error code (less than 100), or a PLC logic error (greater than 1000).



The IP Address is stored in Flash-ROM in the ECOM100 and the execution of this IBox will disable the ECOM100 module for at least a half second until it writes the Flash-ROM. Therefore, it is HIGHLY RECOMMENDED that you only execute this IBox ONCE on first scan. Since it requires a LEADING edge to execute, use a NORMALLY CLOSED SP0 (STR NOT First Scan) to drive the power flow to the IBox.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.

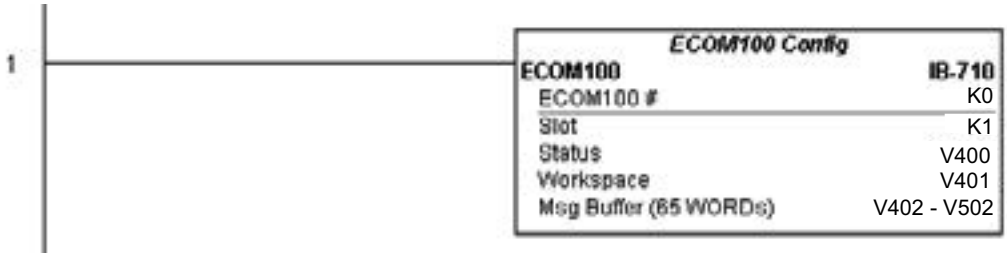
ECWRIP Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not successfully completed
- Error Code: specifies the location where the Error Code will be written
- IP Address: specifies the IP Address that will be written to the module

Parameter		DL06 Range
ECOM100#	K	K0-255
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map
Error Code	V	See DL06 V-memory map - Data Words
IP Address		0.0.0.1. to 255.255.255.254

ECWRIP Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module.V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130 byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.



Rung 2: On the 2nd scan, assign the IP Address of the ECOM100 to 192.168.12.100

The ECWRIP is leading edge triggered, not power-flow driven (similar to a counter input leg). The command to write the IP Address will be sent to the ECOM100 whenever the power flow into the IBox goes from OFF to ON.

If successful, turn on C100. If there is a failure, turn on C101. If it fails, you can look at V2000 for the specific error code.

To configure all of the ECOM100 TCP/IP parameters in one IBox, see the ECOM100 IP Setup (ECIPSUP) IBox.

ECOM100 Write Module ID (ECWRMID) (IB-721)

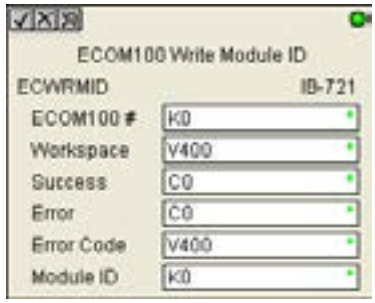
DS	Used
HPP	N/A

ECOM100 Write Module ID will write the given Module ID on a leading edge transition to the IBox

If the Module ID is set in the hardware using the dipswitches, this IBox will fail and return error code 1005 (decimal).

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Either the Success or Error bit parameter will turn on once the command is complete. If there is an error, the Error Code parameter will report an ECOM100 error code (less than 100), or a PLC logic error (greater than 1000).



The Module ID is stored in Flash-ROM in the ECOM100 and the execution of this IBox will disable the ECOM100 module for at least a half second until it writes the Flash-ROM. Therefore, it is HIGHLY RECOMMENDED that you only execute this IBox ONCE on first scan. Since it requires a LEADING edge to execute, use a NORMALLY CLOSED SP0 (STR NOT First Scan) to drive the power flow to the IBox.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.

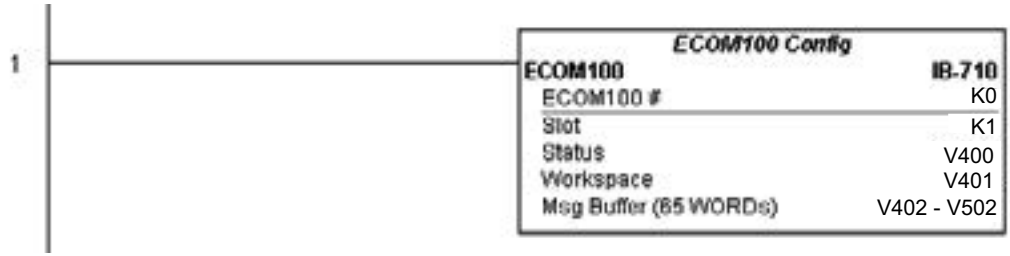
ECWRMID Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not successfully completed
- Error Code: specifies the location where the Error Code will be written
- Module ID: specifies the Module ID that will be written to the module

Parameter		DL06 Range
ECOM100#	K	K0-255
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map
Error Code	V	See DL06 V-memory map - Data Words
Module ID		K0-65535

ECWRMID Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module.V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130 byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.



Rung 2: On the 2nd scan, set the Module ID of the ECOM100. Typically this is done using NetEdit, but this IBox allows you to configure the module ID of the ECOM100 using your ladder program.

The EWRMID is leading edge triggered, not power-flow driven (similar to a counter input leg). The command to write the module ID will be sent to the ECOM100 whenever the power flow into the IBox goes from OFF to ON.

If successful, turn on C100. If there is a failure, turn on C101. If it fails, you can look at V2000 for the specific error code.

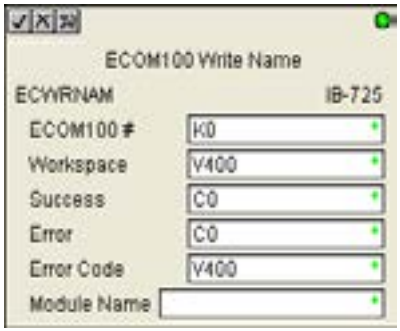
ECOM100 Write Name (ECWRNAM) (IB-725)

DS	Used
HPP	N/A

ECOM100 Write Name will write the given Name to the ECOM100 module on a leading edge transition to the IBox. If you use a dollar sign ($) or double quote (“), use the PRINT/VPRINT escape sequence of TWO dollar signs ($$) for a single dollar sign or dollar sign-double quote ($”) for a double quote character.

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Either the Success or Error bit parameter will turn on once the command is complete. If there is an error, the Error Code parameter will report an ECOM100 error code (less than 100), or a PLC logic error (greater than 1000).



The Name is stored in Flash-ROM in the ECOM100 and the execution of this IBox will disable the ECOM100 module for at least a half second until it writes the Flash-ROM. Therefore, it is HIGHLY RECOMMENDED that you only execute this IBox ONCE on first scan. Since it requires a LEADING edge to execute, use a NORMALLY CLOSED SP0 (STR NOT First Scan) to drive the power flow to the IBox.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.

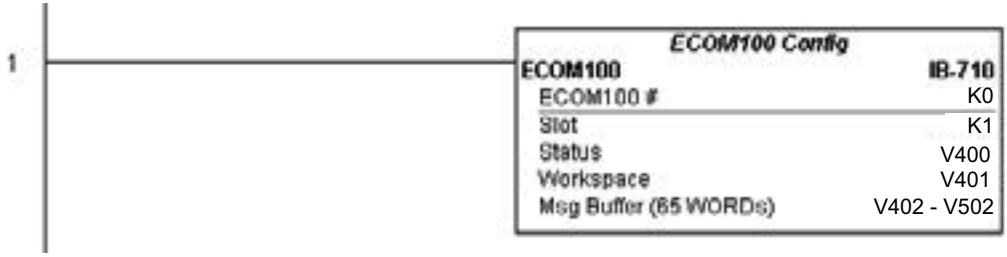
ECWRNAM Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not successfully completed
- Error Code: specifies the location where the Error Code will be written
- Module Name: specifies the Name that will be written to the module

Parameter		DL06 Range
ECOM100#	K	K0-255
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map
Error Code	V	See DL06 V-memory map - Data Words
Module Name		Text

ECWRNAM Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module.V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130 byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.



Rung 2: On the 2nd scan, set the Module Name of the ECOM100. Typically this is done using NetEdit, but this IBox allows you to configure the module name of the ECOM100 using your ladder program.

The EWRNAM is leading edge triggered, not power-flow driven (similar to a counter input leg). The command to write the module name will be sent to the ECOM100 whenever the power flow into the IBox goes from OFF to ON.

If successful, turn on C100. If there is a failure, turn on C101. If it fails, you can look at V2000 for the specific error code.

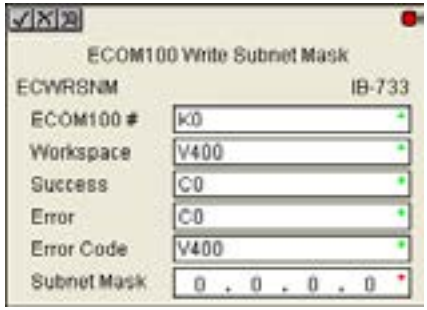
ECOM100 Write Subnet Mask (ECWRSNM) (IB-733)

DS	Used
HPP	N/A

ECOM100 Write Subnet Mask will write the given Subnet Mask to the ECOM100 module on a leading edge transition to the IBox. See also ECOM100 IP Setup (ECIPSUP) IBox 717 to setup ALL of the TCP/IP parameters in a single instruction - IP Address, Subnet Mask, and Gateway Address.

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Either the Success or Error bit parameter will turn on once the command is complete. If there is an error, the Error Code parameter will report an ECOM100 error code (less than 100), or a PLC logic error (greater than 1000).



The Subnet Mask is stored in Flash-ROM in the ECOM100 and the execution of this IBox will disable the ECOM100 module for at least a half second until it writes the Flash-ROM. Therefore, it is HIGHLY RECOMMENDED that you only execute this IBox ONCE on first scan. Since it requires a LEADING edge to execute, use a NORMALLY CLOSED SP0 (STR NOT First Scan) to drive the power flow to the IBox.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.

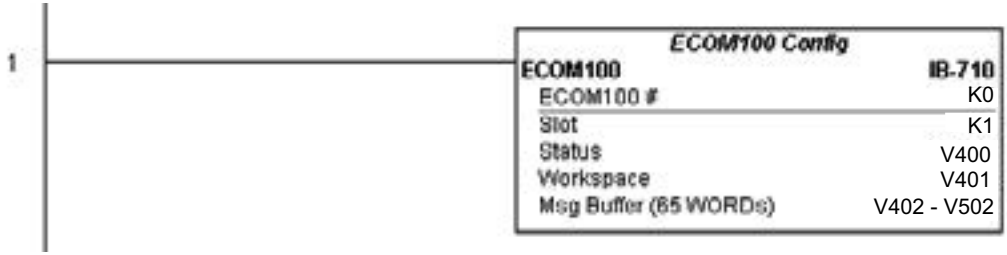
ECWRSNM Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not successfully completed
- Error Code: specifies the location where the Error Code will be written
- Subnet Mask: specifies the Subnet Mask that will be written to the module

Parameter		DL06 Range
ECOM100#	K	K0-255
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map
Error Code	V	See DL06 V-memory map - Data Words
Subnet Mask		Masked IP Address

ECWRSNM Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module.V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130 byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.



Rung 2: On the 2nd scan, assign the Subnet Mask of the ECOM100 to 255.255.0.0

The ECWRSNM is leading edge triggered, not power-flow driven (similar to a counter input leg). The command to write the Subnet Mask will be sent to the ECOM100 whenever the power flow into the IBox goes from OFF to ON.

If successful, turn on C100. If there is a failure, turn on C101. If it fails, you can look at V2000 for the specific error code.

To configure all of the ECOM100 TCP/IP parameters in one IBox, see the ECOM100 IP Setup (ECIPSUP) IBox.

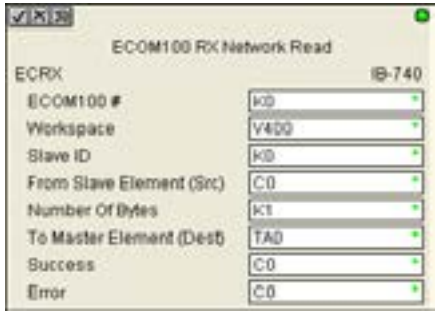
ECOM100 RX Network Read (ECRX) (IB-740)

DS	Used
HPP	N/A

ECOM100 RX Network Read performs the RX instruction with built-in interlocking with all other ECOM100 RX (ECRX) and ECOM100 WX (ECWX) IBoxes in your program to simplify communications networking. It will perform the RX on the specified ECOM100#'s network, which corresponds to a specific unique ECOM100 Configuration (ECOM100) IBox at the top of your program.

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Whenever this IBox has power, it will read element data from the specified slave into the given destination V-Memory buffer, giving other ECOM100 RX and ECOM100 WX IBoxes on that ECOM100# network a chance to execute.



For example, if you wish to read and write data continuously from 5 different slaves, you can have all of these ECRX and ECWX instructions in ONE RUNG driven by SP1 (Always On). They will execute round-robin style, automatically.

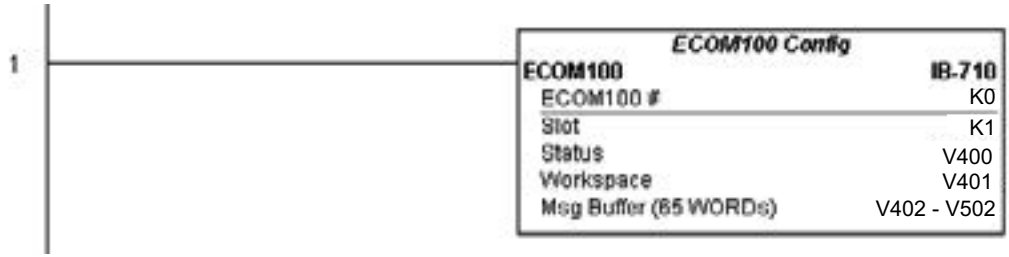
ECRX Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Workspace: specifies a V-memory location that will be used by the instruction
- Slave ID: specifies the slave ECOM(100) PLC that will be targeted by the ECRX instruction
- From Slave Element (Src): specifies the slave address of the data to be read
- Number of Bytes: specifies the number of bytes to read from the slave ECOM(100) PLC
- To Master Element (Dest): specifies the location where the slave data will be placed in the master ECOM100 PLC
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not successfully completed

Parameter		DL06 Range
ECOM100#	K	K0-255
Workspace	V	See DL06 V-memory map - Data Words
Slave ID	K	K0-90
From Slave Element (Src)	X,Y,C,S,T,CT,GX,GY,V,P	See DL06 V-memory map
Number of Bytes	K	K1-128
To Master Element (Dest)	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map

ECRX Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module.V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130 byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.



(example continued on next page)

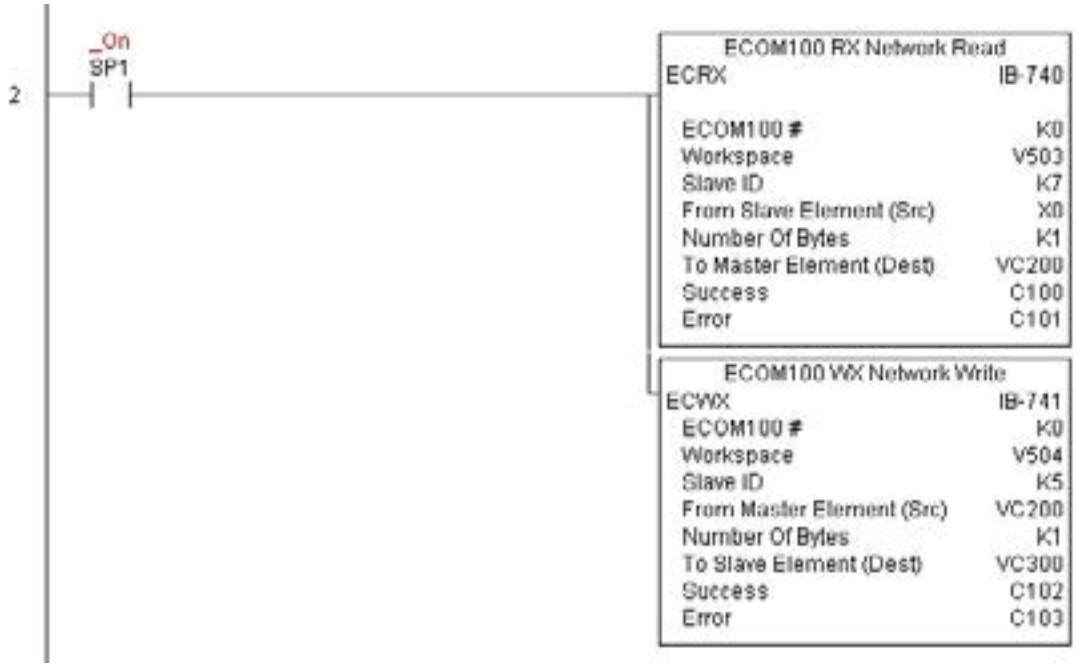
ECRX Example (cont'd)

Rung 2: Using ECOM100# K0, read X0-X7 from Slave K7 and write them to slave K5 as fast as possible. Store them in this local PLC in C200-C207, and write them to C300-C307 in slave K5.

Both the ECRX and ECWX work with the ECOM100 Config IBox to simplify all networking by handling all of the interlocks and proper resource sharing. They also provide very simplified error reporting. You no longer need to worry about any SP "busy bits" or "error bits", or what slot number a module is in, or have any counters or shift registers or any other interlocks for resource management.

In this example, SP1 (always ON) is driving both the ECRX and ECWX IBoxes in the same rung. On the scan that the Network Read completes, the Network Write will start that same scan. As soon as the Network Write completes, any pending operations below it in the program would get a turn. If there are no pending ECOM100 IBoxes below the ECWX, then the very next scan the ECRX would start its request again.

Using the ECRX and ECWX for all of your ECOM100 network reads and writes is the fastest the PLC can do networking. For local Serial Ports, DCM modules, or the original ECOM modules, use the NETCFG and NETRX/NETWX IBoxes.



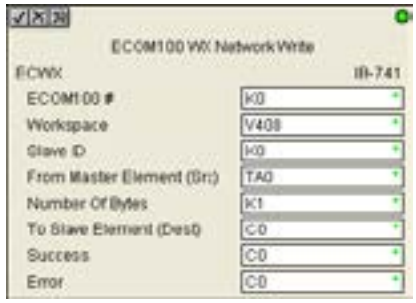
ECOM100 WX Network Write(ECWX) (IB-741)

DS	Used
HPP	N/A

ECOM100 WX Network Write performs the WX instruction with built-in interlocking with all other ECOM100 RX (ECRX) and ECOM100 WX (ECWX) IBoxes in your program to simplify communications networking. It will perform the WX on the specified ECOM100#'s network, which corresponds to a specific unique ECOM100 Configuration (ECOM100) IBox at the top of your program.

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Whenever this IBox has power, it will write data from the master's V-Memory buffer to the specified slave starting with the given slave element, giving other ECOM100 RX and ECOM100 WX IBoxes on that ECOM100# network a chance to execute.



For example, if you wish to read and write data continuously from 5 different slaves, you can have all of these ECRX and ECWX instructions in ONE RUNG driven by SP1 (Always On). They will execute round-robin style, automatically.

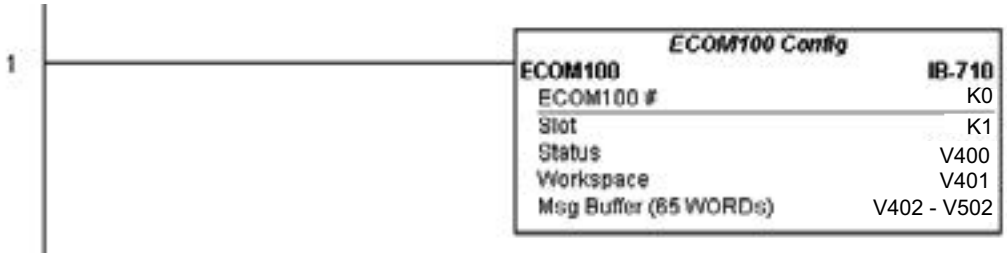
ECWX Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Workspace: specifies a V-memory location that will be used by the instruction
- Slave ID: specifies the slave ECOM(100) PLC that will be targeted by the ECWX instruction
- From Master Element (Src): specifies the location in the master ECOM100 PLC where the data will be sourced from
- Number of Bytes: specifies the number of bytes to write to the slave ECOM(100) PLC
- To Slave Element (Dest): specifies the slave address the data will be written to
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not successfully completed

Parameter		DL06 Range
ECOM100#	K	K0-255
Workspace	V	See DL06 V-memory map - Data Words
Slave ID	K	K0-90
From Master Element (Src)	V	See DL06 V-memory map - Data Words
Number of Bytes	K	K1-128
To Slave Element (Dest)	X,Y,C,S,T,CT,GX,GY,V,P	See DL06 V-memory map
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map

ECWX Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module.V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130 byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.



(example continued on next page)

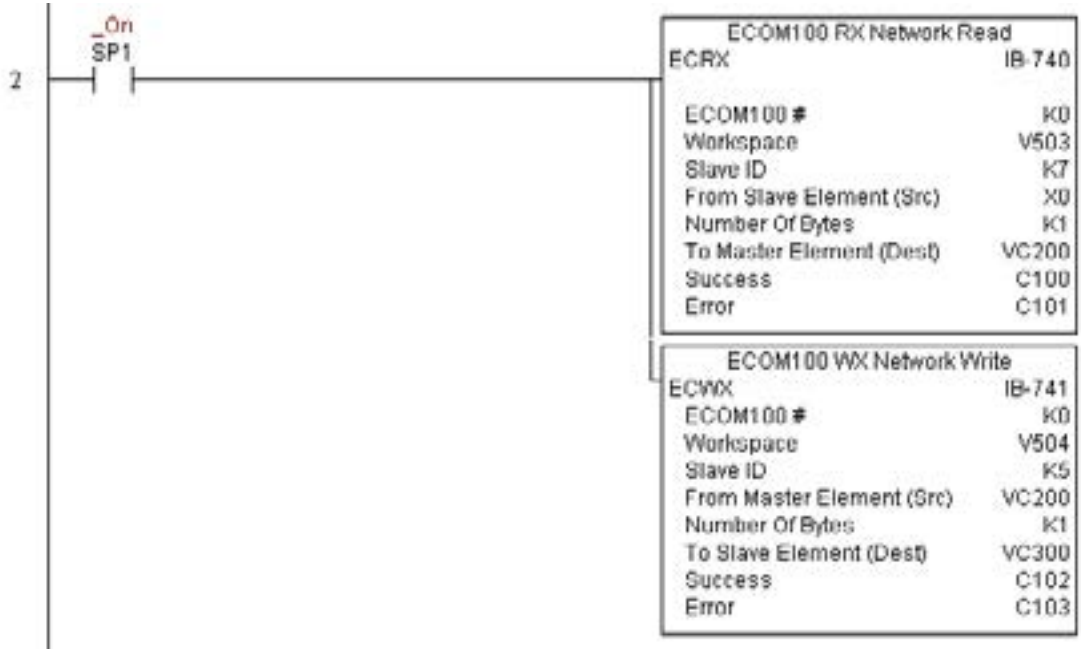
ECWX Example (cont'd)

Rung 2: Using ECOM100# K0, read X0-X7 from Slave K7 and write them to slave K5 as fast as possible. Store them in this local PLC in C200-C207, and write them to C300-C307 in slave K5.

Both the ECRX and ECWX work with the ECOM100 Config IBox to simplify all networking by handling all of the interlocks and proper resource sharing. They also provide very simplified error reporting. You no longer need to worry about any SP "busy bits" or "error bits", or what slot number a module is in, or have any counters or shift registers or any other interlocks for resource management.

In this example, SP1 (always ON) is driving both the ECRX and ECWX IBoxes in the same rung. On the scan that the Network Read completes, the Network Write will start that same scan. As soon as the Network Write completes, any pending operations below it in the program would get a turn. If there are no pending ECOM100 IBoxes below the ECWX, then the very next scan the ECRX would start its request again.

Using the ECRX and ECWX for all of your ECOM100 network reads and writes is the fastest the PLC can do networking. For local Serial Ports, DCM modules, or the original ECOM modules, use the NETCFG and NETRX/NETWX IBoxes.



NETCFG Network Configuration (NETCFG) (IB-700)

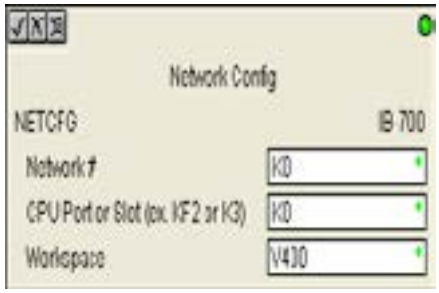
DS	Used
HPP	N/A

Network Config defines all the common information necessary for performing RX/WX Networking using the NETRX and NETWX IBox instructions via a local CPU serial port, DCM or ECOM module.

You must have the Network Config instruction at the top of your ladder/stage program with any other configuration IBoxes.

If you use more than one local serial port, DCM or ECOM in your PLC for RX/WX Networking, you must have a different Network Config instruction for EACH RX/WX network in your system that utilizes any NETRX/NETWX IBox instructions.

The Workspace parameter is an internal, private register used by the Network Config IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.



The 2nd parameter "CPU Port or Slot" is the same value as in the high byte of the first LD instruction if you were coding the RX or WX rung yourself. This value is CPU and port specific (check your PLC manual). Use KF2 for the DL06 CPU serial port 2. If using a DCM or ECOM module, use Kx, where x equals the slot where the module is installed.

Since this logic only executes on the first scan, this IBox cannot have any input logic.

NETCFG Parameters

- Network#: specifies a unique # for each ECOM(100) or DCM network to use
- CPU Port or Slot: specifies the CPU port number or slot number of DCM/ECOM(100) used
- Workspace: specifies a V-memory location that will be used by the instruction

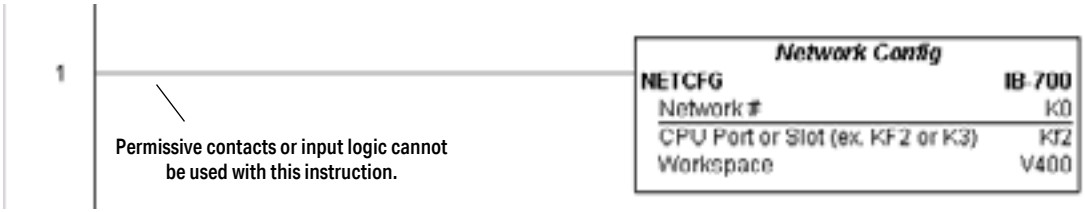
Parameter		DL06 Range
Network#	K	K0-255
CPU Port or Slot	K	K0-FF
Workspace	V	See DL06 V-memory map - Data Words

NETCFG Example

The Network Configuration IBox coordinates all of the interaction with other Network IBoxes (NETRX/NETWX). You must have a Network Configuration IBox for each serial port network, DCM module network, or original ECOM module network in your system. Configuration IBoxes must be at the top of your program and must execute every scan.

This IBox defines Network# K0 to be for the local CPU serial port #2 (KF2). For local CPU serial ports or DCM/ECOM modules, use the same value you would use in the most significant byte of the first LD instruction in a normal RX/WX rung to reference the port or module. Any NETRX or NETWX IBoxes that need to reference this specific network would enter K0 for their Network# parameter.

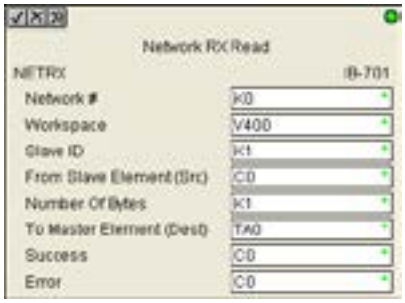
The Workspace register is used to maintain state information about the port or module, along with proper sharing and interlocking with the other NETRX and NETWX IBoxes in the program. This V-memory register must not be used anywhere else in the entire program.



Network RX Read (NETRX) (IB-701)

DS	Used
HPP	N/A

Network RX Read performs the RX instruction with built-in interlocking with all other Network RX (NETRX) and Network WX (NETWX) IBoxes in your program to simplify communications networking. It will perform the RX on the specified Network #, which corresponds to a specific unique Network Configuration (NETCFG) at the top of your program.



The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Whenever this IBox has power, it will read element data from the specified slave into the given destination V-Memory buffer, giving other Network RX and Network WX IBoxes on that Network # a chance to execute.

For example, if you wish to read and write data continuously from 5 different slaves, you can have all of these NETRX and NETWX instructions in ONE RUNG driven by SP1 (Always On). They will execute round-robin style, automatically.

NETRX Parameters

- Network#: specifies the (CPU port's, DCM's, ECOM's) Network # defined by the NETCFG instruction
- Workspace: specifies a V-memory location that will be used by the instruction
- Slave ID: specifies the slave PLC that will be targeted by the NETRX instruction
- From Slave Element (Src): specifies the slave address of the data to be read
- Number of Bytes: specifies the number of bytes to read from the slave device
- To Master Element (Dest): specifies the location where the slave data will be placed in the master PLC
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not successfully completed

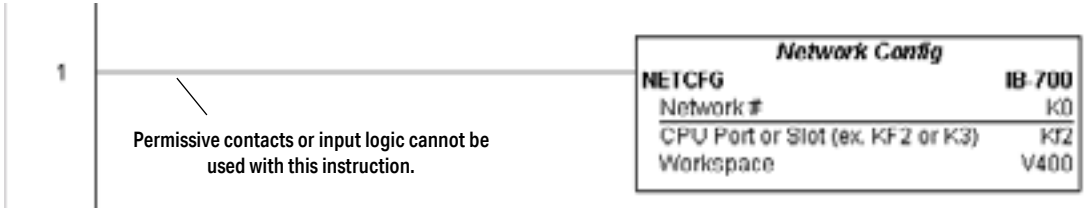
Parameter		DL06 Range
Network#	K	K0-255
Workspace	V	See DL06 V-memory map - Data Words
Slave ID	K, V	K0-90: See DL06 V-memory map
From Slave Element (Src)	X,Y,C,S,T,CT,GX,GY,V,P	See DL06 V-memory map
Number of Bytes	K	K1-128
To Master Element (Dest)	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map

NETRX Example

Rung 1: The Network Configuration IBox coordinates all of the interaction with other Network IBoxes (NETRX/NETWX). You must have a Network Configuration IBox for each serial port network, DCM module network, or original ECOM module network in your system. Configuration IBoxes must be at the top of your program and must execute every scan.

This IBox defines Network# K0 to be for the local CPU serial port #2 (KF2). For local CPU serial ports or DCM/ECOM modules, use the same value you would use in the most significant byte of the first LD instruction in a normal RX/WX rung to reference the port or module. Any NETRX or NETWX IBoxes that need to reference this specific network would enter K0 for their Network# parameter.

The Workspace register is used to maintain state information about the port or module, along with proper sharing and interlocking with the other NETRX and NETWX IBoxes in the program. This V-Memory register must not be used anywhere else in the entire program.



(example continued on next page)

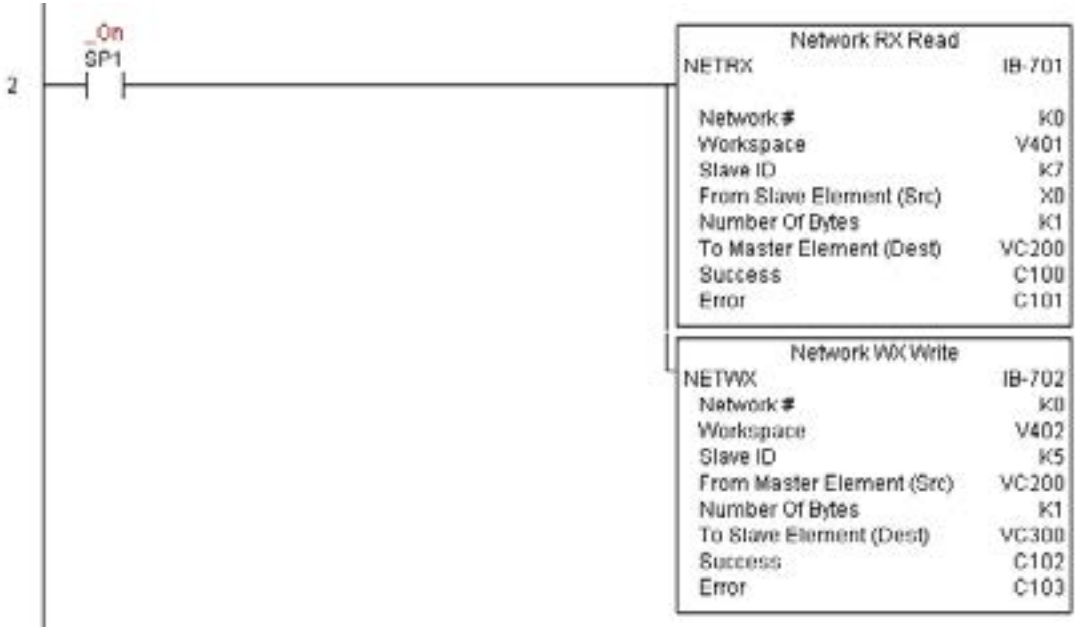
NETRX Example (cont'd)

Rung 2: Using Network# K0, read X0-X7 from Slave K7 and write them to slave K5 as fast as possible. Store them in this local PLC in C200-C207, and write them to C300-C307 in slave K5.

Both the NETRX and NETWX work with the Network Config IBox to simplify all networking by handling all of the interlocks and proper resource sharing. They also provide very simplified error reporting. You no longer need to worry about any SP "busy bits" or "error bits", or what port number or slot number a module is in, or have any counters or shift registers or any other interlocks for resource management.

In this example, SP1 (always ON) is driving both the NETRX and NETWX IBoxes in the same rung. On the scan that the Network Read completes, the Network Write will start that same scan. As soon as the Network Write completes, any pending operations below it in the program would get a turn. If there are no pending NETRX or NETWX IBoxes below this IBox, then the very next scan the NETRX would start its request again.

Using the NETRX and NETWX for all of your serial port, DCM, or original ECOM network reads and writes is the fastest the PLC can do networking. For ECOM100 modules, use the ECOM100 and ECRX/ECWX IBoxes.



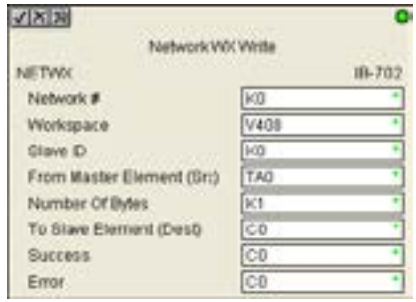
Network WX Write (NETWX) (IB-702)

DS	Used
HPP	N/A

Network WX Write performs the WX instruction with built-in interlocking with all other Network RX (NETRX) and Network WX (NETWX) IBoxes in your program to simplify communications networking. It will perform the WX on the specified Network #, which corresponds to a specific unique Network Configuration (NETCFG) at the top of your program.

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Whenever this IBox has power, it will write data from the master's V-Memory buffer to the specified slave starting with the given slave element, giving other Network RX and Network WX IBoxes on that Network # a chance to execute.



For example, if you wish to read and write data continuously from 5 different slaves, you can have all of these NETRX and NETWX instructions in ONE RUNG driven by SP1 (Always On). They will execute round-robin style, automatically.

NETWX Parameters

- Network#: specifies the (CPU port's, DCM's, ECOM's) Network # defined by the NETCFG instruction
- Workspace: specifies a V-memory location that will be used by the instruction
- Slave ID: specifies the slave PLC that will be targeted by the NETWX instruction
- From Master Element (Src): specifies the location in the master PLC where the data will be sourced from
- Number of Bytes: specifies the number of bytes to write to the slave PLC
- To Slave Element (Dest): specifies the slave address the data will be written to
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not successfully completed

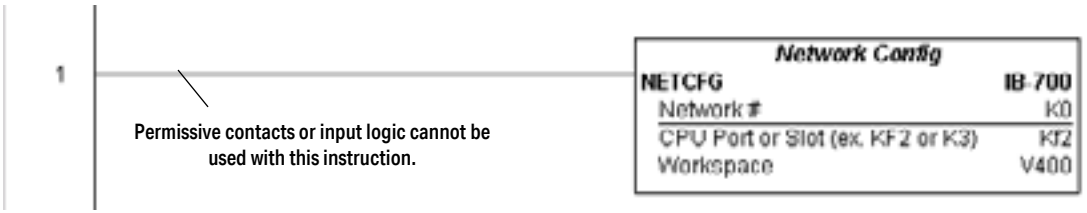
Parameter		DL06 Range
Network#	K	K0-255
Workspace	V	See DL06 V-memory map - Data Words
Slave ID	K,V	K0-90: See DL06 V-memory map
From Master Element (Src)	V	See DL06 V-memory map - Data Words
Number of Bytes	K	K1-128
To Slave Element (Dest)	X,Y,C,S,T,CT,GX,GY,V,P	See DL06 V-memory map
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map

NETWX Example

Rung 1: The Network Configuration IBox coordinates all of the interaction with other Network IBoxes (NETRX/NETWX). You must have a Network Configuration IBox for each serial port network, DCM module network, or original ECOM module network in your system. Configuration IBoxes must be at the top of your program and must execute every scan.

This IBox defines Network# K0 to be for the local CPU serial port #2 (KF2). For local CPU serial ports or DCM/ECOM modules, use the same value you would use in the most significant byte of the first LD instruction in a normal RX/WX rung to reference the port or module. Any NETRX or NETWX IBoxes that need to reference this specific network would enter K0 for their Network# parameter.

The Workspace register is used to maintain state information about the port or module, along with proper sharing and interlocking with the other NETRX and NETWX IBoxes in the program. This V-Memory register must not be used anywhere else in the entire program.



(example continued on next page)

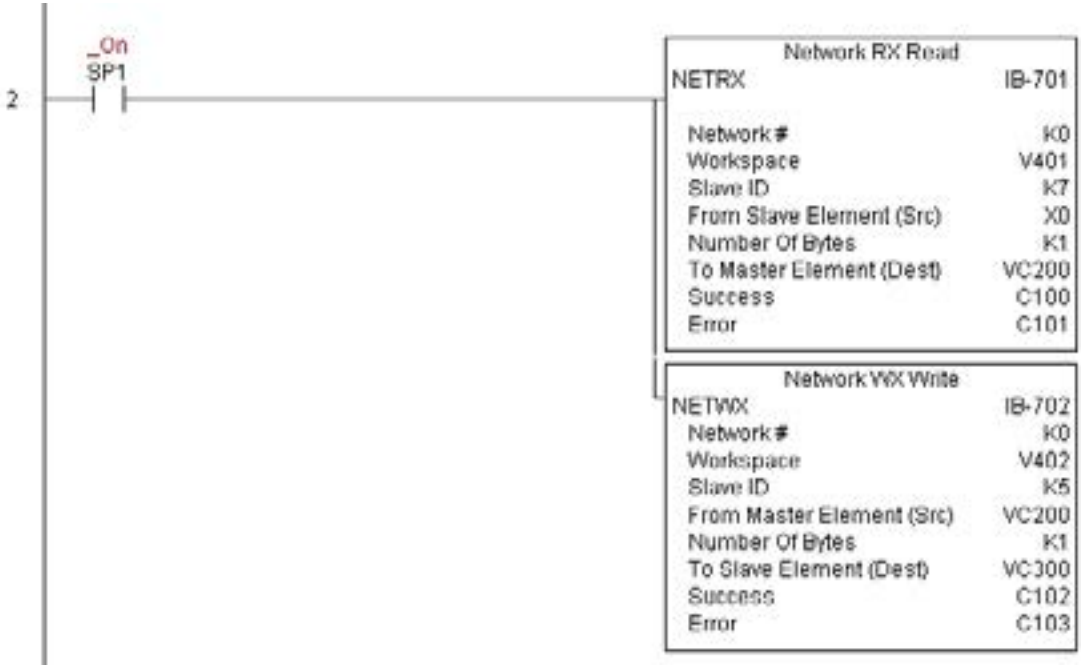
NETWX Example (cont'd)

Rung 2: Using Network# K0, read X0-X7 from Slave K7 and write them to slave K5 as fast as possible. Store them in this local PLC in C200-C207, and write them to C300-C307 in slave K5.

Both the NETRX and NETWX work with the Network Config IBox to simplify all networking by handling all of the interlocks and proper resource sharing. They also provide very simplified error reporting. You no longer need to worry about any SP "busy bits" or "error bits", or what port number or slot number a module is in, or have any counters or shift registers or any other interlocks for resource management.

In this example, SP1 (always ON) is driving both the NETRX and NETWX IBoxes in the same rung. On the scan that the Network Read completes, the Network Write will start that same scan. As soon as the Network Write completes, any pending operations below it in the program would get a turn. If there are no pending NETRX or NETWX IBoxes below this IBox, then the very next scan the NETRX would start its request again.

Using the NETRX and NETWX for all of your serial port, DCM, or original ECOM network reads and writes is the fastest the PLC can do networking. For ECOM100 modules, use the ECOM100 and ECRX/ECWX IBoxes.

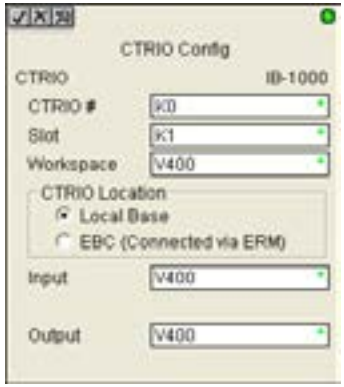


CTRIO Configuration (CTRIO) (IB-1000)

DS	Used
HPP	N/A

CTRIO Config defines all the common information for one specific CTRIO module which is used by the other CTRIO IBox instructions (for example, CTRLDPR - CTRIO Load Profile, CTREDRL - CTRIO Edit and Reload Preset Table, CTRRTLM - CTRIO Run to Limit Mode, ...).

The Input/Output parameters for this instruction can be copied directly from the CTRIO Workbench configuration for this CTRIO module. Since the behavior is slightly different when the CTRIO module is in an EBC Base via an ERM, you must specify whether the CTRIO module is in a local base or in an EBC base. The DL06 PLC only supports local base operation at this time.



You must have the CTRIO Config IBox at the top of your ladder/stage program along with any other configuration IBoxes.

If you have more than one CTRIO in your PLC, you must have a different CTRIO Config IBox for EACH CTRIO module in your system that utilizes any CTRIO IBox instructions. Each CTRIO Config IBox must have a UNIQUE CTRIO# value. This is how the CTRIO IBoxes differentiate between the different CTRIO modules in your system.

The Workspace parameter is an internal, private register used by the CTRIO Config IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Since this logic only executes on the first scan, this IBox cannot have any input logic.

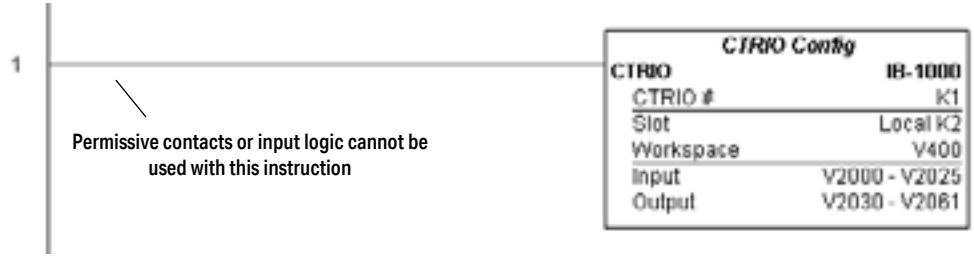
CTRIO Parameters

- CTRIO#: specifies a specific CTRIO module based on a user defined number
- Slot: specifies which PLC option slot the CTRIO module occupies
- Workspace: specifies a V-memory location that will be used by the instruction
- CTRIO Location: specifies where the module is located (local base only for DL06)
- Input: This needs to be set to the same V-memory register as is specified in CTRIO Workbench as 'Starting V address for inputs' for this unique CTRIO.
- Output: This needs to be set to the same V-memory register as is specified in CTRIO Workbench as 'Starting V address for outputs' for this unique CTRIO.

Parameter		DL06 Range
CTRIO#	K	K0-255
Slot	K	K1-4
Workspace	V	See DL06 V-memory map - Data Words
Input	V	See DL06 V-memory map - Data Words
Output	V	See DL06 V-memory map - Data Words

CTRIO Example

Rung 1: This sets up the CTRIO card in slot 2 of the local base. Each CTRIO in the system will need a separate CTRIO I-box before any CTRxxxx I-boxes can be used for them. The CTRIO has been configured to use V2000 through V2025 for its input data, and V2030 through V2061 for its output data.



CTRIO Add Entry to End of Preset Table (CTRADPT) (IB-1005)

DS	Used
HPP	N/A

CTRIO Add Entry to End of Preset Table, on a leading edge transition to this IBox, will append an entry to the end of a memory based Preset Table on a specific CTRIO Output resource. This IBox will take more than 1 PLC scan to execute. Either the Success or Error bit will turn on when the command is complete. If the Error Bit is on, you can use the CTRIO Read Error Code (CTRRDER) IBox to get extended error information.

Entry Type:

K0: Set

K1: Reset

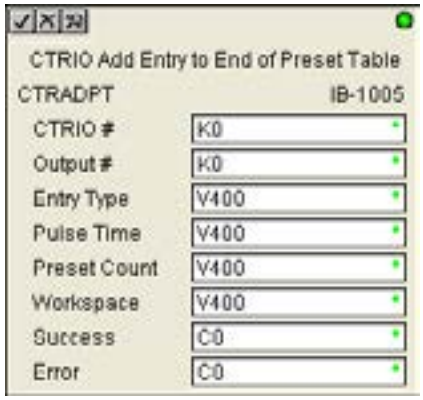
K2: Pulse On (uses Pulse Time)

K3: Pulse Off (uses Pulse Time)

K4: Toggle

K5: Reset Count

Note that the Pulse Time parameter is ignored by some Entry Types.



The Workspace register is for internal use by this IBox instruction and MUST NOT be used anywhere else in your program.

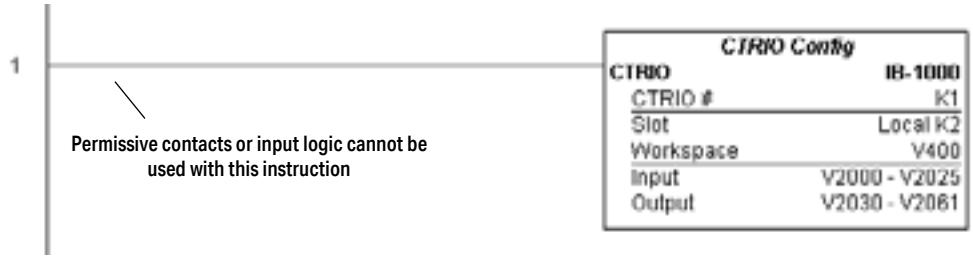
CTRAPT Parameters

- CTRIO#: specifies a specific CTRIO module based on a user defined number (see CTRIO Config)
- Output#: specifies a CTRIO output to be used by the instruction
- Entry Type: specifies the Entry Type to be added to the end of a Preset Table
- Pulse Time: specifies a pulse time for the Pulse On and Pulse Off Entry Types
- Preset Count: specifies an initial count value to begin at after Reset
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the instruction has successfully completed
- Error: specifies a bit that will turn on if the instruction does not complete successfully

Parameter		DL06 Range
CTRIO#	K	K0-255
Output#	K	K0-3
Entry Type	V,K	K0-5; See DL06 V-memory map - Data Words
Pulse Time	V,K	K0-65535; See DL06 V-memory map - Data Words
Preset Count	V,K	K0-2147434528; See DL06 V-memory map
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map

CTRADPT Example

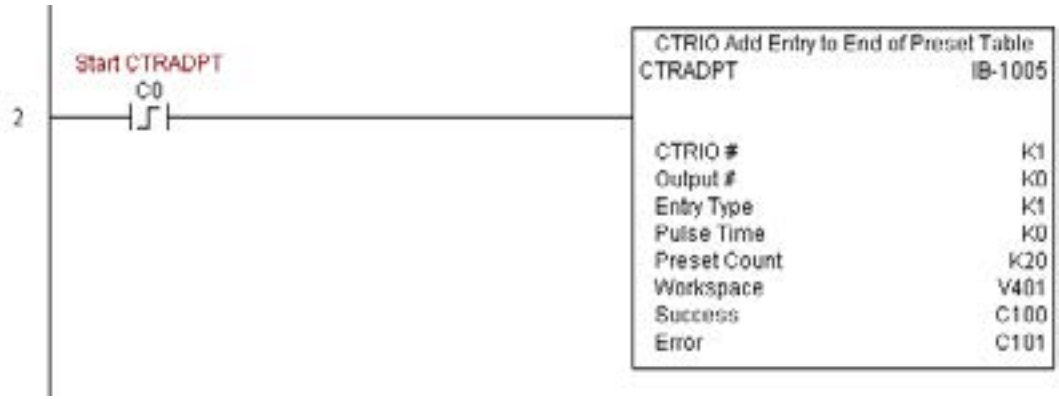
Rung 1: This sets up the CTRIO card in slot 2 of the local base. Each CTRIO in the system will need a separate CTRIO I-box before any CTRxxxx I-boxes can be used for them. The CTRIO has been configured to use V2000 through V2025 for its input data, and V2030 through V2061 for its output data.



Rung 2: This rung is a sample method for enabling the CTRADPT command. A C-bit is used to allow the programmer to control the command from Data View for testing purposes.

Turning on C0 will cause the CTRADPT instruction to add a new preset to the preset table for output #0 on the CTRIO in slot 2. The new preset will be a command to RESET (entry type K1=reset), pulse time is left at zero as the reset type does not use this, and the count at which it will reset will be 20.

Operating procedure for this example code is to load the CTRADPT_ex1.cwb file to your CTRIO, then enter the code shown here, change to RUN mode, enable output #0 by turning on C2 in Data View, turn encoder on CTRIO to value above 10 and output #0 light will come on and stay on for all counts past 10. Now reset the counter with C1, enable C0 to execute CTRADPT command to add a reset for output #0 at a count of 20, turn on C2 to enable output #0, then turn encoder to value of 10+ (output #0 should turn on) and then continue on to count of 20+ (output #0 should turn off).



(example continued on next page)

CTRADPT Example (cont'd)

Rung 3: This rung allows the programmer to reset the counter from the ladder logic.

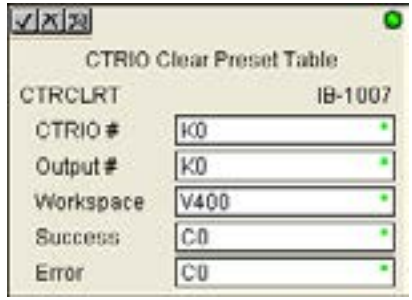
Rung 4: This rung allows the operator to enable output #0 from the ladder code.

CTRIO Clear Preset Table (CTRCLRT) (IB-1007)

DS	Used
HPP	N/A

CTRIO Clear Preset Table will clear the RAM based Preset Table on a leading edge transition to this IBox. This IBox will take more than 1 PLC scan to execute. Either the Success or Error bit will turn on when the command is complete. If the Error Bit is on, you can use the CTRIO Read Error Code (CTRRDER) IBox to get extended error information.

The Workspace register is for internal use by this IBox instruction and MUST NOT be used anywhere else in your program.



CTRCLRT Parameters

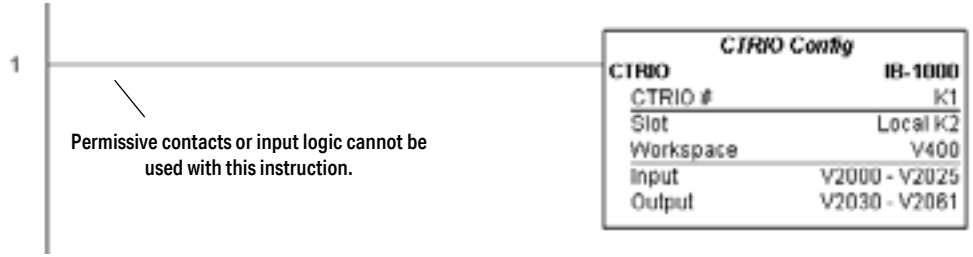
- CTRIO#: specifies a specific CTRIO module based on a user defined number (see CTRIO Config)
- Output#: specifies a CTRIO output to be used by the instruction
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the instruction has successfully completed
- Error: specifies a bit that will turn on if the instruction does not complete successfully

Parameter		DL06 Range
CTRIO#	K	K0-255
Output#	K	K0-3
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map

•

CTRCLRT Example

Rung 1: This sets up the CTRIO card in slot 2 of the local base. Each CTRIO in the system will need a separate CTRIO I-box before any CTRxxxx I-boxes can be used for them. The CTRIO has been configured to use V2000 through V2025 for its input data, and V2030 through V2061 for its output data.



Rung 2: This rung is a sample method for enabling the CTRCLRT command. A C-bit is used to allow the programmer to control the command from Data View for testing purposes.

Turning on C0 will cause the CTRCLRT instruction to clear the preset table for output #0 on the CTRIO in slot 2.

Operating procedure for this example code is to load the CTRCLRT_ex1.cwb file to your CTRIO, then enter the code shown here, change to RUN mode, enable output #0 by turning on C2 in Data View, turn encoder on CTRIO to value above 10 and output #0 light will come on and stay on until a count of 20 is reached, where it will turn off. Now reset the counter with C1, enable C0 to execute CTRCLRT command to clear the preset table, turn on C2 to enable output #0, then turn encoder to value of 10+ (output #0 should NOT turn on).

(example continued on next page)

CTRCLRT Example (cont'd)

Rung 3: This rung allows the programmer to reset the counter from the ladder logic.

Rung 4: This rung allows the operator to enable output #0 from the ladder code.

CTRIO Edit Preset Table Entry (CTREDPT) (IB-1003)

DS	Used
HPP	N/A

CTRIO Edit Preset Table Entry, on a leading edge transition to this IBox, will edit a single entry in a Preset Table on a specific CTRIO Output resource. This IBox is good if you are editing more than one entry in a file at a time. If you wish to do just one edit and then reload the table immediately, see the CTRIO Edit and Reload Preset Table Entry (CTREDRL) IBox. This IBox will take more than 1 PLC scan to execute. Either the Success or Error bit will turn on when the command is complete. If the Error Bit is on, you can use the CTRIO Read Error Code (CTRRDER) IBox to get extended error information.

Entry Type:

K0: Set

K1: Reset

K2: Pulse On (uses Pulse Time)

K3: Pulse Off (uses Pulse Time)

K4: Toggle

K5: Reset Count

Note that the Pulse Time parameter is ignored by some Entry Types.

The Workspace register is for internal use by this IBox instruction and MUST NOT be used anywhere else in your program.

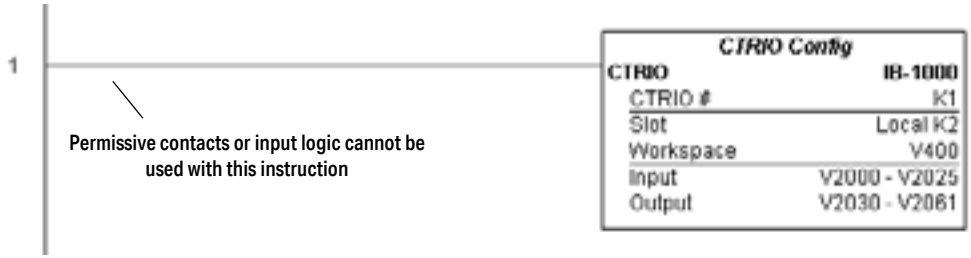
CTREDPT Parameters

- CTRIO#: specifies a specific CTRIO module based on a user defined number (see CTRIO Config Ibox)
- Output#: specifies a CTRIO output to be used by the instruction
- Table#: specifies the Table number of which an Entry is to be edited
- Entry#: specifies the Entry location in the Preset Table to be edited
- Entry Type: specifies the Entry Type to add during the edit
- Pulse Time: specifies a pulse time for the Pulse On and Pulse Off Entry Types
- Preset Count: specifies an initial count value to begin at after Reset
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the instruction has successfully completed
- Error: specifies a bit that will turn on if the instruction does not complete successfully

Parameter		DL06 Range
CTRIO#	K	K0-255
Output#	K	K0-3
Table#	V,K	K0-255; See DL06 V-memory map - Data Words
Entry#	V,K	K0-255; See DL06 V-memory map - Data Words
Entry Type	V,K	K0-5; See DL06 V-memory map - Data Words
Pulse Time	V,K	K0-65535; See DL06 V-memory map - Data Words
Preset Count	V,K	K0-2147434528; See DL06 V-memory map
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map

CTREDPT Example

Rung 1: This sets up the CTRIO card in slot 2 of the local base. Each CTRIO in the system will need a separate CTRIO I-box before any CTRxxxx I-boxes can be used for them. The CTRIO has been configured to use V2000 through V2025 for its input data, and V2030 through V2061 for its output data.



(example continued on next page)

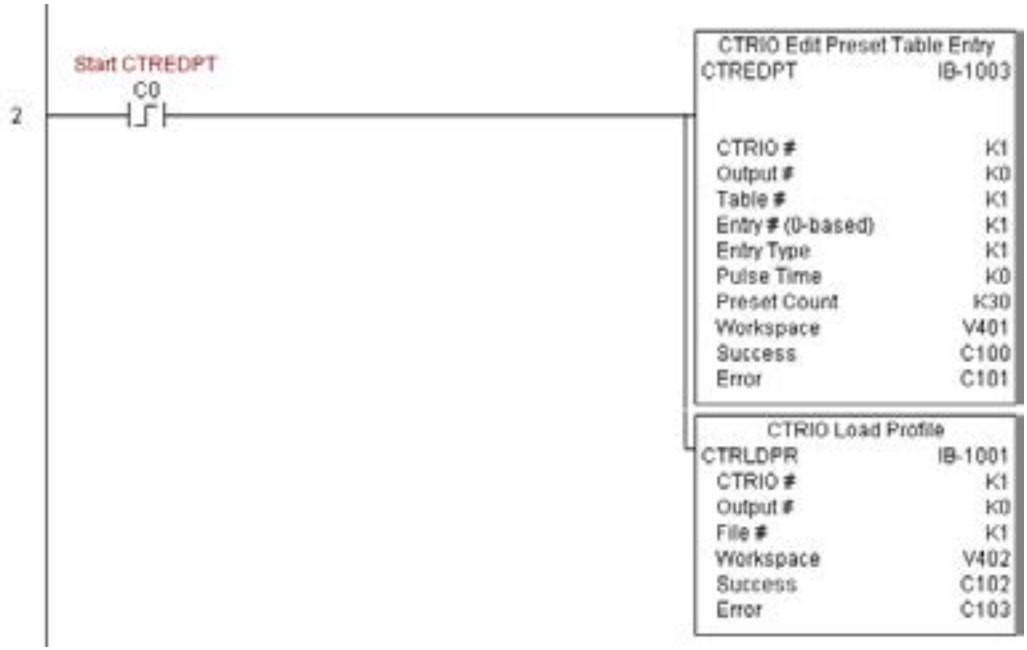
CTREDPT Example (cont'd)

Rung 2: This rung is a sample method for enabling the CTREDPT command. A C-bit is used to allow the programmer to control the command from Data View for testing purposes.

Turning on C0 will cause the CTREDPT instruction to change the second preset from a reset at a count of 20 to a reset at a count of 30 for output #0 on the CTRIO in slot 2.

Operating procedure for this example code is to load the CTREDPT_ex1.cwb file to your CTRIO, then enter the code shown here, change to RUN mode, enable output #0 by turning on C2 in Data View, turn encoder on CTRIO to value above 10 and output #0 light will come on and stay on until a count of 20 is reached, where it will turn off. Now reset the counter with C1, enable C0 to execute CTREDPT command to change the second preset, turn on C2 to enable output #0, then turn encoder to value of 10+ (output #0 should turn on) and then continue past a count of 30 (output #0 should turn off).

Note that we must also reload the profile after changing the preset(s), this is why the CTRLDPR command follows the CTREDPT command in this example.



(example continued on next page)

CTREDPT Example (cont'd)

Rung 3: This rung allows the programmer to reset the counter from the ladder logic.

Rung 4: This rung allows the operator to enable output #0 from the ladder code.

CTRIO Edit Preset Table Entry and Reload (CTREDRL) (IB-1002)

DS	Used
HPP	N/A

CTRIO Edit Preset Table Entry and Reload, on a leading edge transition to this IBox, will perform this dual operation to a CTRIO Output resource in one CTRIO command. This IBox will take more than 1 PLC scan to execute. Either the Success or Error bit will turn on when the command is complete. If the Error Bit is on, you can use the CTRIO Read Error Code (CTRRDER) IBox to get extended error information.

Entry Type:

K0: Set

K1: Reset

K2: Pulse On (uses Pulse Time)

K3: Pulse Off (uses Pulse Time)

K4: Toggle

K5: Reset Count

Note that the Pulse Time parameter is ignored by some Entry Types.

The Workspace register is for internal use by this IBox instruction and MUST NOT be used anywhere else in your program.

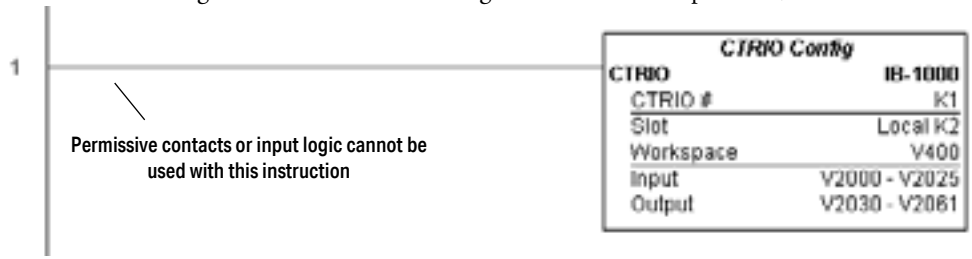
CTREDRL Parameters

- CTRIO#: specifies a specific CTRIO module based on a user defined number (see CTRIO Config Ibox)
- Output#: specifies a CTRIO output to be used by the instruction
- Table#: specifies the Table number of which an Entry is to be edited
- Entry#: specifies the Entry location in the Preset Table to be edited
- Entry Type: specifies the Entry Type to add during the edit
- Pulse Time: specifies a pulse time for the Pulse On and Pulse Off Entry Types
- Preset Count: specifies an initial count value to begin at after Reset
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the instruction has successfully completed
- Error: specifies a bit that will turn on if the instruction does not complete successfully
-

Parameter		DL06 Range
CTRIO#	K	K0-255
Output#	K	K0-3
Table#	V,K	K0-255; See DL06 V-memory map - Data Words
Entry#	V,K	K0-255; See DL06 V-memory map - Data Words
Entry Type	V,K	K0-5; See DL06 V-memory map - Data Words
Pulse Time	V,K	K0-65535; See DL06 V-memory map - Data Words
Preset Count	V,K	K0-2147434528; See DL06 V-memory map
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map

CTREDRL Example

Rung 1: This sets up the CTRIO card in slot 2 of the local base. Each CTRIO in the system will need a separate CTRIO I-box before any CTRxxxx I-boxes can be used for them. The CTRIO has been configured to use V2000 through V2025 for its input data, and V2030 through



V2061 for its output data.

(example continued on next page)

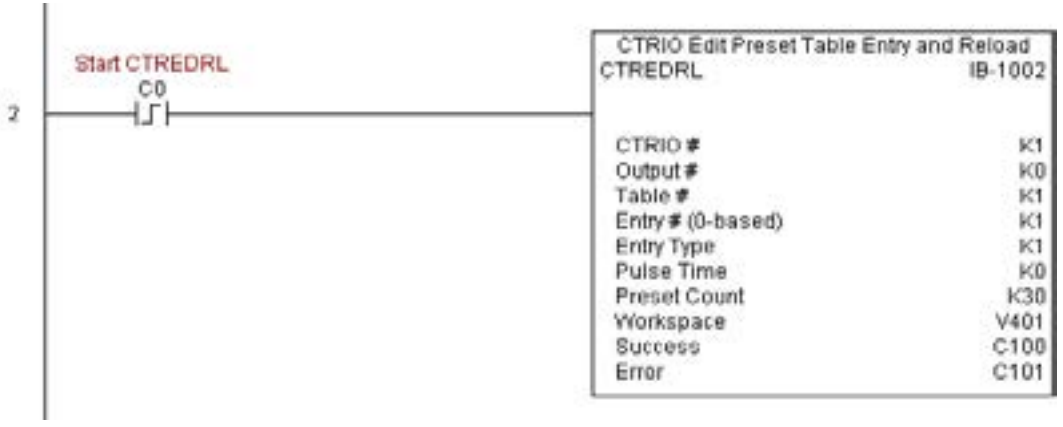
CTREDRL Example (cont'd)

Rung 2: This rung is a sample method for enabling the CTREDRL command. A C-bit is used to allow the programmer to control the command from Data View for testing purposes.

Turning on C0 will cause the CTREDRL instruction to change the second preset in file 1 from a reset at a value of 20 to a reset at a value of 30.

Operating procedure for this example code is to load the CTREDRL_ex1.cwb file to your CTRIO, then enter the code shown here, change to RUN mode, enable output #0 by turning on C2 in Data View, turn encoder on CTRIO to value above 10 and output #0 light will come on, continue to a count above 20 and the output #0 light will turn off. Now reset the counter with C1, enable C0 to execute CTREDRL command to change the second preset count value to 30, then turn encoder to value of 10+ (output #0 should turn on) and continue on to a value of 30+ and the output #0 light will turn off.

Note that it is not necessary to reload this file separately, however, the command can only change one value at a time.



(example continued on next page)

CTREDRL Example (cont'd)

Rung 3: This rung allows the programmer to reset the counter from the ladder logic.

Rung 4: This rung allows the operator to enable output #0 from the ladder code.

CTRIO Initialize Preset Table (CTRINPT) (IB-1004)

DS	Used
HPP	N/A

CTRIO Initialize Preset Table, on a leading edge transition to this IBox, will create a single entry Preset Table in memory but not as a file, on a specific CTRIO Output resource. This IBox will take more than 1 PLC scan to execute. Either the Success or Error bit will turn on when the command is complete. If the Error Bit is on, you can use the CTRIO Read Error Code (CTRRDER) IBox to get extended error information.

Entry Type:

K0: Set

K1: Reset

K2: Pulse On (uses Pulse Time)

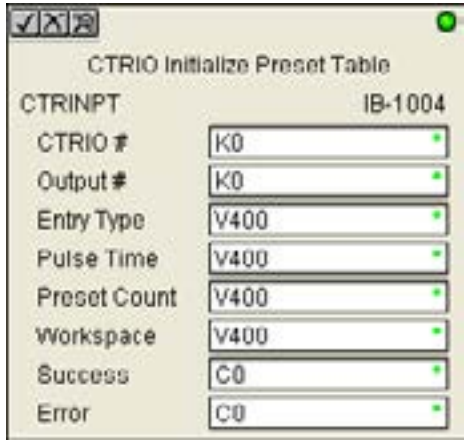
K3: Pulse Off (uses Pulse Time)

K4: Toggle

K5: Reset Count

Note that the Pulse Time parameter is ignored by some Entry Types.

The Workspace register is for internal use by this IBox instruction and MUST NOT be used anywhere else in your program.



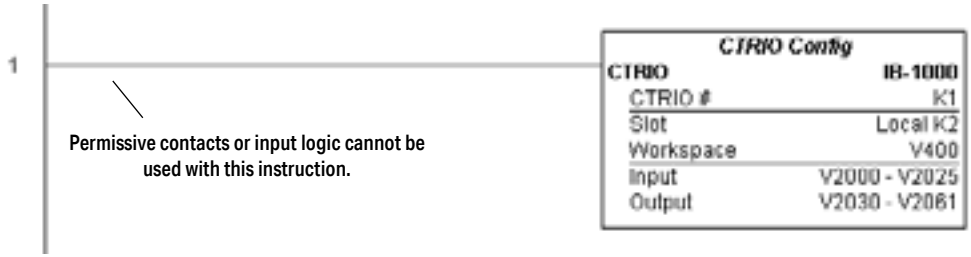
CTRINPT Parameters

- CTRIO#: specifies a specific CTRIO module based on a user defined number (see CTRIO Config Ibox)
- Output#: specifies a CTRIO output to be used by the instruction
- Entry Type: specifies the Entry Type to add during the edit
- Pulse Time: specifies a pulse time for the Pulse On and Pulse Off Entry Types
- Preset Count: specifies an initial count value to begin at after Reset
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the instruction has successfully completed
- Error: specifies a bit that will turn on if the instruction does not complete successfully
-

Parameter		DL06 Range
CTRIO#	K	K0-255
Output#	K	K0-3
Entry Type	V,K	K0-5; See DL06 V-memory map - Data Words
Pulse Time	V,K	K0-65535; See DL06 V-memory map - Data Words
Preset Count	V,K	K0-2147434528; See DL06 V-memory map
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map

CTRINPT Example

Rung 1: This sets up the CTRIO card in slot 2 of the local base. Each CTRIO in the system will need a separate CTRIO I-box before any CTRxxxx I-boxes can be used for them. The CTRIO has been configured to use V2000 through V2025 for its input data, and V2030 through V2061 for its output data.



(example continued on next page)

CTRINPT Example (cont'd)

Rung 2: This rung is a sample method for enabling the CTRINPT command. A C-bit is used to allow the programmer to control the command from Data View for testing purposes.

Turning on C0 will cause the CTRINPT instruction to create a single entry preset table, but not as a file, and use it for the output #0. In this case the single preset will be a set at a count of 15 for output #0.

Operating procedure for this example code is to load the CTRINPT_ex1.cwb file to your CTRIO, then enter the code shown here, change to RUN mode, enable output #0 by turning on C2 in Data View, turn encoder on CTRIO to value above 15 and output #0 light will not come on. Now reset the counter with C1, enable C0 to execute CTRINPT command to create a single preset table with a preset to set output #0 at a count of 15, then turn encoder to value of 15+ (output #0 should turn on).

(example continued on next page)

CTRINPT Example (cont'd)

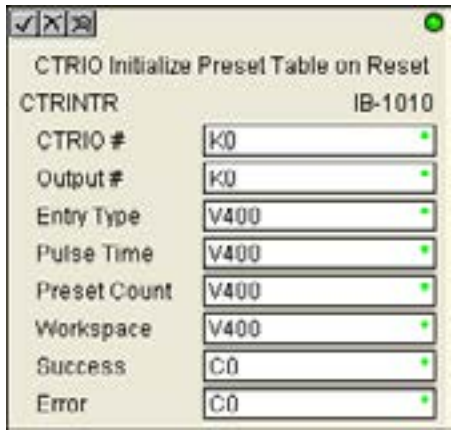
Rung 3: This rung allows the programmer to reset the counter from the ladder logic.

Rung 4: This rung allows the operator to enable output #0 from the ladder code.

CTRIO Initialize Preset Table (CTRINTR) (IB-1010)

DS	Used
HPP	N/A

CTRIO Initialize Preset Table, on a leading edge transition to this IBox, will create a single entry Preset Table in memory but not as a file, on a specific CTRIO Output resource.This IBox will take more than 1 PLC scan to execute. Either the Success or Error bit will turn on when the command is complete. If the Error Bit is on, you can use the CTRIO Read Error Code (CTRRDER) IBox to get extended error information.



Entry Type:

K0: Set

K1: Reset

K2: Pulse On (uses Pulse Time)

K3: Pulse Off (uses Pulse Time)

K4: Toggle

K5: Reset Count

Note that the Pulse Time parameter is ignored by some Entry Types.

The Workspace register is for internal use by this IBox instruction and MUST NOT be used anywhere else in your program.

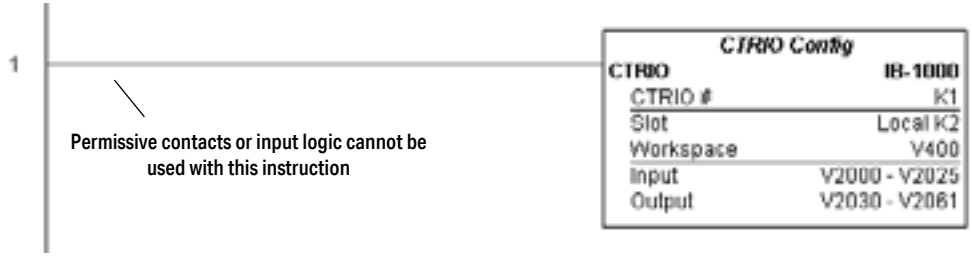
CTRINTR Parameters

- CTRIO#: specifies a specific CTRIO module based on a user defined number (see CTRIO Config Ibox)
- Output#: specifies a CTRIO output to be used by the instruction
- Entry Type: specifies the Entry Type to add during the edit
- Pulse Time: specifies a pulse time for the Pulse On and Pulse Off Entry Types
- Preset Count: specifies an initial count value to begin at after Reset
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the instruction has successfully completed
- Error: specifies a bit that will turn on if the instruction does not complete successfully

Parameter		DL06 Range
CTRIO#	K	K0-255
Output#	K	K0-3
Entry Type	V,K	K0-5; See DL06 V-memory map - Data Words
Pulse Time	V,K	K0-65535; See DL06 V-memory map - Data Words
Preset Count	V,K	K0-2147434528; See DL06 V-memory map
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map

CTRINTR Example

Rung 1: This sets up the CTRIO card in slot 2 of the local base. Each CTRIO in the system will need a separate CTRIO I-box before any CTRxxxx I-boxes can be used for them. The CTRIO has been configured to use V2000 through V2025 for its input data, and V2030



(example continued on next page)

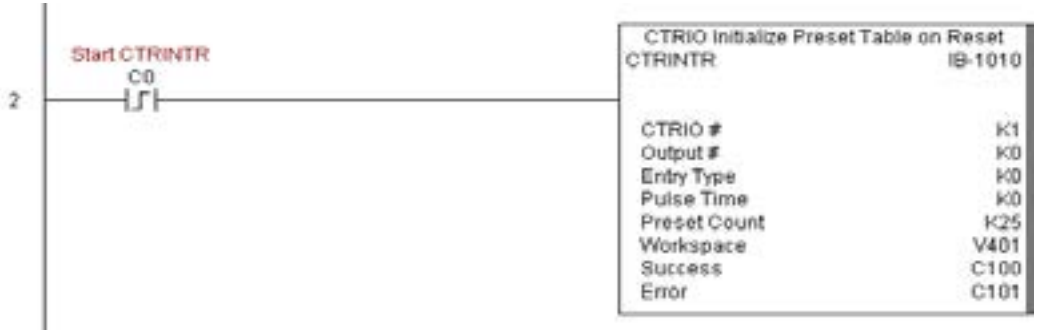
through V2061 for its output data.

CTRINTR Example (cont'd)

Rung 2: This rung is a sample method for enabling the CTRINTR command. A C-bit is used to allow the programmer to control the command from Data View for testing purposes.

Turning on C0 will cause the CTRINTR instruction to create a single entry preset table, but not as a file, and use it for output #0, the new preset will be loaded when the current count is reset. In this case the single preset will be a set at a count of 25 for output #0.

Operating procedure for this example code is to load the CTRINTR_ex1.cwb file to your CTRIO, then enter the code shown here, change to RUN mode, enable output #0 by turning on C2 in Data View, turn encoder on CTRIO to value above 10 and output #0 light will come on. Now turn on C0 to execute the CTRINTR command, reset the counter with C1, then turn encoder to value of 25+ (output #0 should turn on).



(example continued on next page)

CTRINTR Example (cont'd)

Rung 3: This rung allows the programmer to reset the counter from the ladder logic.

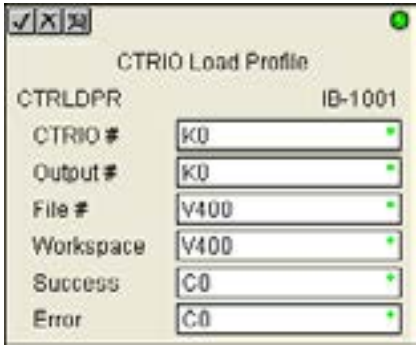
Rung 4: This rung allows the operator to enable output #0 from the ladder code.

CTRIO Load Profile (CTRLDPR) (IB-1001)

DS	Used
HPP	N/A

CTRIO Load Profile loads a CTRIO Profile File to a CTRIO Output resource on a leading edge transition to this IBox. This IBox will take more than 1 PLC scan to execute. Either the Success or Error bit will turn on when the command is complete. If the Error Bit is on, you can use the CTRIO Read Error Code (CTRRDER) IBox to get extended error information.

The Workspace register is for internal use by this IBox instruction and MUST NOT be used anywhere else in your program.



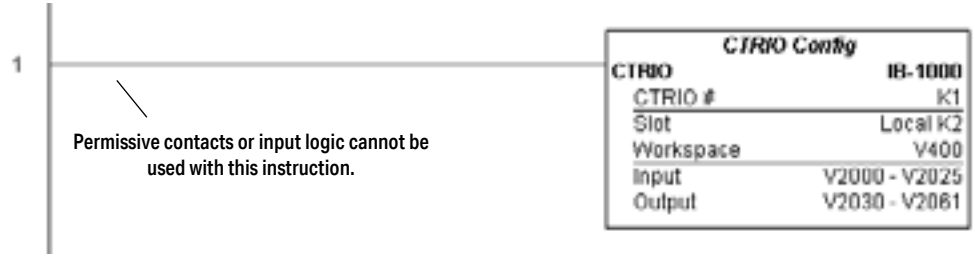
CTRLDPR Parameters

- CTRIO#: specifies a specific CTRIO module based on a user defined number (see CTRIO Config)
- Output#: specifies a CTRIO output to be used by the instruction
- File#: specifies a CTRIO profile File number to be loaded
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the instruction has successfully completed
- Error: specifies a bit that will turn on if the instruction does not complete successfully

Parameter		DL06 Range
CTRIO#	K	K0-255
Output#	K	K0-3
File#	V,K	K0-255; See DL06 V-memory map - Data Words
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map

CTRLDPR Example

Rung 1: This sets up the CTRIO card in slot 2 of the local base. Each CTRIO in the system will need a separate CTRIO I-box before any CTRxxxx I-boxes can be used for them. The CTRIO has been configured to use V2000 through V2025 for its input data, and V2030 through V2061 for its output data.



Rung 2: This CTRIO Load Profile IBox will load File #1 into the working memory of Output 0 in CTRIO #1. This example program requires that you load CTRLDPR_IBox.cwb into your Hx-CTRIO(2) module.

(example continued on next page)

CTRLDPR Example (cont'd)

Rung 3: If the file is successfully loaded, set Profile_Loaded.

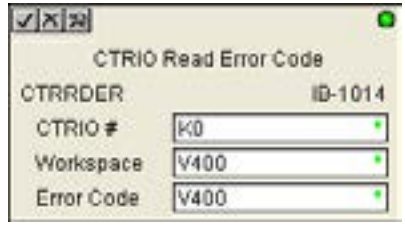
CTRIO Read Error (CTRRDER) (IB-1014)

DS	Used
HPP	N/A

CTRIO Read Error Code will get the decimal error code value from the CTRIO module (listed below) and place it into the given Error Code register, on a leading edge transition to the IBox

Since the Error Code in the CTRIO is only maintained until another CTRIO command is given, you must use this instruction immediately after the CTRIO IBox that reports an error via its Error bit parameter.

The Workspace register is for internal use by this IBox instruction and MUST NOT be used anywhere else in your program.



Error Codes:

0: No Error

100: Specified command code is unknown or unsupported

101: File number not found in the file system

102: File type is incorrect for specified output function

103: Profile type is unknown

104: Specified input is not configured as a limit on this output

105: Specified limit input edge is out of range

106: Specified input function is un configured or invalid

107: Specified input function number is out of range

108: Specified preset function is invalid

109: Preset table is full

110: Specified Table entry is out of range

111: Specified register number is out of range

112: Specified register is an unconfigured input or output

2001: Error reading Error Code - cannot access CTRIO via ERM

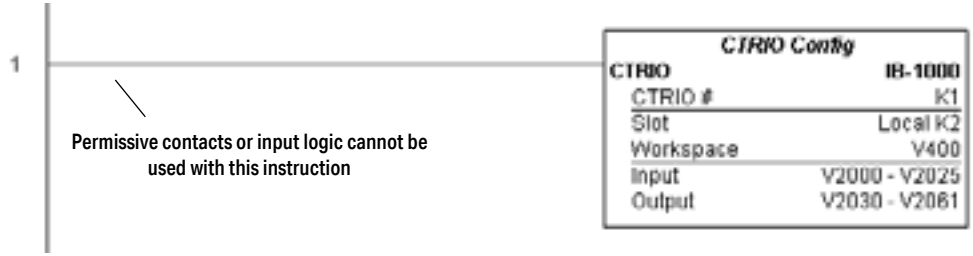
CTRRDER Parameters

- CTRIO#: specifies a specific CTRIO module based on a user defined number (see CTRIO Config)
- Workspace: specifies a V-memory location that will be used by the instruction
- Error Code: specifies the location where the Error Code will be written

Parameter		DL06 Range
CTRIO#	K	K0-255
Workspace	V	See DL06 V-memory map - Data Words
Error Code	V	See DL06 V-memory map - Data Words

CTRRDER Example

Rung 1: This sets up the CTRIO card in slot 2 of the local base. Each CTRIO in the system will need a separate CTRIO I-box before any CTRxxxx I-boxes can be used for them. The CTRIO has been configured to use V2000 through V2025 for its input data, and V2030 through V2061 for its output data.



Rung 2: This CTRIO Read Error Code IBox will read the Extended Error information from CTRIO #1. This example program requires that you load CTRRDER_IBox.cwb into your Hx-CTRIO(2) module.

CTRIO Run to Limit Mode (CTRRTLM) (IB-1011)

DS	Used
HPP	N/A

CTRIO Run To Limit Mode, on a leading edge transition to this IBox, loads the Run to Limit command and given parameters on a specific Output resource. The CTRIO's Input(s) must be configured as Limit(s) for this function to work.

Valid Hexadecimal Limit Values:

K00 - Rising Edge of Ch1/C

K10 - Falling Edge of Ch1/C

K20 - Both Edges of Ch1/C

K01 - Rising Edge of Ch1/D

K11 - Falling Edge of Ch1/D

K21 - Both Edges of Ch1/D

K02 - Rising Edge of Ch2/C

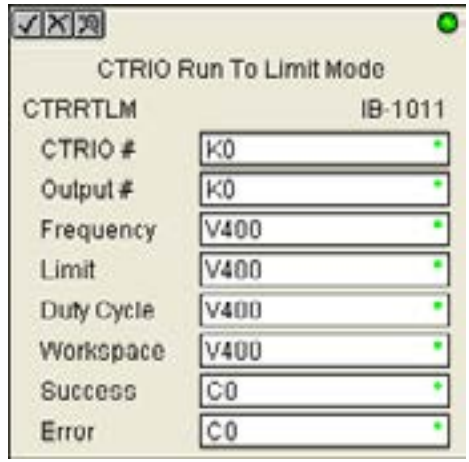
K12 - Falling Edge of Ch2/C

K22 - Both Edges of Ch2/C

K03 - Rising Edge of Ch2/D

K13 - Falling Edge of Ch2/D

K23 - Both Edges of Ch2/D



This IBox will take more than 1 PLC scan to execute. Either the Success or Error bit will turn on when the command is complete. If the Error Bit is on, you can use the CTRIO Read Error Code (CTRRDER) IBox to get extended error information.

The Workspace register is for internal use by this IBox instruction and MUST NOT be used anywhere else in your program.

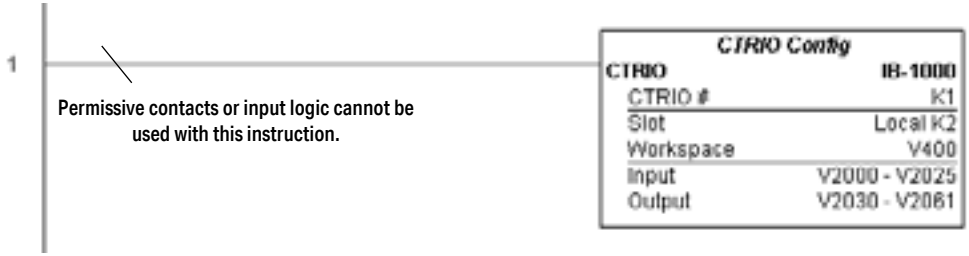
CTRRTLM Parameters

- CTRIO#: specifies a specific CTRIO module based on a user defined number (see CTRIO Config Ibox)
- Output#: specifies a CTRIO output to be used by the instruction
- Frequency: specifies the output pulse rate (H0-CTRIO: 20Hz - 25KHz / H0-CTRIO2: 20Hz - 250 KHz)
- Limit: the CTRIO's Input(s) must be configured as Limit(s) for this function to operate
- Duty Cycle: specifies the % of on time versus off time. This is a hex number. Default of 0 is 50%, also entering 50 will yield 50%. 50% duty cycle is defined as on half the time and off half the time
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the instruction has successfully completed
- Error: specifies a bit that will turn on if the instruction does not complete successfully

Parameter		DL06 Range
CTRIO#	K	K0-255
Output#	K	K0-3
Frequency	V,K	K20-20000; See DL06 V-memory map - Data Words
Limit	V,K	K0-FF; See DL06 V-memory map - Data Words
Duty Cycle	V,K	K0-99; See DL06 V-memory map - Data Words
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map

CTRRTLM Example

Rung 1: This sets up the CTRIO card in slot 2 of the local base. Each CTRIO in the system will need a separate CTRIO I-box before any CTRxxxx I-boxes can be used for them. The CTRIO has been configured to use V2000 through V2025 for its input data, and V2030 through V2061 for its output data.



Rung 2: This CTRIO Run To Limit Mode IBox sets up Output #0 in CTRIO #1 to output pulses at a Frequency of 1000 Hz until Llimit #0 comes on. This example program requires that you load CTRRTLM_IBox.cwb into your Hx-CTRIO(2) module.

(example continued on next page)

CTRRTLM Example (cont'd)

Rung 3: If the Run To Limit Mode parameters are OK, set the Direction Bit and Enable the output.

CTRIO Run to Position Mode (CTRRTPM) (IB-1012)

DS	Used
HPP	N/A

CTRIO Run To Position Mode, on a leading edge transition to this IBox, loads the Run to Position command and given parameters on a specific Output resource.

Valid Function Values are:

00: Less Than Ch1/Fn1

10: Greater Than Ch1/Fn1

01: Less Than Ch1/Fn2

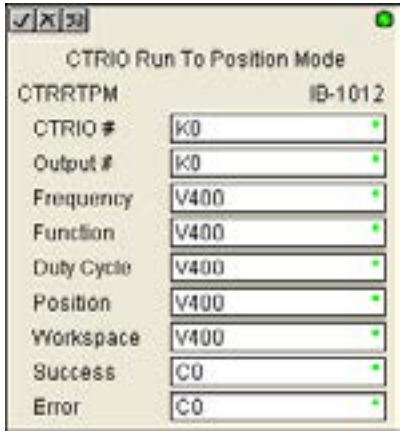
11: Greater Than Ch1/Fn2

02: Less Than Ch2/Fn1

12: Greater Than Ch2/Fn1

03: Less Than Ch2/Fn2

13: Greater Than Ch2/Fn2



This IBox will take more than 1 PLC scan to execute. Either the Success or Error bit will turn on when the command is complete. If the Error Bit is on, you can use the CTRIO Read Error Code (CTRRDER) IBox to get extended error information.

The Workspace register is for internal use by this IBox instruction and MUST NOT be used anywhere else in your program.

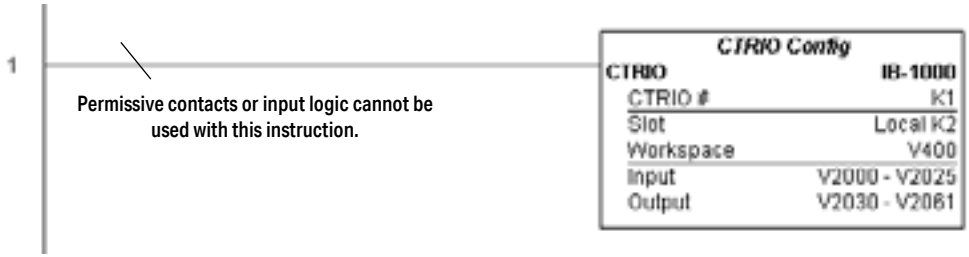
CTRRTPM Parameters

- CTRIO#: specifies a specific CTRIO module based on a user defined number (see CTRIO Config Ibox)
- Output#: specifies a CTRIO output to be used by the instruction
- Frequency: specifies the output pulse rate (H0-CTRIO: 20Hz - 25KHz / H0-CTRIO2: 20Hz - 250 KHz)
- Duty Cycle: specifies the % of on time versus off time. This is a hex number. Default of 0 is 50%, also entering 50 will yield 50%. 50% duty cycle is defined as on half the time and off half the time
- Position: specifies the count value, as measured on the encoder input, at which the output pulse train will be turned off
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the instruction has successfully completed
- Error: specifies a bit that will turn on if the instruction does not complete successfully

Parameter		DL06 Range
CTRIO#	K	K0-255
Output#	K	K0-3
Frequency	V,K	K20-20000; See DL06 V-memory map - Data Words
Duty Cycle	V,K	K0-99; See DL06 V-memory map
Position	V,K	K0-2147434528; See DL06 V-memory map
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map

CTRRTPM Example

Rung 1: This sets up the CTRIO card in slot 2 of the local base. Each CTRIO in the system will need a separate CTRIO I-box before any CTRxxxx I-boxes can be used for them. The CTRIO has been configured to use V2000 through V2025 for its input data, and V2030 through V2061 for its output data.



(example continued on next page)

CTRRTPM Example (cont'd)

Rung 2: This CTRIO Run To Position Mode IBox sets up Output #0 in CTRIO #1 to output pulses at a Frequency of 1000 Hz, use the 'Greater than Ch1/Fn1' comparison operator, until the input position of 1500 is reached. This example program requires that you load CTRRTPM_IBox.cwb into your Hx-CTRIO(2) module.

Rung 3: If the Run To Position Mode parameters are OK, set the Direction Bit and Enable the output.

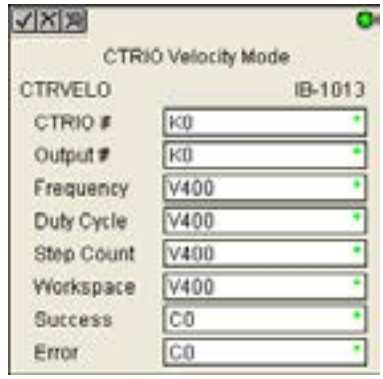
CTRIO Velocity Mode (CTRVELO) (IB-1013)

DS	Used
HPP	N/A

CTRIO Velocity Mode loads the Velocity command and given parameters on a specific Output resource on a leading edge transition to this IBox.

This IBox will take more than 1 PLC scan to execute. Either the Success or Error bit will turn on when the command is complete. If the Error Bit is on, you can use the CTRIO Read Error Code (CTRRDER) IBox to get extended error information.

The Workspace register is for internal use by this IBox instruction and MUST NOT be used anywhere else in your program.



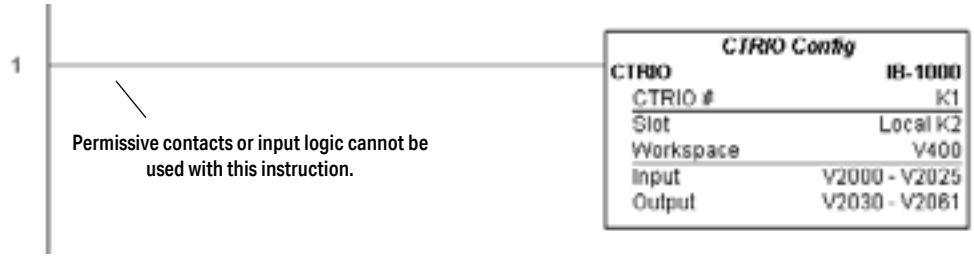
CTRVELO Parameters

- CTRIO#: specifies a specific CTRIO module based on a user defined number (see CTRIO Config Ibox)
- Output#: specifies a CTRIO output to be used by the instruction
- Frequency: specifies the output pulse rate (H0-CTRIO: 20Hz - 25KHz / H0-CTRIO2: 20Hz - 250 KHz)
- Duty Cycle: specifies the % of on time versus off time. This is a hex number. Default of 0 is 50%, also entering 50 will yield 50%. 50% duty cycle is defined as on half the time and off half the time
- Step Count: This DWORD value specifies the number of pulses to output. A Step Count value of -1 (or 0XFFFFFFFF) causes the CTRIO to output pulses continuously. Negative Step Count values must be V-Memory references.
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the instruction has successfully completed
- Error: specifies a bit that will turn on if the instruction does not complete successfully

Parameter		DL06 Range
CTRIO#	K	K0-255
Output#	K	K0-3
Frequency	V,K	K20-20000; See DL06 V-memory map - Data Words
Duty Cycle	V,K	K0-99; See DL06 V-memory map
Step Count	V,K	K0-2147434528; See DL06 V-memory map
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map

CTRVELO Example

Rung 1: This sets up the CTRIO card in slot 2 of the local base. Each CTRIO in the system will need a separate CTRIO I-box before any CTRxxxx I-boxes can be used for them. The CTRIO has been configured to use V2000 through V2025 for its input data, and V2030 through V2061 for its output data.



Rung 2: This CTRIO Velocity Mode IBox sets up Output #0 in CTRIO #1 to output 10,000 pulses at a Frequency of 1000 Hz. This example program requires that you load CTRVELO_IBox.cwb into your Hx-CTRIO(2) module.

(example continued on next page)

CTRVELO Example (cont'd)

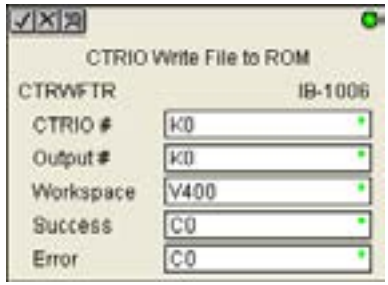
Rung 3: If the Velocity Mode parameters are OK, set the Direction Bit and Enable the output.

CTRIO Write File to ROM (CTRWFTR) (IB-1006)

DS	Used
HPP	N/A

CTRIO Write File to ROM writes the runtime changes made to a loaded CTRIO Preset Table back to Flash ROM on a leading edge transition to this IBox. This IBox will take more than 1 PLC scan to execute. Either the Success or Error bit will turn on when the command is complete. If the Error Bit is on, you can use the CTRIO Read Error Code (CTRRDER) IBox to get extended error information.

The Workspace register is for internal use by this IBox instruction and MUST NOT be used anywhere else in your program.



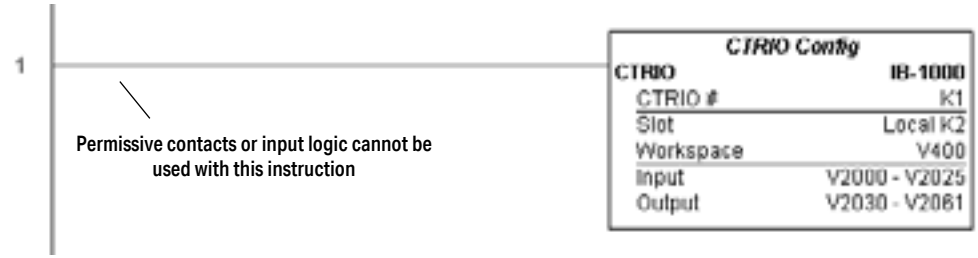
CTRWFTR Parameters

- CTRIO#: specifies a specific CTRIO module based on a user defined number (see CTRIO Config Ibox)
- Output#: specifies a CTRIO output to be used by the instruction
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the instruction has successfully completed
- Error: specifies a bit that will turn on if the instruction does not complete successfully

Parameter		DL06 Range
CTRIO#	K	K0-255
Output#	K	K0-3
Workspace	V	See DL06 V-memory map - Data Words
Success	X,Y,C,GX,GY,B	See DL06 V-memory map
Error	X,Y,C,GX,GY,B	See DL06 V-memory map

CTRWFTR Example

Rung 1: This sets up the CTRIO card in slot 2 of the local base. Each CTRIO in the system will need a separate CTRIO I-box before any CTRxxxx I-boxes can be used for them. The CTRIO has been configured to use V2000 through V2025 for its input data, and V2030 through V2061 for its output data.



Rung 2: This CTRIO Edit Preset Table Entry IBox will change Entry 0 in Table #2 to be a RESET at Count 3456. This example program requires that you load CTRWFTR_IBox.cwb into your Hx-CTRIO(2) module.

(example continued on next page)

CTRWFTR Example (cont'd)

Rung 3: If the file is successfully edited, use a Write File To ROM IBox to save the edited table back to the CTRIO's ROM, thereby making the changes retentive.

Drum Instruction Programming

In This Chapter...

Introduction

Purpose

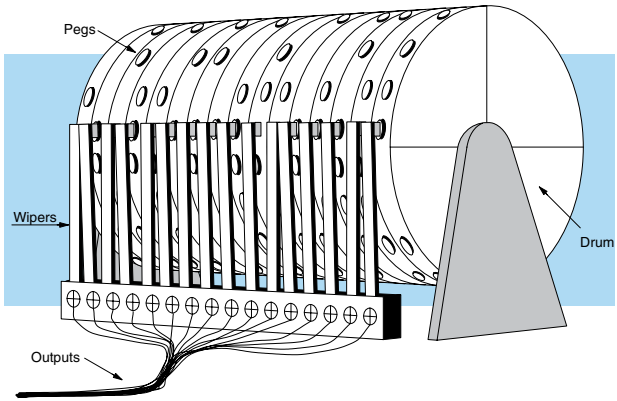
The Event Drum (EDRUM) instruction in the DL06 CPU electronically simulates an electro-mechanical drum sequencer. The instruction offers enhancements to the basic principle, which we describe first.

Drum Terminology

Drum instructions are best suited for repetitive processes that consist of a finite number of steps. They can do the work of many rungs of ladder logic with elegant simplicity. Therefore, drums can save a lot of programming and debugging time.

We introduce some terminology associated with the **drum** instruction by describing the original mechanical drum shown below. The mechanical drum generally has pegs on its curved surface. The pegs are populated in a particular **pattern**, representing a set of desired actions for machine control. A motor or solenoid rotates the drum a precise amount at specific times. During rotation, stationary wipers sense the presence of pegs (present = on, absent = off). This interaction makes or breaks electrical contact with the wipers, creating electrical **outputs** from the drum. The outputs are wired to devices on a machine for On/Off control.

Drums usually have a finite number of positions within one rotation, called **steps**. Each step represents some process step. At powerup, the drum **resets** to a particular step. The drum rotates from one step to the next based on a **timer**, or on some external **event**. During special conditions, a machine operator can manually increment the drum step using a **jog** control on the drum's drive mechanism. The contact closure of each wiper generates a unique on/off pattern called a **sequence**, designed for controlling a specific machine. Because the drum is circular, it automatically repeats the sequence once per rotation. Applications vary greatly, and a particular drum may rotate once per second, or as slowly as once per week.



Electronic drums provide the benefits of mechanical drums and more. For example, they have a **preset** feature that is impossible for mechanical drums: The preset function lets you move from the present step *directly* to any other step on command!

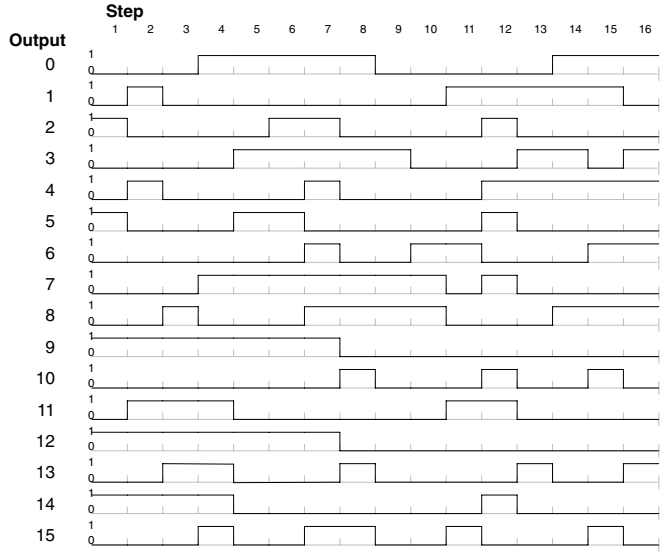
Drum Chart Representation

For editing purposes, the electronic drum is presented in chart form in *Direct*SOFT and in this manual. Imagine slicing the surface of a hollow drum cylinder between two rows of pegs, then pressing it flat. Now you can view the drum as a chart as shown below. Each row represents a step, numbered 1 through 16. Each column represents an output, numbered 0 through 15 (to match word bit numbering). The solid circles in the chart represent pegs (On state) in the mechanical drum, and the open circles are empty peg sites (Off state).

	OUTPUTS															
STEP	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	○	●	○	●	○	○	●	○	○	○	●	○	○	●	○	○
2	○	●	○	●	●	○	●	○	○	○	○	●	○	○	●	○
3	○	●	●	●	●	○	●	●	○	○	○	○	○	○	○	○
4	●	●	●	●	●	○	●	○	●	○	○	○	○	○	○	●
5	○	○	○	●	○	○	●	○	●	○	●	○	●	○	○	●
6	○	○	○	●	○	○	●	○	●	○	●	○	●	●	○	●
7	●	●	○	●	○	○	●	●	●	●	○	●	●	●	○	●
8	●	○	●	○	○	●	○	●	●	○	○	○	●	○	○	●
9	○	○	○	○	○	○	○	●	●	○	○	○	●	○	○	○
10	○	○	○	○	○	○	○	●	●	●	○	○	○	○	○	○
11	●	○	○	○	●	○	○	○	○	●	○	○	○	○	●	○
12	○	●	○	○	●	●	○	○	●	○	●	●	○	●	●	○
13	○	○	●	○	○	○	○	○	○	○	○	●	●	○	●	○
14	○	○	○	○	○	○	○	●	○	○	○	●	●	○	●	●
15	●	○	○	○	○	●	○	●	○	●	○	●	○	○	●	●
16	○	○	●	○	○	○	○	●	○	●	○	●	●	○	○	●

Output Sequences

The mechanical drum sequencer derives its name from sequences of control changes on its electrical outputs. The following figure shows the sequence of On/Off controls generated by the drum pattern above. Compare the two, and you will find that they are equivalent! If you can see their equivalence, you are well on your way to understanding drum instruction operation.



Step Transitions

Drum Instruction Types

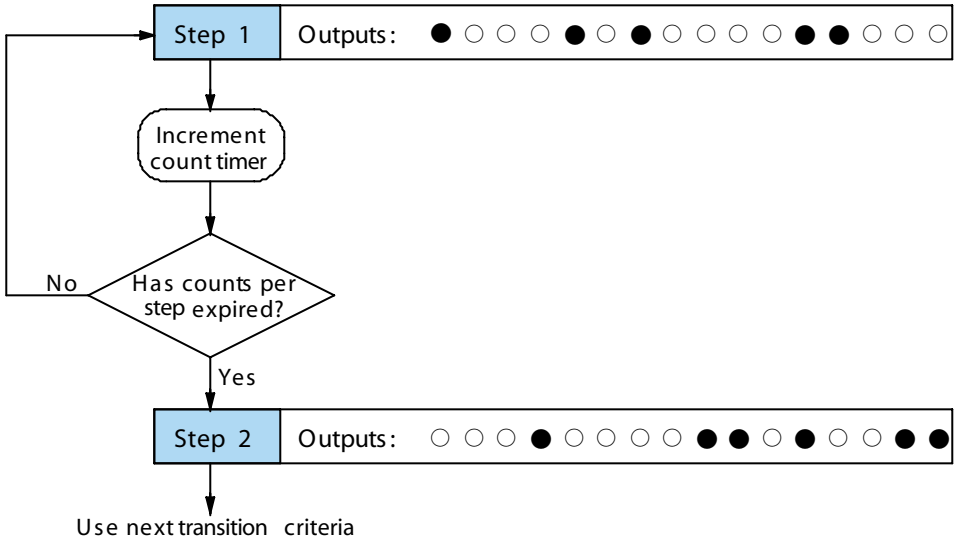
There are two types of Drum instructions in the DL06 CPU:

- Timed Drum with Discrete Outputs (DRUM)
- Time and Event Drum with Discrete Outputs (EDRUM)

The two drum instructions include time-based step transitions, and the EDRUM includes event-based transitions as well. Each drum has 16 steps, and each step has 16 outputs. Refer to the figure below. Each output can be either a Y or C coil, offering programming flexibility. We assign Step 1 an arbitrary unique output pattern.

Timer-Only Transitions

Drums move from one step to another based on time and/or an external event (input). Each step has its own transition condition which you assign during the drum instruction entry. The figure below shows how timer-only transitions work.



The drum stays in Step 1 for a specific duration (user-programmable). The timebase of the timer is programmable, from 0.01 seconds to 99.99 seconds. This establishes the resolution, or the duration of each "tick of the clock." Each step uses the same timebase, but has its own unique counts per step, which you program. When the counts for Step 1 have expired, then the drum moves to Step 2. The outputs change immediately to match the new pattern for Step 2.

The drum spends a specific amount of time in each step, given by the formula:

Time in step = 0.01 seconds X Timebase x Counts per step

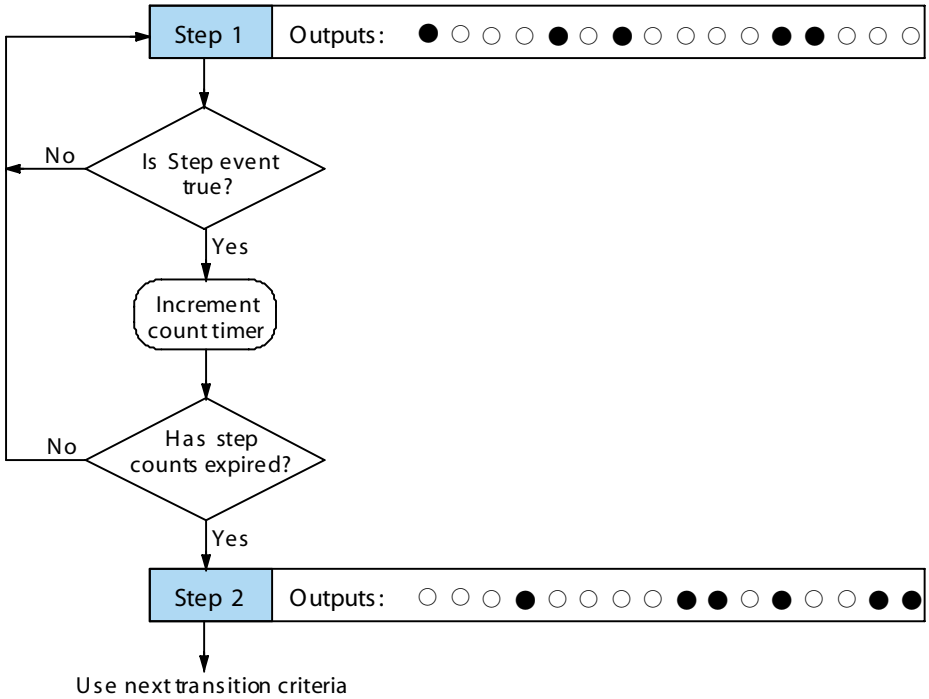
For example, if you program a 5 second time base and 12 counts for Step 1, then the drum will spend 60 seconds in Step 1. The maximum time for any step is given by the formula:

Max Time per step = 0.01 seconds X 9999 X 9999
= 999,800 seconds = 277.7 hours = 11.6 days

***NOTE**: When first choosing the timebase resolution, a good rule of thumb is to make it about 1/10 the duration of the shortest step in your drum. Then you will be able to optimize the duration of that step in 10% increments. Other steps with longer durations allow optimizing by even smaller increments (percentage-wise). Also, note that the drum instruction executes once per CPU scan. Therefore, it is pointless to specify a drum timebase that is much faster than the CPU scan time.*

Timer and Event Transitions

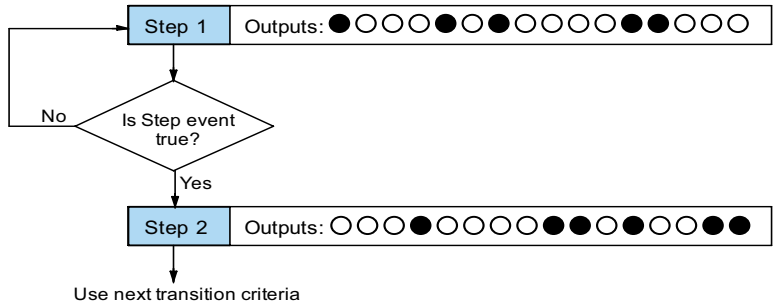
Step transitions may also occur based on time and/or external events. The figure below shows how step transitions work in these cases.



When the drum enters Step 1, it sets the output pattern as shown. Then it begins polling the external input programmed for that step. You can define event inputs as X, Y, or C discrete point types. Suppose we select X0 for the Step 1 event input. If X0 is off, then the drum remains in Step 1. When X0 is On, the event criteria is met and the timer increments. The timer increments as long as the event (X0) remains true. When the counts for Step 1 have expired, then the drum moves to Step 2. The outputs change immediately to match the new pattern for Step 2.

Event-Only Transitions

Step transitions do not require both the event and the timer criteria programmed for each step. You have the option of programming just one of the two, and even mixing transition types among all the steps of the drum. For example, you might want Step 1 to transition on an event, Step 2 to transition on time only, and Step 3 to transition on both time and an event. Furthermore, you may elect to use only part of the 16 steps, and only part of the 16 outputs.



Counter Assignments

Each drum instruction uses the resources of four counters in the CPU. When programming the drum instruction, you select the first counter number. The drum also uses the next three counters automatically. The counter bit associated with the first counter turns on when the drum has completed its cycle, going off when the drum is reset. These counter values and the counter bit precisely indicate the progress of the drum instruction, and can be monitored by your ladder program.

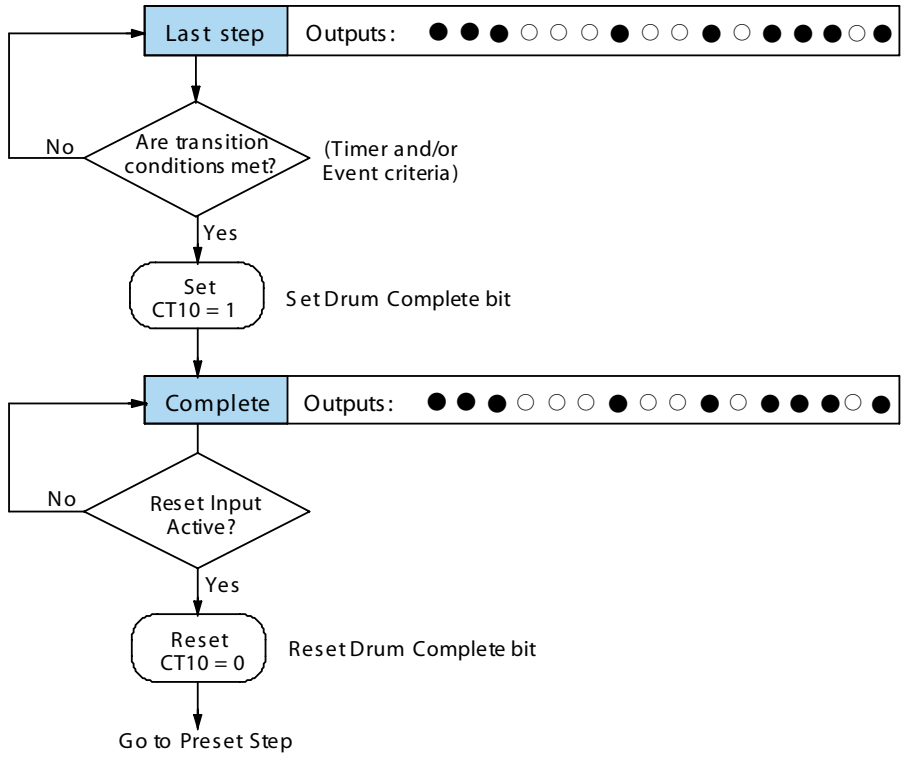
Suppose we program a timer drum to have 8 steps, and we select CT10 for the counter number (remember, counter numbering is in octal). Counter usage is shown to the right. The right column holds typical values, interpreted below.

Counter Assignments			
CT10	Counts in step	V1010	1528
CT11	Timer Value	V1011	0200
CT12	Preset Step	V1012	0001
CT13	Current Step	V1013	0004

CT10 shows that we are at the 1528th count in the current step, which is step 4 (shown in CT13). If we have programmed step 4 to have 3000 counts, then the step is just over half completed. CT11 is the count timer, shown in units of 0.01 seconds. So, each least-significant-digit change represents 0.01 seconds. The value of 200 means that we have been in the current count (1528) for 2 seconds (0.01 x 200). Finally, CT12 holds the preset step value which was programmed into the drum instruction. When the drum's Reset input is active, it presets to step 1 in this case. The value of CT12 changes only if the ladder program writes to it, or the drum instruction is edited and the program is restarted. Counter bit CT10 turns on when the drum cycle is complete, and turns off when the drum is reset.

Last Step Completion

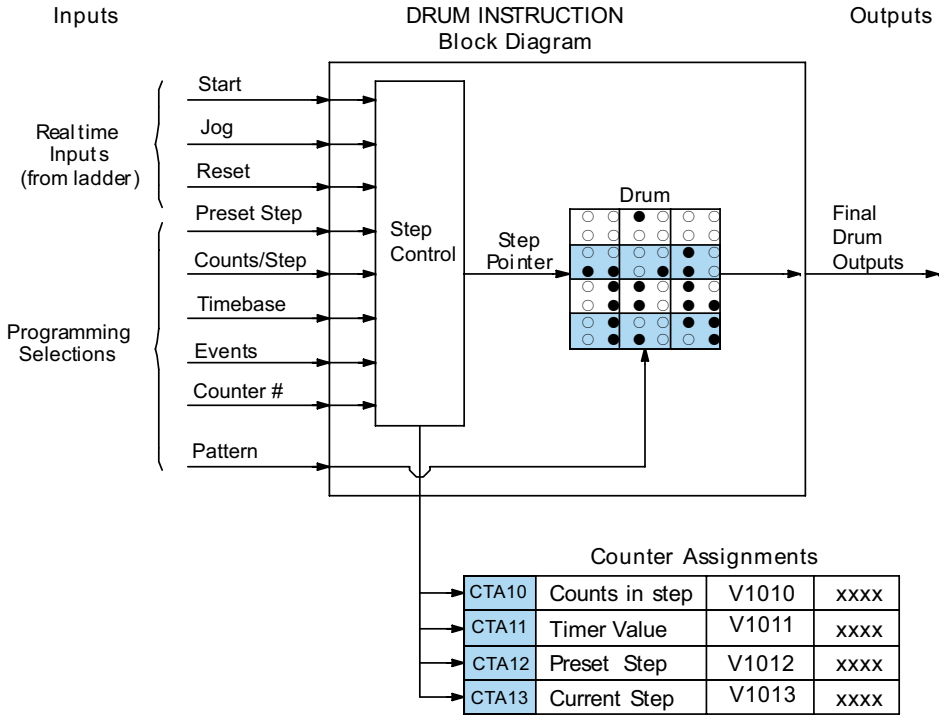
The last step in a drum sequence may be any step number, since partial drums are valid. Refer to the following figure. When the transition conditions of the last step are met, the drum sets the counter bit corresponding to the counter named in the **drum instruction** box (such as CT10). Then it moves to a final "**drum complete**" state. The drum outputs remain in the pattern defined for the last step. Having finished a drum cycle, the Start and Jog inputs have no effect at this point. The drum leaves the "**drum complete**" state when the Reset input becomes active (or on a program-to–run mode transition). It resets the **drum complete** bit (such as CT10), and then goes directly to the appropriate step number defined as the preset step.



Overview of Drum Operation

Drum Instruction Block Diagram

The **drum instruction** utilizes various inputs and outputs in addition to the drum pattern itself. Refer to the figure below.



The **drum instruction** accepts several inputs for step control, the main control of the drum. The inputs and their functions are:

- **Start** – The Start input is effective only when Reset is off. When Start is on, the drum timer runs if it is in a timed transition, and the drum looks for the input event during event transitions. When Start is off, the drum freezes in its current state (Reset must remain off), and the drum outputs maintain their current on/off pattern.
- **Jog** – The jog input is only effective when Reset is off (Start may be either on or off). The jog input increments the drum to the next step on each off-to-on transition (only EDRUM supports the jog input).
- **Reset** – The Reset input has priority over the Start input. When Reset is on, the drum moves to its preset step. When Reset is off, then the Start input operates normally.
- **Preset Step** – A step number from 1 to 16 that you define (typically is step 1). The drum moves to this step whenever Reset is on, and whenever the CPU first enters run mode.

- **Counts/Step** – The number of timer counts the drum spends in each step. Each step has its own counts parameter. However, programming the counts/step is optional.
- **Timer Value** – the current value of the counts/step timer.
- **Counter #** – The counter number specifies the first of four consecutive counters which the drum uses for step control. You can monitor these to determine the drum's progress through its control cycle. The DL06 has 128 counters (CT0 – CT177 in octal).
- **Events** – Either an X, Y, C, S, T, or CT type discrete point serves as step transition inputs. Each step has its own event. However, programming the event is optional.

WARNING: The outputs of a drum are enabled any time the CPU is in Run Mode. The Start Input does not have to be on, and the Reset input does not disable the outputs. Upon entering Run Mode, drum outputs automatically turn on or off according to the pattern of the current step of the drum. This initial step number depends on the counter memory configuration: non-retentive versus retentive.

Powerup State of Drum Registers

The choice of the starting step on powerup and program-to-run mode transitions are important to consider for your application. Please refer to the following chart. If the counter memory is configured as non-retentive, the drum is initialized the same way on every powerup or program-to-run mode transition. However, if the counter memory is configured to be retentive, the drum will stay in its previous state.

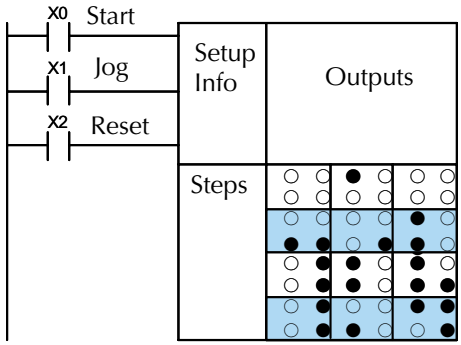
Counter Number	Function	Initialization on Powerup	
		Non-Retentive Case	Retentive Case
CTA(n)	Current Step Count	Initialize = 0	Use Previous (no change)
CTA(n + 1)	Counter Timer Value	Initialize = 0	Use Previous (no change)
CTA(n + 2)	Preset Step	Initialize = Preset Step #	Use Previous (no change)
CTA(n + 3)	Current Step #	Initialize = Preset Step #	Use Previous (no change)

Applications with relatively fast drum cycle times typically will need to be reset on powerup, using the non-retentive option. Applications with relatively long drum cycle times may need to resume at the previous point where operations stopped, using the retentive case. The default option is the retentive case. This means that if you initialize scratchpad V-memory, the memory will be retentive.

Drum Control Techniques

Drum Control Inputs

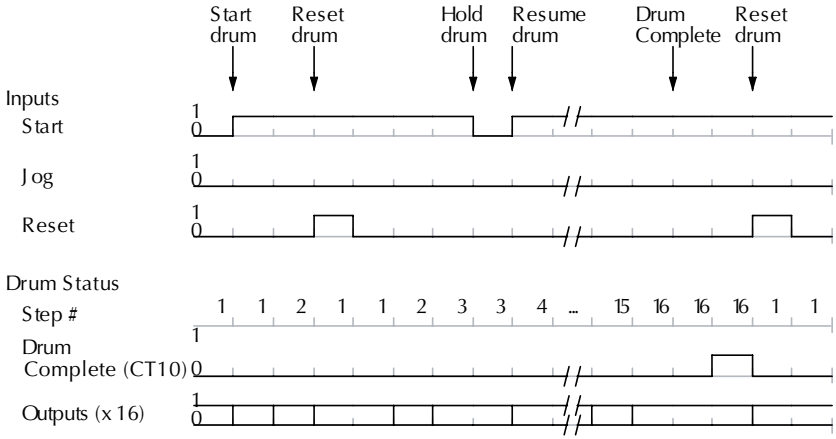
Now we are ready to put together the concepts on the previous pages and demonstrate general control of the drum instruction box. The drawing to the right shows a simplified generic drum instruction. Inputs from ladder logic control the Start, Jog, and Reset Inputs (only the EDRUM instruction supports the Jog Input). The first counter bit of the drum (CT10, for example) indicates the drum cycle is done.



The timing diagram below shows an arbitrary timer drum input sequence and how the drum responds. As the CPU enters Run mode it initializes the step number to the preset step number (typically it is Step 1). When the Start input turns on the drum begins running, waiting for an event and/or running the timer (depends on the setup).

After the drum enters Step 2, Reset turns On while Start is still On. Since Reset has priority over Start, the drum goes to the preset step (Step 1). Note that the drum is *held* in the preset step during Reset, and that step does *not run* (respond to events or run the timer) until Reset turns off.

After the drum has entered step 3, the Start input goes off momentarily, halting the drum's timer until Start turns on again.

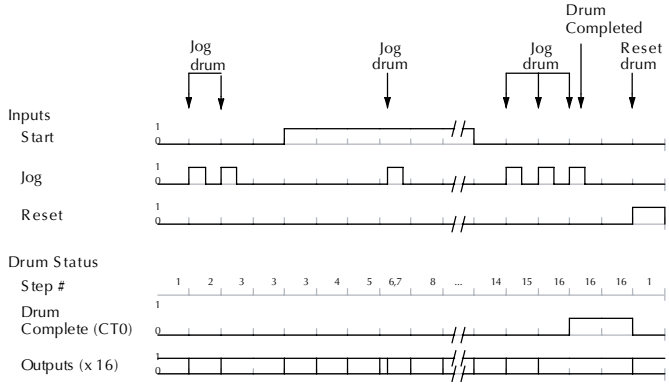


When the drum completes the last step (Step 16 in this example), the Drum Complete bit (CT10) turns on, and the step number remains at 16. When the Reset input turns on, it turns off the Drum Complete bit (CT10), and forces the drum to enter the preset step.

***NOTE**: The timing diagram shows all steps using equal time durations. Step times can vary greatly, depending on the counts/step programmed.*

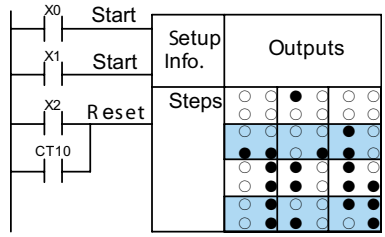
In the figure below, we focus on how the Jog input works on event drums. To the left of the diagram, note that the off-to-on transitions of the Jog input increments the step. Start may be either on or off (however, Reset must be off). Two jogs takes the drum to step three. Next, the Start input turns on, and the drum begins running normally. During step 6 another Jog input signal occurs. This increments the drum to step 7, setting the timer to 0. The drum begins running immediately in step 7, because Start is already on. The drum advances to step 8 normally.

As the drum enters step 14, the Start input turns off. Two more Jog signals moves the drum to step 16. However, note that a third Jog signal is required to move the drum through step 16 to "**drum complete**". Finally, a **Reset** input signal arrives which forces the drum into the preset step and turns off the drum complete bit.



Self-Resetting Drum

Applications often require drums that automatically start over once they complete a cycle. This is easily accomplished, using the drum complete bit. In the figure to the right, the drum instruction setup is for CT10, so we logically OR the drum complete bit (CT10) with the Reset input. When the last step is done, the drum turns on CT10 which resets itself to the preset step, also resetting CT10. Contact X2 still works as a manual reset.



Initializing Drum Outputs

The outputs of a drum are enabled any time the CPU is in run mode. On program-to-run mode transitions, the drum goes to the preset step, and the outputs energize according to the pattern of that step. If your application requires all outputs to be off at powerup, make the preset step in the drum a "**reset step**", with all outputs off.

Using Complex Event Step Transitions

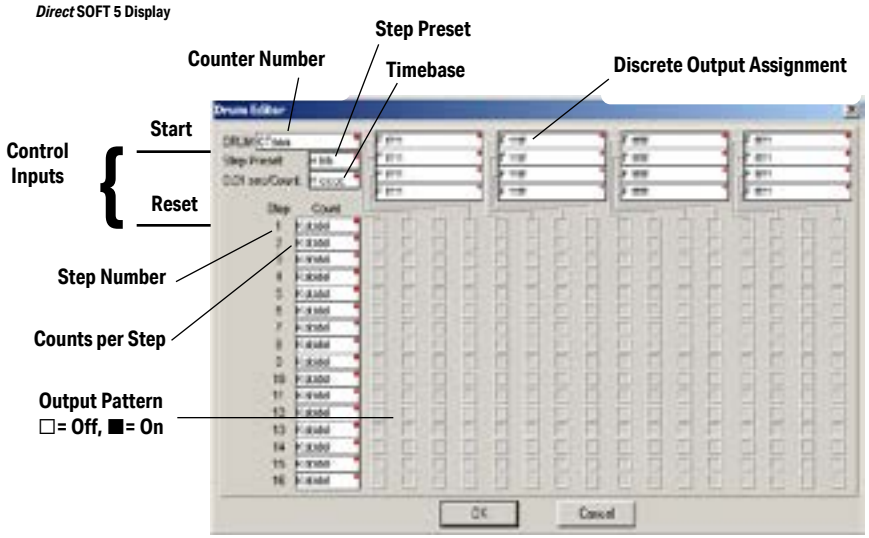
Each event-based transition accepts only one contact reference for the event. However, this does not limit events to just one contact. Just use a control relay contact such as C0 for the step transition event. Elsewhere in ladder logic, you may use C0 as an output coil, making it dependent on many other events (contacts).

Drum Instruction

The DL06 drum instructions may be programmed using *Direct*SOFT, or for the EDRUM instruction only you can use a handheld programmer (firmware version v2.21 or later). This section covers entry using *Direct*SOFT for all instructions plus the handheld mnemonics for the EDRUM instruction.

Timed Drum with Discrete Outputs (DRUM)

The Timed Drum with Discrete Outputs is the most basic of the DL06's drum instructions. It operates according to the principles covered on the previous pages. Below is the instruction in chart form as displayed by *Direct*SOFT.



The Timed Drum features 16 steps and 16 outputs. Step transitions occur only on a timed basis, specified in counts per step. Unused steps must be programmed with "counts per step"=0 (this is the default entry). The discrete output points may be individually assigned as X, Y, or C types, or may be left unused. The output pattern may be edited graphically with *Direct*SOFT.

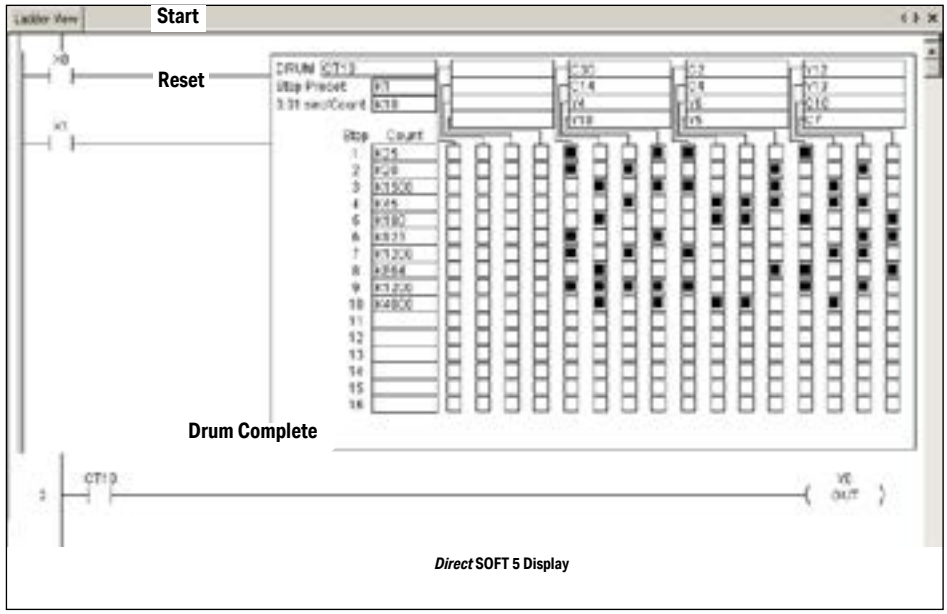
Whenever the Start input is energized, the drum's timer is enabled. It stops when the last step is complete, or when the Reset input is energized. The drum enters the preset step chosen upon a CPU program-to-run mode transition, and whenever the Reset input is energized.

Drum Parameters	Field	Data Types	Ranges
Counter Number	aaa --	0 --174	
Preset Step	bb	K	1 -- 16
Timer base	cccc	K	0 -- 99.99 seconds
Counts per step	dddd	K	0 -- 9999
Discrete Outputs	Fffff	X, Y, C	see memory map

Drum instructions use four counters in the CPU. The ladder program can read the counter values for the drum's status. The ladder program may write a new preset step number to CTA(n+2) at any time. However, the other counters are for monitoring purposes only.

Counter Number	Ranges of (n)	Function	Counter Bit Function
CTA(n)	0 -- 174	Counts in step	CT(n) = Drum Complete
CTA(n+1)	1 -- 175	Timer value	CT(n+1) = (not used)
CTA(n+2)	2 -- 176	Preset Step	CT(n+2) = (not used)
CTA(n+3)	3 -- 177	Current Step	CT(n+3) = (not used)

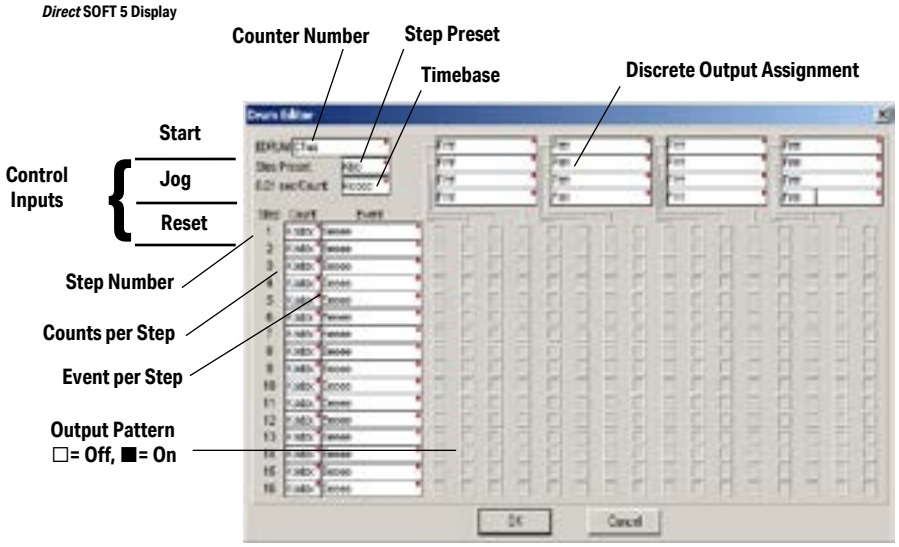
The following ladder program shows the DRUM instruction in a typical ladder program, as shown by ***Direct***SOFT. Steps 1 through 10 are used, and twelve of the sixteen output points are used. The preset step is step 1. The timebase runs at (K10 x 0.01) = 0.1 second per count. Therefore, the duration of step 1 is (25 x 0.1) = 2.5 seconds. In the last rung, the Drum Complete bit (CT10) turns on output Y0 upon completion of the last step (step 10). A drum reset also resets CT10.



Direct **SOFT 5 Display**

Event Drum (EDRUM)

The Event Drum (EDRUM) features time-based and event-based step transitions. It operates according to the general principles of drum operation covered in the beginning of this chapter. Below is the instruction as displayed by *Direct*SOFT.



The Event Drum features 16 steps and 16 discrete outputs. Step transitions occur on timed and/or event basis. The jog input also advances the step on each off-to-on transition. Time is specified in counts per step, and events are specified as discrete contacts. Unused steps and events must be left blank. The discrete output points may be individually assigned.

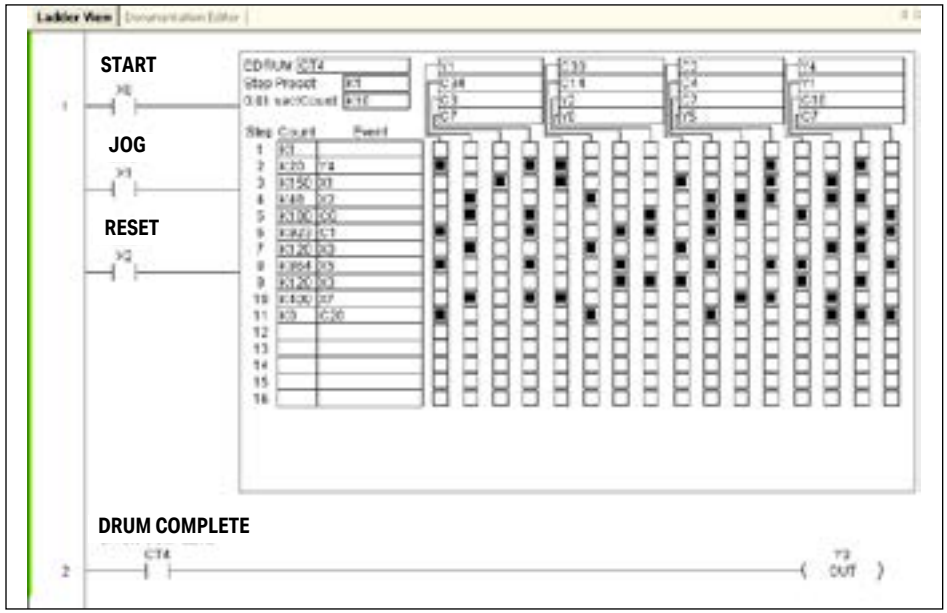
Drum Parameters	Field	Data Types	Ranges
Counter Number	aa	--	0 -- 174
Preset Step	bb	K	1 -- 16
Timer base	cccc	K	0 -- 99.99 seconds
Counts per step	dddd	K	0 -- 9999
Event	Eeeee	X, Y, C, S, T, CT, SP	see memory map
Discrete Outputs	ffff	X, Y, C	see memory map

Whenever the Start input is energized, the drum's timer is enabled. As long as the event is true for the current step, the timer runs during that step. When the step count equals the counts per step, the drum transitions to the next step. This process stops when the last step is complete, or when the Reset input is energized. The drum enters the preset step chosen upon a CPU program-to-run mode transition, and whenever the Reset input is energized.

Drum instructions use four counters in the CPU. The ladder program can read the counter values for the drum's status. The ladder program may write a new preset step number to CTA(n+2) at any time. However, the other counters are for monitoring purposes only.

Counter Number	Ranges of (n)	Function	Counter Bit Function
CTA(n)	0 -- 174	Counts in step	CT(n)= Drum Complete
CTA(n+1)	1 -- 175	Timer value	CT(n+1) = (not used)
CTA(n+2)	2 -- 176	Preset Step	CT(n+2) = (not used)
CTA(n+3)	3 -- 177	Current Step	CT(n+3) = (not used)

The following ladder program shows the EDRUM instruction in a typical ladder program, as shown by ***Direct***SOFT. Steps 1 through 11 are used, and all sixteen output points are used. The preset step is step 1. The timebase runs at (K10 x 0.01) = 0.1 second per count. Therefore, the duration of step 1 is (1 x 0.1) = 0.1 second. Note that step 1 is time-based only (event is left blank). And, the output pattern for step 1 programs all outputs off, which is a typically desirable powerup condition. In the last rung, the Drum Complete bit (CT4) turns on output Y0 upon completion of the last step (step 11). A drum reset also resets CT4.

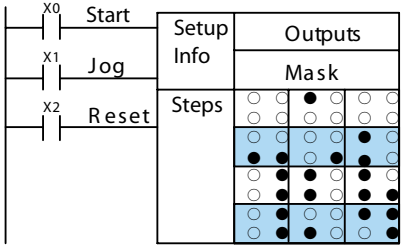


Handheld Programmer Drum Mnemonics

The EDRUM instruction can also be programmed using a handheld programmer. This section explains entry via the handheld programmer.

First, enter Store instructions for the ladder rung controlling the drum's ladder inputs. In the exampl to the right, the timer drum's Start, Jog, and Rese inputs are controlled by X0, X1 and X2 respectively The required keystrokes are listed beside th mnemonic.

These keystrokes precede the EDRUM instructio mnemonic. Note that the ladder rungs for Start, Jog and Reset inputs are not limited to being single-contact rungs.



Handheld Programmer Keystrokes

Store X0 | $ STR | → | A 0 | ENT

(Repeat for Store X1 and Store X2)

Handheld Programmer Keystrokes

EDRUM CNT4 | SHFT | E 4 | D 3 | R ORN | U ISG | M ORST | → | E 4 | ENT

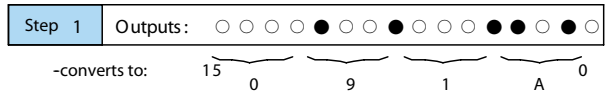
After the Store instructions, enter the EDRUM (using Counter CT0) as shown:

After entering the EDRUM mnemonic as above, the handheld programmer creates an input form for all the drum parameters. The input form consists of approximately fifty or more default mnemonic entries containing DEF (define) statements. The default mnemonics are already input for you, so they appear automatically. Use the NXT and PREV keys to move forward and backward through the form. Only the editing of default values is required, thus eliminating many keystrokes. The entries required for the basic timer drum are in the chart below.

***NOTE**: Default entries for output points and events are "**DEF 0000",** which means they are unassigned. If you need to go back and change an assigned output as unused again, enter "**K0000"**. The entry will again show as "**DEF 0000"**.*

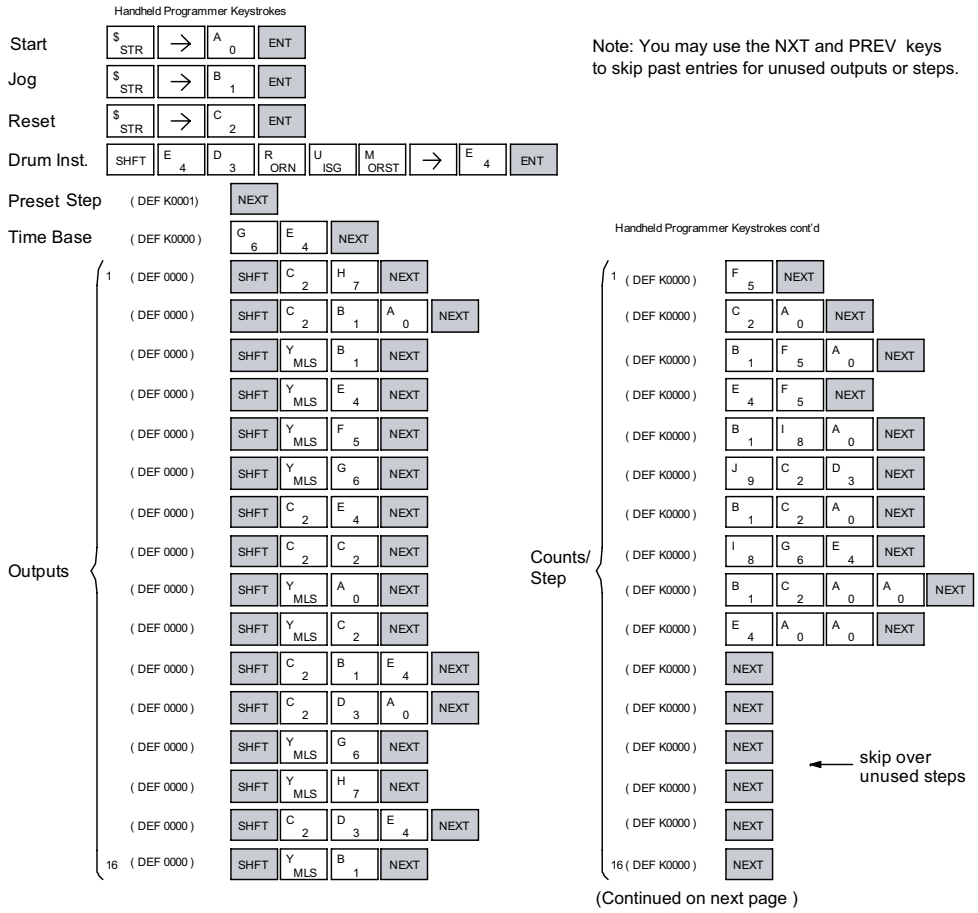
Drum Parameters	Multiple Entries	Mnemonic / Entry	Default Mnemonic	Valid Data Types	Ranges
Start Input	--	STR (plus input rung)	--	--	--
Jog Input	--	STR (plus input rung)	--	--	--
Reset Input	--	STR (plus input rung)	--	--	--
Drum Mnemonic	--	DRUM CNT aa	--	CT	0 -- 174
Preset Step	1	bb	DEF K0000	K	1 -- 16
Timer base	1	cccc	DEF K0000	K	1 -- 9999
Output points	16	ffff	DEF 0000	X, Y, C	see memory map
Counts per step	16	dddd	DEF K0000	K	0 -- 9999
Events	16	dddd	DEF K0000	X, Y, C, S, T, CT, SP	see memory map
Output pattern	16	gggg	DEF K0000	K	0 -- FFFF

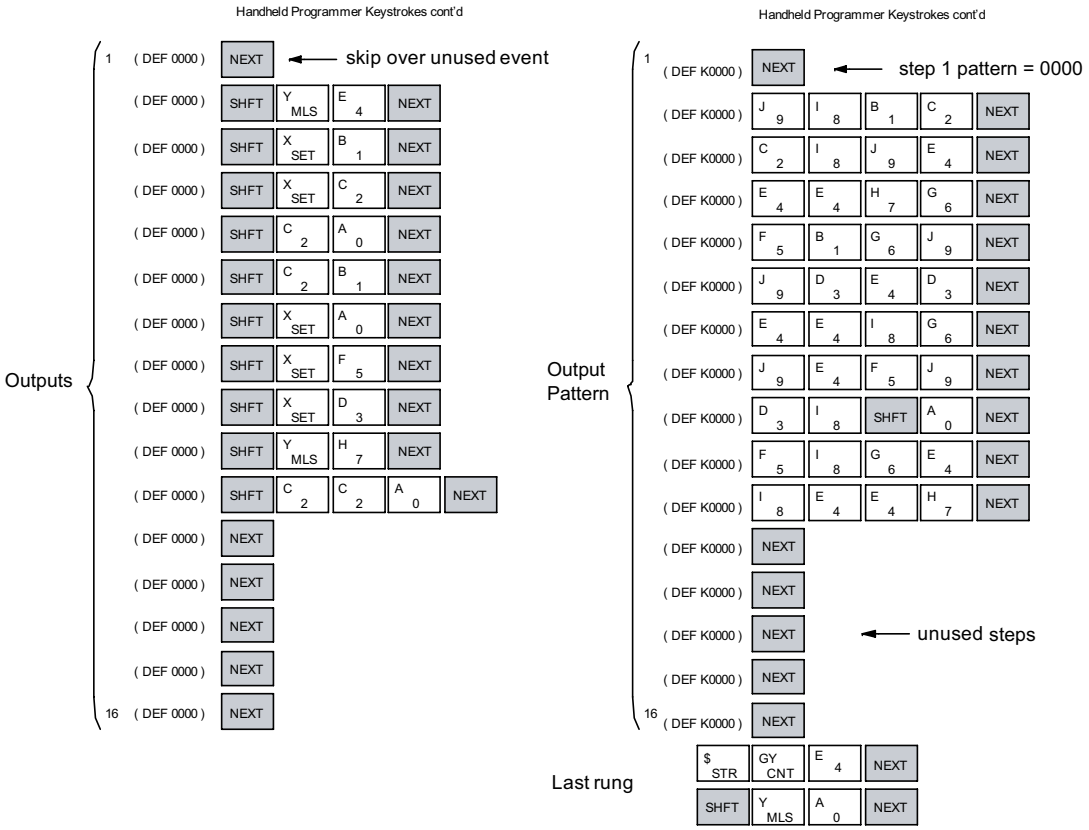
Using the DRUM entry chart (two pages before), we show the method of entry for the basic time/event drum instruction. First, we convert the output pattern for each step to the equivalent hex number, as shown in the following example.



The following diagram shows the method for entering the previous EDRUM example on the HHP. The default entries of the form are in parenthesis. After the drum instruction entry (on the fourth row), the remaining keystrokes over-write the numeric portion of each default DEF statement. **NOTE**: Drum editing requires Handheld Programmer firmware version 2.21 or later.

***NOTE**: You may use the NXT and PREV keys to skip past entries for unused outputs or steps.*



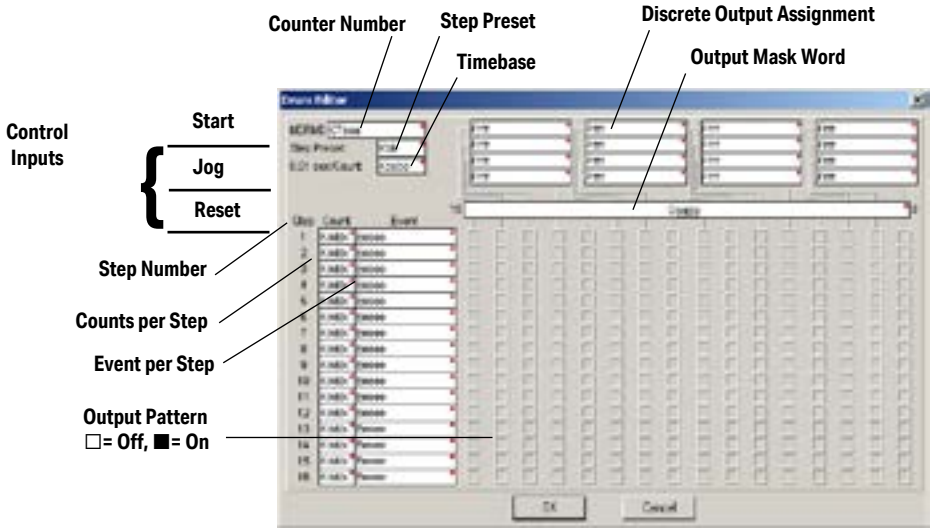


***NOTE**: You may use the NXT and PREV keys to skip past entries for unused outputs or steps.*

***NOTE:** For ease of operation when using the EDRUM instruction, we recommend using **Direct**SOFT over the handheld programmer.*

Masked Event Drum with Discrete Outputs (MDRMD)

The Masked Event Drum with Discrete Outputs has all the features of the basic Event Drum plus final output control for each step. It operates according to the general principles of drum operation covered in the beginning of this section. Below is the instruction in chart form as displayed by *Direct*SOFT.



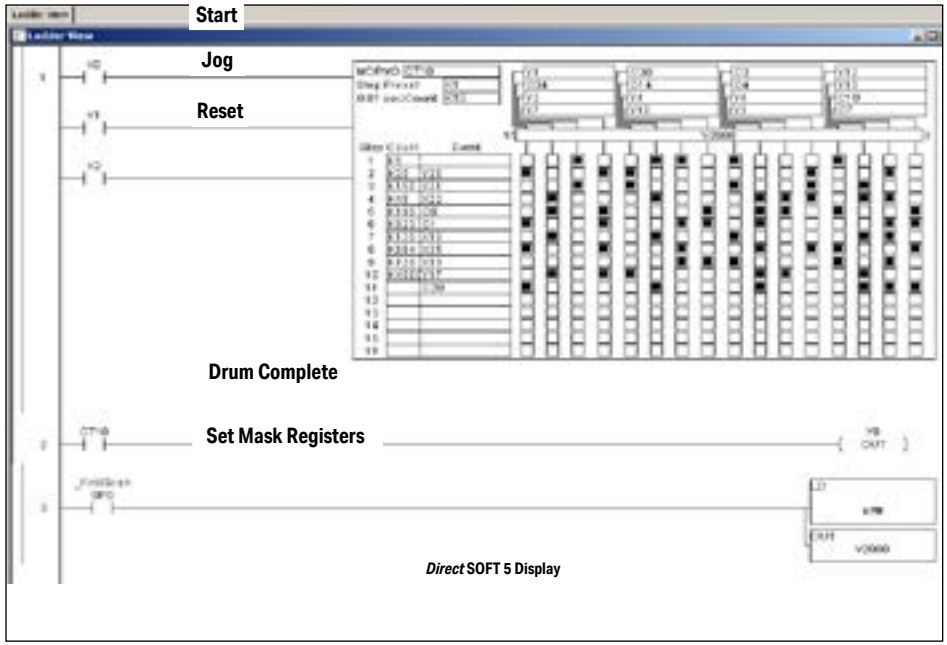
The Masked Event Drum with Discrete Outputs features sixteen steps and sixteen outputs. Drum outputs are logically ANDed bit-by-bit with an output mask word for each step. The Ggggg field specifies the beginning location of the 16 mask words. Step transitions occur on timed and/or event basis. The jog input also advances the step on each off-to-on transition. Time is specified in counts per step, and events are specified as discrete contacts. Unused steps and events can be left blank (this is the default entry). Whenever the Start input is energized, the drum's timer is enabled. As long as the event is true for the current step, the timer runs during that step. When the step count equals the counts per step, the drum transitions to the next step. This process stops when the last step is complete, or when the Reset input is energized. The drum enters the preset step chosen upon a CPU program-to-run mode transition, and whenever the Reset input is energized.

Drum Parameters	Field	Data Types	Ranges
Counter Number	aaa	-	0 – 174
Preset Step	bb	K	1 – 16
Timer base	cccc	K	0 – 99.99 seconds
Counts per step	dddd	K	0 – 9999
Event	eeee	X, Y, C, S, T, ST, GX, GY. CT, SP	see memory map
Discrete Outputs	Fffff	X, Y, C, GX, GY	
Output Mask	Ggggg	V	

Drum instructions use four counters in the CPU. The ladder program can read the counter values for the drum's status. The ladder program may write a new preset step number to CTA(n+2) at any time. However, the other counters are for monitoring purposes only.

Counter Number	Ranges of (n)	Function	Counter Bit Function
CTA(n)	0 – 174	Counts in step	CT(n) = Drum Complete
CTA(n+1)	1 – 175	Timer value	CT(n+1) = (not used)
CTA(n+2)	2 –176	Preset Step	CT(n+2) = (not used)
CTA(n+3)	3 –177	Current Step	CT(n+1) = (not used)

The following ladder program shows the MDRMD instruction in a typical ladder program, as shown by ***Direct***SOFT. Steps 1 through 11 are used, and all 16 output points are used. The output mask word is at V2000. The final drum outputs are shown above the mask word as individual bits. The data bits in V2000 are logically ANDed with the output pattern of the current step in the drum. If you want all drum outputs to be off after powerup, write zeros to V2000 on the first scan. Ladder logic may update the output mask at any time to enable or disable the drum outputs The preset step is step 1. The timebase runs at (K10 x 0.01)=0.1 second per count. Therefore, the duration of step 1 is (5 x 0.1) = 0.5 seconds. Note that step 1 is time-based only (event is left blank). In the last rung, the Drum Complete bit (CT10) turns on output Y0 upon completion of the last step (step 10). A drum reset also resets CT10.

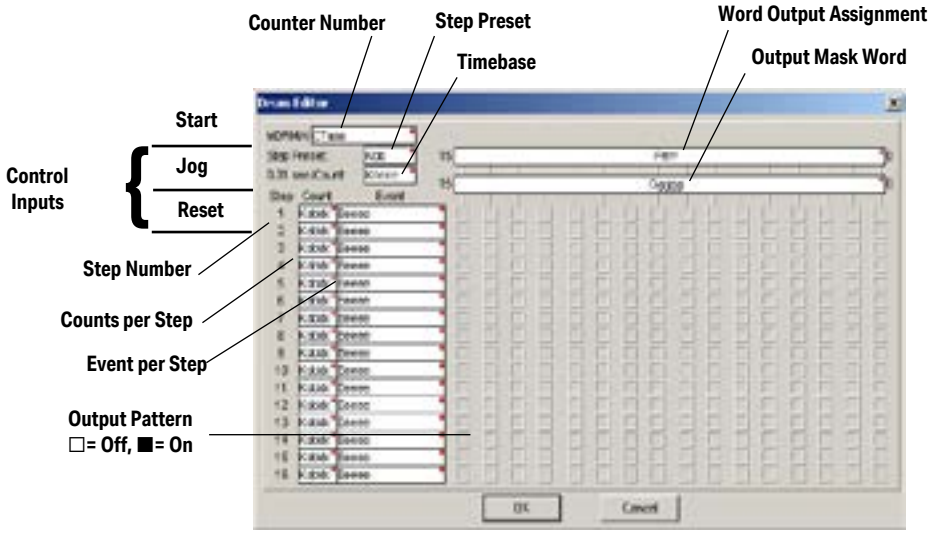


Direct **SOFT 5 Display**

NOTE*: The ladder program must load constants in V2000 through V2012 to cover all mask registers for the eleven steps used in this drum.*

Masked Event Drum with Word Output (MDRMW)

The Masked Event Drum with Word Output features outputs organized as bits of a single word, rather than discrete points. It operates according to the general principles of drum operation covered in the beginning of this section. Below is the instruction in chart form as displayed by *Direct*SOFT.



The Masked Event Drum with Word Output features sixteen steps and sixteen outputs. Drum outputs are logically ANDed bit-by-bit with an output mask word for each step. The Ggggg field specifies the beginning location of the 16 mask words, creating the final output (Fffff field). Step transitions occur on timed and/or event basis. The jog input also advances the step on each off-to-on transition. Time is specified in counts per step, and events are specified as discrete contacts. Unused steps and events can be left blank (this is the default entry).

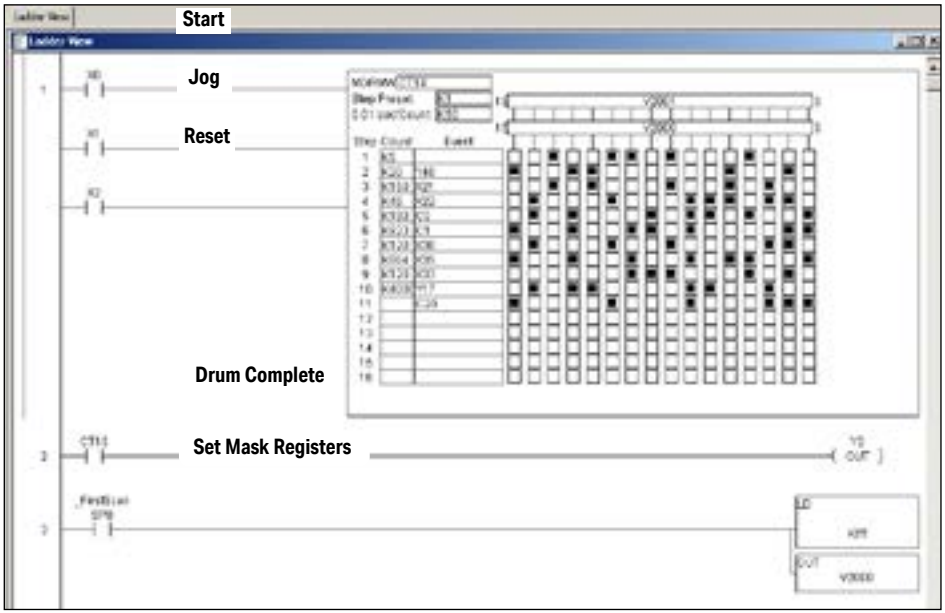
Whenever the Start input is energized, the drum's timer is enabled. As long as the event is true for the current step, the timer runs during that step. When the step count equals the counts per step, the drum transitions to the next step. This process stops when the last step is complete, or when the Reset input is energized. The drum enters the preset step chosen upon a CPU program-to-run mode transition, and whenever the Reset input is energized.

Drum Parameters	Field	Data Types	Ranges
Counter Number	aaa	–	0 – 174
Preset Step	bb	K	1 – 16
Timer base	cccc	K	0 – 99.99 seconds
Counts per step	dddd	K	0 – 9999
Event	eeee	X, Y, C, S, T, ST, GX, GY, SP	see memory map
Word Output	Fffff	V	see memory map
Output Mask	Ggggg	V	see memory map

Drum instructions use four counters in the CPU. The ladder program can read the counter values for the drum's status. The ladder program may write a new preset step number to CTA(n+2) at any time. However, the other counters are for monitoring purposes only.

Counter Number	Ranges of (n)	Function	Counter Bit Function
CTA(n)	0 – 174	Counts in step	CT(n) = Drum Complete
CTA(n+1)	1 – 175	Timer value	CT(n+1) = (not used)
CTA(n+2)	2 –176	Preset Step	CT(n+2) = (not used)
CTA(n+3)	3 –177	Current Step	CT(n+1) = (not used)

The following ladder program shows the MDRMD instruction in a typical ladder program, as shown by ***Direct***SOFT. Steps 1 through 11 are used, and all sixteen output points are used. The output mask word is at V2000. The final drum outputs are shown above the mask word as a word at V2001. The data bits in V2000 are logically ANDed with the output pattern of the current step in the drum, generating the contents of V2001. If you want all drum outputs to be off after powerup, write zeros to V2000 on the first scan. Ladder logic may update the output mask at any time to enable or disable the drum outputs. The preset step is step 1. The timebase runs at (K50 x 0.01)=0.5 seconds per count. Therefore, the duration of step 1 is (5 x 0.5) = 2.5 seconds. Note that step 1 is time-based only (event is left blank). In the last rung, the Drum Complete bit (CT14) turns on output Y0 upon completion of the last step (step 10). A drum reset also resets CT14.



***NOTE**: The ladder program must load constants in V2000 through V2012 to cover all mask registers for the eleven steps used in this drum.*

RLL^{PLUS} STAGE PROGRAMMING

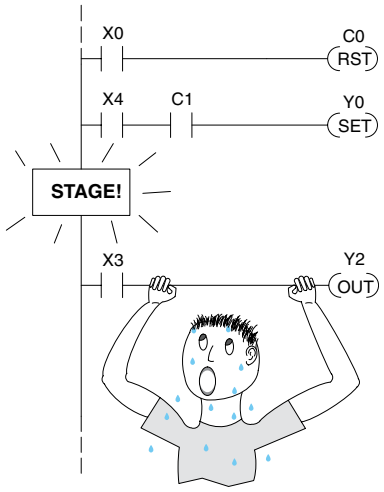
In This Chapter...

Introduction to Stage Programming

Stage Programming provides a way to organize and program complex applications with relative ease, when compared to purely relay ladder logic (RLL) solutions. Stage programming does not replace or negate the use of traditional boolean ladder programming. This is why Stage Programming is also called RLL^{PLUS} You won't have to discard any training or experience you already have. Stage programming simply allows you to divide and organize an RLL program into groups of ladder instructions called stages. This allows quicker and more intuitive ladder program development than traditional RLL alone provides.

Overcoming "Stage Fright"

Many PLC programmers in the industry have become comfortable using RLL for every PLC program they write, but often remain skeptical or even fearful of learning new techniques such as stage programming. While RLL is great at solving boolean logic relationships, it has disadvantages as well:



- Large programs can become almost unmanageable, because of a lack of structure.
- When a process gets stuck, it is difficult to find the rung where the error occurred.
- Programs become difficult to modify later, because they do not intuitively resemble the application problem they are solving.

It's easy to see that these inefficiencies consume a lot of additional time, and time is money. *Stage programming overcomes these obstacles!* We believe a few moments of studying the stage concept is one of the greatest investments in programming speed and efficiency a PLC programmer can make!

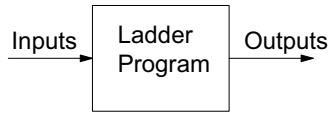
So, we encourage you to study stage programming and add it to your toolbox of programming techniques. This chapter is designed as a self-paced tutorial on stage programming. For best results:

- Start at the beginning and do not skip over any sections.
- Study each stage programming concept by working through each example. The examples build progressively on each other.
- Read the Stage Questions and Answers at the end of the chapter for a quick review.

Learning to Draw State Transition Diagrams

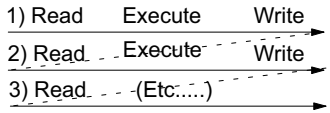
Introduction to Process States

Those familiar with ladder program execution know that the CPU must scan the ladder program repeatedly, over and over. Its three basic steps are:



1. Read the inputs
2. Execute the ladder program
3. Write the outputs

The benefit is that a change at the inputs can affect the outputs in just a few milliseconds.



Most manufacturing processes consist of a series of activities or conditions, each lasting for several seconds, minutes, or even hours. We might call these *process states*, which are either active or inactive at any particular time. A challenge for RLL programs is that a particular input event may last for just a brief instant. We typically create latching relays in RLL to preserve the input event in order to maintain a process state for the required duration.

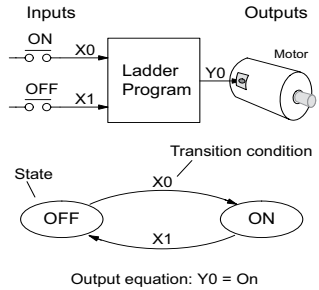
We can organize and divide ladder logic into sections called **stages**, representing process states. But before we describe stages in detail, we will reveal **the secret to understanding stage programming:** state transition diagrams.

The Need for State Diagrams

Sometimes we need to forget about the scan nature of a PLC, and focus our thinking toward the states of the process we need to identify. Clear thinking and concise analysis of an application gives us the best chance at writing efficient, bug-free programs. *State diagrams are just a tool to help us draw a picture of our process!* You'll discover that if we can get the picture right, **our program will also be right!**

A 2–State Process

Consider the simple process shown to the right, which controls an industrial motor. We will use a green momentary SPST pushbutton to turn the motor on, and a red one to turn it off. The machine operator will press the appropriate pushbutton for just a second or so. The two states of our process are ON and OFF.

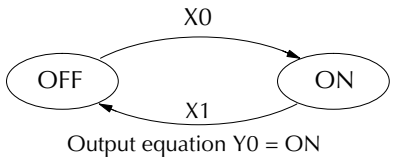


The next step is to draw a state transition diagram, as shown to the right. It shows the two states OFF and ON, with two transition lines in-between. When the event X0 is true, we transition from OFF to ON. When X1 is true, we transition from ON to OFF.

If you're following along, you are very close to grasping the concept and the problem-solving power of state transition diagrams. The output of our controller is Y0, which is true any time we are in the ON state. In a boolean sense, Y0=ON state.

Next, we will implement the state diagram first as RLL, then as a stage program. This will help you see the relationship between the two methods in problem solving.

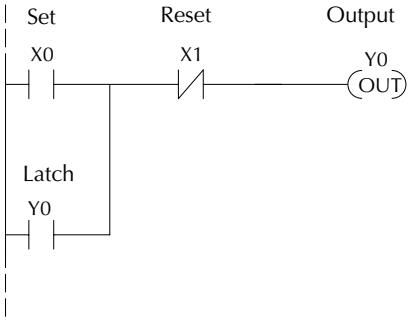
The state transition diagram to the right is a picture of the solution we need to create. The beauty of it is this: it expresses the problem independently of the programming language we may use to realize it. In other words, *by drawing the diagram we have already solved the control problem!*



First, we'll translate the state diagram to traditional RLL. Then, we'll show how easy it is to translate the diagram into a stage programming solution.

RLL Equivalent

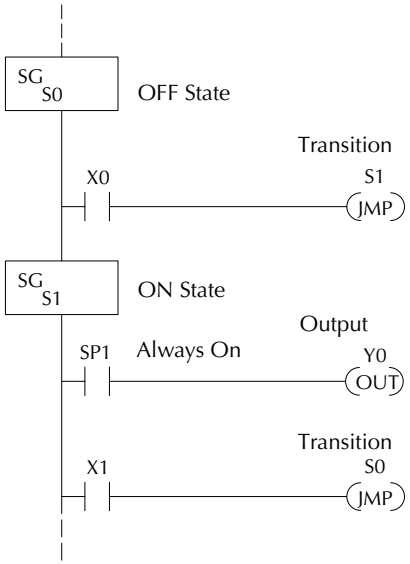
The RLL solution is shown to the right. Output control relay, Y0, has a dual purpose. It turns the motor on and off and acts as a latching relay. When the On pushbutton (X0) is pressed, output coil Y0 turns on and the Y0 contact on the second row latches itself on. So, X0 turns on the motor output Y0 which now has power flow and **sets the latch** Y0. It will remain on after the X0 contact opens.



When the Off pushbutton (X1) is pressed, it opens the normally-closed X1 contact, which turns off motor output Y0 and also **resets the latch**.

Stage Equivalent

The stage program solution is shown to the right. The two inline stage boxes S0 and S1 correspond to the two states OFF and ON. The ladder rung(s) below each stage box belong to each respective stage. This means that the PLC only has to scan those rungs when the corresponding stage is active!



For now, let's assume we begin in the OFF State, so stage S0 is active. When the On pushbutton (X0) is pressed, a stage transition occurs. The JMP S1 instruction executes, which simply turns off the Stage bit S0 and turns on Stage bit S1. So on the next PLC scan, the CPU will not execute Stage S0, but will execute stage S1!

In the On State (Stage S1), we want the motor to always be on. The special relay contact SP1 is defined as always on, so Y0 turns the motor on.

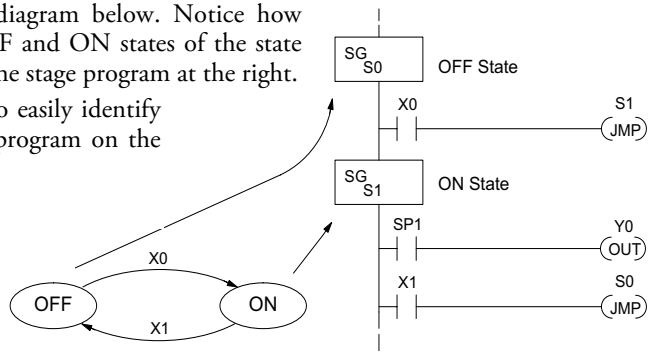
When the Off pushbutton (X1) is pressed, a transition back to the Off State occurs. The JMP S0 instruction executes, which simply turns off the Stage bit S1 and turns on Stage bit S0. On the next PLC scan, the CPU will not execute Stage S1, so the motor output Y0 will turn off. The Off state (Stage 0) will be ready for the next cycle.

Let's Compare

Right now, you may be thinking, "I don't see the big advantage to Stage Programming ... in fact, the stage program is longer than the plain RLL program." Well, now is the time to exercise a bit of faith. As control problems grow in complexity, stage programming quickly out-performs RLL in simplicity, program size, etc.

For example, consider the diagram below. Notice how easy it is to correlate the OFF and ON states of the state transition diagram below to the stage program at the right.

Now, we challenge anyone to easily identify the same states in the RLL program on the previous page!

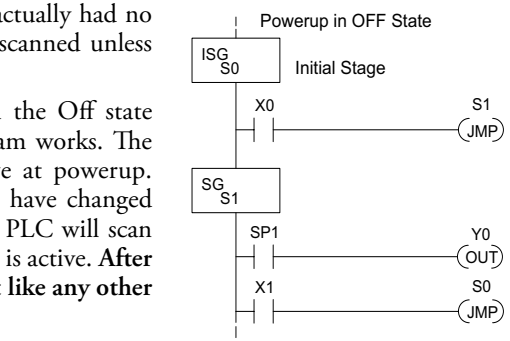


Initial Stages

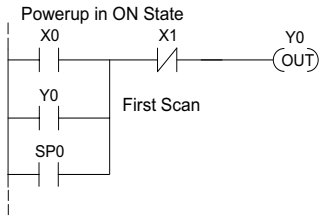
At powerup and Program-to-Run Mode transitions, the PLC always begins with all normal stages (SG) off.

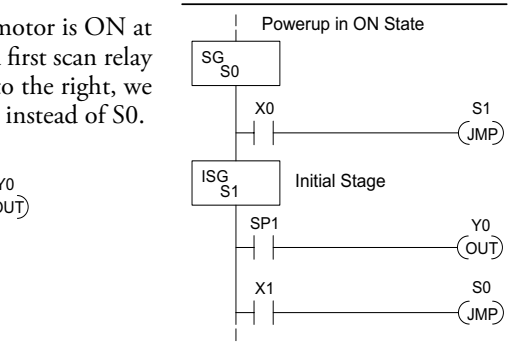
So, the stage programs shown so far have actually had no way to get started (because rungs are not scanned unless their stage is active).

Assume that we want to always begin in the Off state (motor off), which is how the RLL program works. The Initial Stage (ISG) is defined to be active at powerup. In the modified program to the right, we have changed stage S0 to the ISG type. This ensures the PLC will scan contact X0 after powerup, because Stage S0 is active. **After powerup, an Initial Stage (ISG) works just like any other stage!**



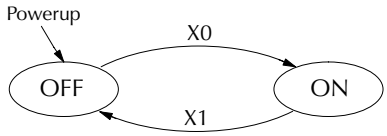
We can change both programs so that the motor is ON at powerup. In the RLL below, we must add a first scan relay SP0, latching Y0 on. In the stage example to the right, we simply make Stage S1 an initial stage (ISG) instead of S0.





NOTE: *If the ISG is within the retentive range for stages, the ISG will remain in the state it was in before power down and will NOT turn itself on during the first scan.*

We can mark our desired powerup state as shown to the right, which helps us remember to use the appropriate Initia Stages when creating a stage program. It is permissible to have as many initial stages as the process requires.

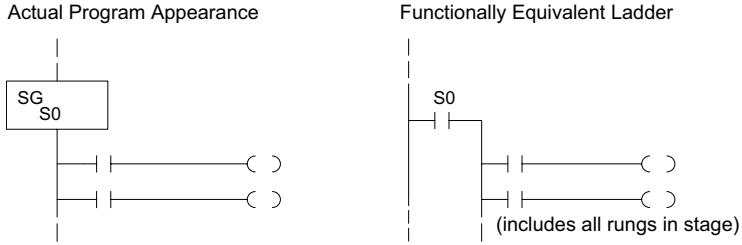


What Stage Bits Do

You may recall that a stage is just a section of ladder program which is either active or inactive at a given moment. All stage bits (S0 to 1777) reside in the PLC's image register as individual status bits. Each stage bit is either a boolean 0 or 1 at any time.

Program execution always reads ladder rungs from top to bottom, and from left to right. The drawing below shows the effect of stage bit status. The ladder rungs below the stage instruction continuing until the next stage instruction or the end of program belong to stage 0. Its equivalent operation is shown on the right. When S0 is true, the two rungs have power flow.

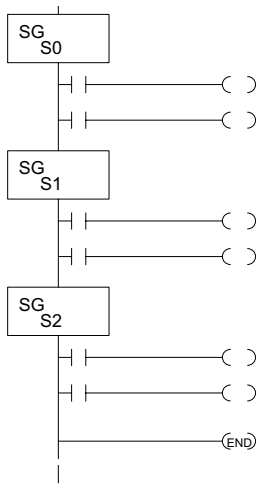
- If Stage bit S0 = 0, its ladder rungs are not scanned (executed).
- If Stage bit S0 = 1, its ladder rungs are scanned (executed).



Stage Instruction Characteristics

The inline stage boxes on the left power rail divide the ladder program rungs into stages. Some stage rules are:

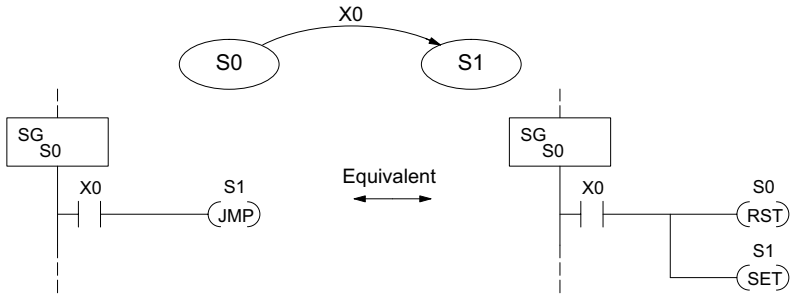
1. **Execution** – Only logic in active stages are executed on any scan.
2. **Transitions** – Stage transition instructions take effect on the next occurrence of the stages involved.
3. **Octal numbering** – Stages are numbered in octal, like I/O points, etc. So "S8" is not valid.
4. **Total Stages** – The DL06 offers up to 1024 stages (S0 to 1777 in octal).
5. **No duplicates** –Each stage number is unique and can be used just once.
6. **Any order** – You can skip numbers and sequence the stage numbers in any order.
7. **Last Stage** – The last stage in the ladder program includes all rungs from its stage box until the end coil.



Using the Stage Jump Instruction for State Transitions

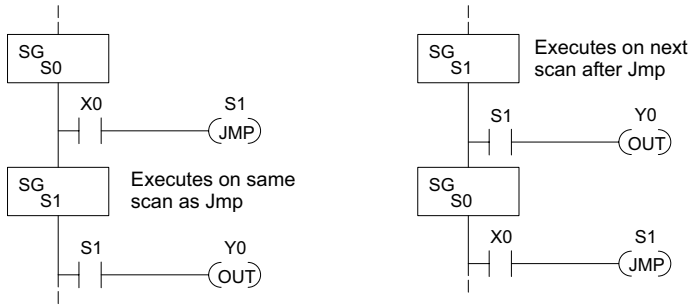
Stage Jump, Set, and Reset Instructions

The Stage JMP instruction we have used deactivates the stage in which the instruction occurs, while activating the stage in the JMP instruction. Refer to the state transition shown below. When contact X0 energizes, the state transition from S0 to S1 occurs. The two stage examples shown below are equivalent. So, the Stage Jump instruction is equal to a Stage Reset of the current stage, plus a Stage Set instruction for the stage to which we want to transition.



Please Read Carefully – The jump instruction is easily misunderstood. The "jump" does not occur immediately like a GOTO or GOSUB program control instruction when executed. Here's how it works:

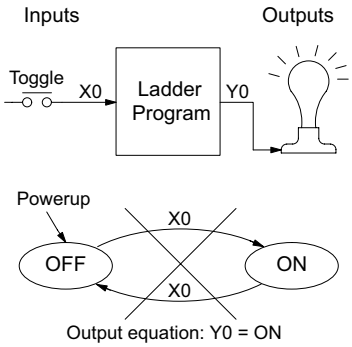
- The jump instruction resets the stage bit of the stage in which it occurs. All rungs in the stage still finish executing during the current scan, *even if there are other rungs in the stage below the jump instruction!*
- The reset will be in effect on the following scan, so the stage that executed the jump instruction previously will be inactive and bypassed.
- The stage bit of the stage named in the Jump instruction will be set immediately, so the stage will be executed on its next occurrence. In the left program shown below, stage S1 executes during the same scan as the JMP S1 occurs in S0. In the example on the right, Stage S1 executes on the next scan after the JMP S1 executes, because stage S1 is located above stage S0.



***NOTE**: Assume we start with Stage 0 active and Stage 1 inactive for both examples.*

Stage Program Example: Toggle On/Off Lamp Controller

A 4–State Process

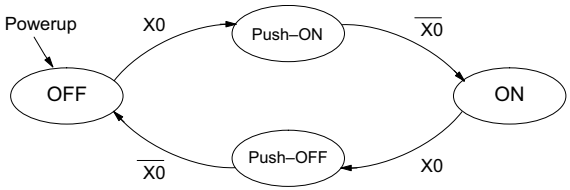


In the process shown to the right, we use an ordinary momentary pushbutton to control a light bulb. The ladder program will latch the switch input, so that we will push and release to turn on the light, push and release again to turn it off (sometimes called toggle function). Sure, we could just buy a mechanical switch with the alternate on/off action built in... However, this example is educational and also fun! Next we draw the state transition diagram.

A typical first approach is to use X0 for both transitions (like the example shown to the right). However, *this is incorrect* (please keep reading).

Note that this example differs from the motor example, because now we have just one pushbutton. When we press the pushbutton, both transition conditions are met. We would just transition around the state diagram at top speed. If implemented in Stage, this solution would flash the light on or off each scan (obviously undesirable)!

The solution is to make the push and the release of the pushbutton separate events. Refer to the new state transition diagram below. At powerup we enter the OFF state. When switch X0 is pressed, we enter the Press-ON state. When it is released, we enter the ON state. Note that X0

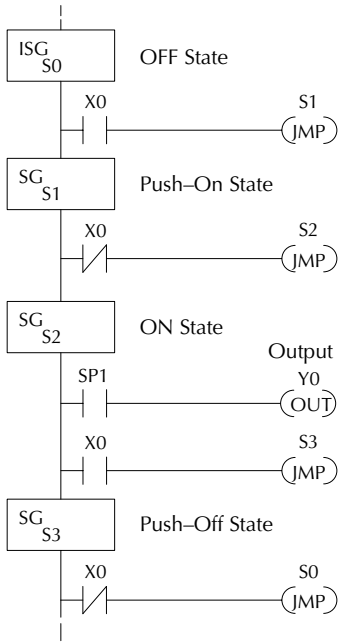


with the bar above it denotes X0 NOT.

When in the ON state, another push and release cycle similarly takes us back to the OFF state. Now we have two unique states (OFF and ON) used when the pushbutton is released, which is what was required to solve the control problem.

The equivalent stage program is shown to the right. The desired powerup state is OFF, so we make S0 an initial stage (ISG). In the ON state, we add special relay contact SP1, which is always on.

Note that even as our programs grow more complex, it is still easy to correlate the state transition diagram with the stage program!



Four Steps to Writing a Stage Program

By now, you've probably noticed that we follow the same steps to solve each example problem. The steps will probably come to you automatically if you work through all the examples in this chapter. It's helpful to have a checklist to guide us through the problem solving. The following steps summarize the stage program design procedure:

1. Write a Word Description of the application.

Describe all functions of the process in your own words. Start by listing what happens first, then next, etc. If you find there are too many things happening at once, try dividing the problem into more than one process. Remember, you can still have the processes communicate with each other to coordinate their overall activity.

2. Draw the Block Diagram.

Inputs represent all the information the process needs for decisions, and outputs connect to all devices controlled by the process.

- Make lists of inputs and outputs for the process.
- Assign I/O point numbers (X and Y) to physical inputs and outputs.

3. Draw the State Transition Diagram.

The state transition diagram describes the central function of the block diagram, reading inputs and generating outputs.

- Identify and name the states of the process.
- Identify the event(s) required for each transition between states.
- Ensure the process has a way to re-start itself, or is cyclical.
- Choose the powerup state for your process.
- Write the output equations.

4. Write the Stage Program.

Translate the state transition diagram into a stage program.

- Make each state a stage. Remember to number stages in octal. Up to 1024 total stages are available in the DL06, numbered 0 to 1777 in octal.
- Put transition logic inside the stage which originates each transition (the stage each arrow points away from).
- Use an initial stage (ISG) for any states that must be active at powerup.
- Place the outputs or actions in the appropriate stages.

You'll notice that Steps 1 through 3 just prepare us to write the stage program in Step 4. However, the program virtually writes itself because of the preparation beforehand. Soon you'll be able to start with a word description of an application and create a stage program in one easy session!

Stage Program Example: A Garage Door Opener

Garage Door Opener Example

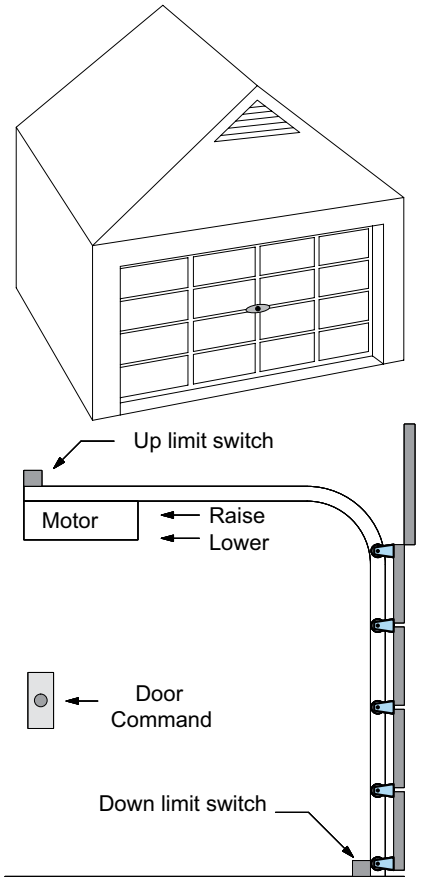
In this next stage programming example, we'll create a garage door opener controller. Hopefully, most readers are familiar with this application, and we can have fun, too!

The first step we must take is to describe how the door opener works. We will start by achieving the basic operation, waiting to add extra features later. Stage programs are very easy to modify.

Our garage door controller has a motor which raises or lowers the door on command. The garage owner pushes and releases a momentary pushbutton once to raise the door. After the door is up, another push-release cycle will lower the door.

In order to identify the inputs and outputs of the system, it's sometimes helpful to sketch its main components, as shown in the door side view to the right. The door has an up limit and a down limit switch. Each limit switch closes only when the door has reach the end of travel in the corresponding direction. In the middle of travel, neither limit switch is closed.

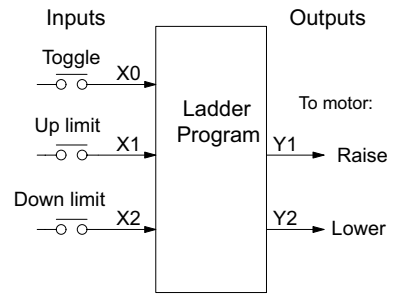
The motor has two command inputs: raise and lower. When neither input is active, the motor is stopped. The door command is just a simple pushbutton. Whether wall-mounted as shown, or a radio-remote control, all door control commands logical OR together as one pair of switch contacts.



Draw the Block Diagram

The block diagram of the controller is shown to the right. Input X0 is from the pushbutton door control. Input X1 energizes when the door reaches the full up position. Input X2 energizes when the door reaches the full down position. When the door is positioned between fully up or down, both limit switches are open.

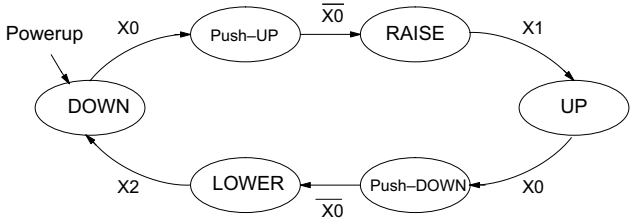
The controller has two outputs to drive the motor. Y1 is the up (raise the door) command, and Y2 is the down (lower the door) command.



Draw the State Diagram

Now we are ready to draw the state transition diagram. Like the previous light bulb controller example, this application also has just one switch for the command input. Refer to the figure below.

- When the door is down (DOWN state), nothing happens until X0 energizes. Its push and release brings us to the RAISE state, where output Y1 turns on and causes the motor to raise the door.
- We transition to the UP state when the up limit switch (X1) energizes, and turns off the motor.
- Then nothing happens until another X0 press-release cycle occurs. That takes us to the LOWER state, turning on output Y2 to command the motor to lower the door. We transition back to the DOWN state when the down limit switch (X2) energizes.

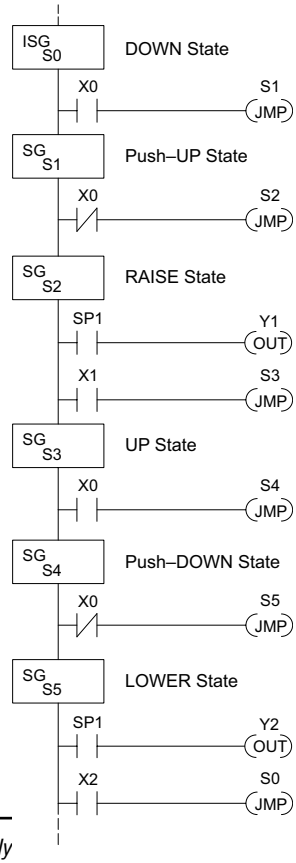


Output equations: Y1 = Raise Y2 = Lower

The equivalent stage program is shown to the right. For now, we will assume the door is down at powerup, so the desired powerup state is DOWN. We make S0 an initial stage (ISG). Stage S0 remains active until the door control pushbutton activates. Then we transition (JMP) to Push-UP stage, S1.

A push-release cycle of the pushbutton takes us through stage S1 to the RAISE stage, S2. We use the always-on contact SP1 to energize the motor's raise command, Y1. When the door reaches the fully-raised position, the up limit switch X1 activates. This takes us to the UP Stage S3, where we wait until another door control command occurs.

In the UP Stage S3, a push-release cycle of the pushbutton will take us to the LOWER Stage S5, where we activate Y2 to command the motor to lower the door. This continues until the door reaches the down limit switch, X2. When X2 closes, we transition from Stage S5 to the DOWN stage S0, where we began.



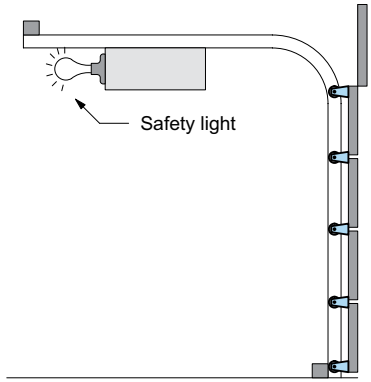
***NOTE**: The only special thing about an initial stage (ISG) is that it is automatically active at powerup. Afterwards, it is just like any other.*

Add Safety Light Feature

Next we will add a safety light feature to the door opener system. It's best to get the main function working first as we have done, then adding the secondary features.

The safety light is standard on many commercially-available garage door openers. It is shown to the right, mounted on the motor housing. The light turns on upon any door activity, remaining on for approximately 3 minutes afterwards.

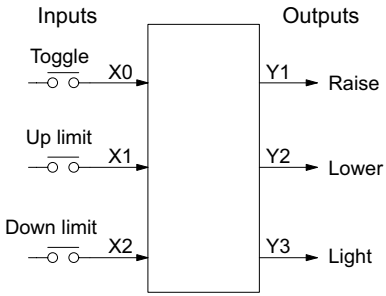
This part of the exercise will demonstrate the use of parallel states in our state diagram. Instead of using the JMP instruction, we'll use the set and reset commands.



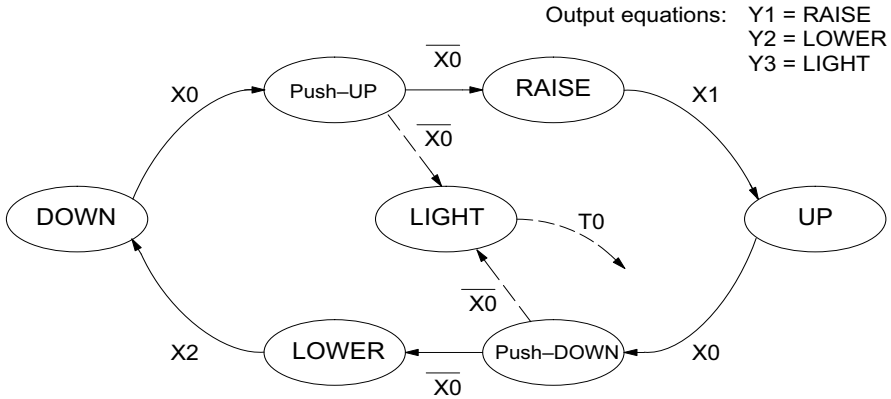
Modify the Block Diagram and State Diagram

To control the light bulb, we add an output to our controller block diagram, shown to the right, Y3 is the light control output.

In the diagram below, we add an additional state called "LIGHT". Whenever the garage owner presses the door control switch and releases, the RAISE or LOWER state is active *and the LIGHT state is simultaneously active*. The line to the Light state is dashed, because it is not the primary path.



We can think of the Light state as a parallel process to the raise and lower state. The paths to the Light state are not a transition (Stage JMP), but a State Set command. In the logic of the Light stage, we will place a three-minute timer. When it expires, timer bit T0 turns on and resets the Light stage. The path out of the Light stage goes nowhere, indicating the Light stage just becomes inactive, and the light goes out!



Using a Timer Inside a Stage

The finished modified program is shown to the right. The shaded areas indicate the program additions.

In the Push-UP stage S1, we add the Set Stage Bit S6 instruction. When contact X0 closes, we transition from S1 and go to two new active states: S2 and S6. In the Push-DOWN state S4, we make the same additions. So, any time someone presses the door control pushbutton, the light turns on.

Most new stage programmers would be concerned about where to place the Light Stage in the ladder, and how to number it. The good news is that it doesn't matter!

- Just choose an unused Stage number, and use it for the new stage and as the reference from other stages.
- Placement in the program is not critical, so we place it at the end.

You might think that each stage has to be directly under the stage that transitions to it. While it is good practice, it is not required (that's good, because our two locations for the Set S6 instruction make that impossible). Stage numbers and how they are used determines the transition paths.

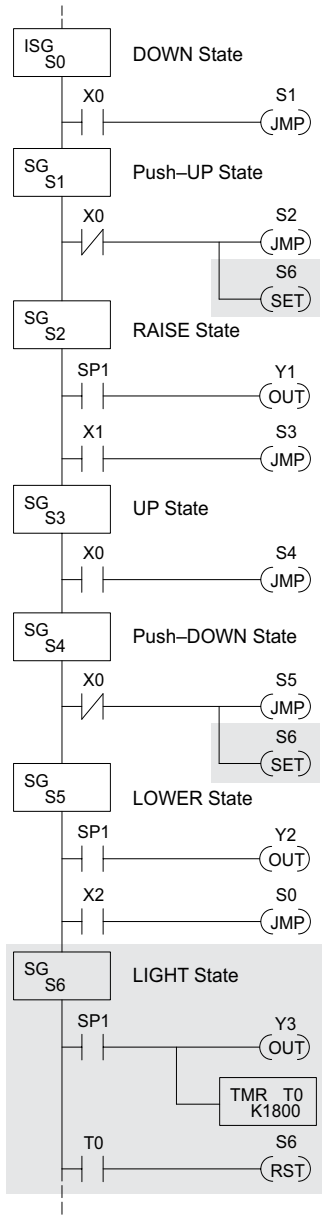
In stage S6, we turn on the safety light by energizing Y3. Special relay contact SP1 is always on. Timer T0 times at 0.1 second per count. To achieve 3 minutes time period, we calculate:

$$K = \frac{3 \text{ min.} \times 60 \text{ sec/min}}{0.1 \text{ sec/count}}$$

$$K = 1800 \text{ counts}$$

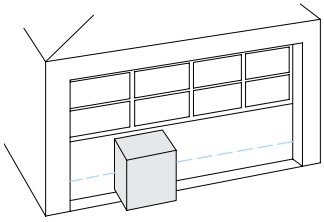
The timer has power flow whenever stage S6 is active. The corresponding timer bit T0 is set when the timer expires. So three minutes later, T0=1 and the instruction Reset S6 causes the stage to be inactive.

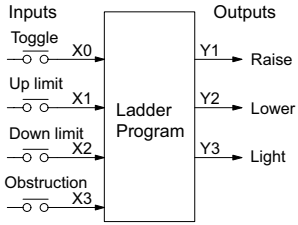
While Stage S6 is active and the light is on, stage transitions in the primary path continue normally and independently of Stage 6. That is, the door can go up, down, or whatever, but the light will be on for precisely 3 minutes.



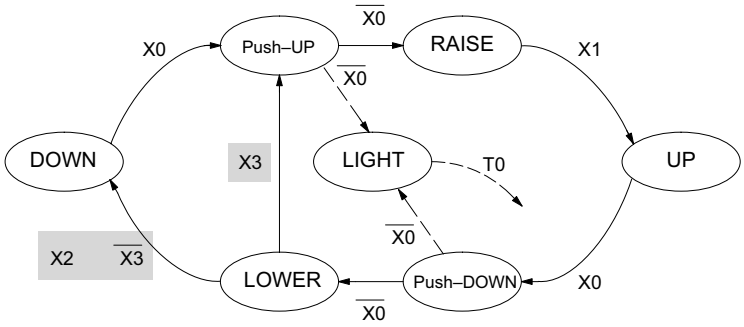
Add Emergency Stop Feature

Some garage door openers today will detect an object under the door. This halts further lowering of the door. Usually implemented with a photocell (electric-eye), a door in the process of being lowered will halt and begin raising. We will define our safety feature to work in this way, adding the input from the photocell to the block diagram as shown to the right. X3 will be on if an object is in the path of the door.





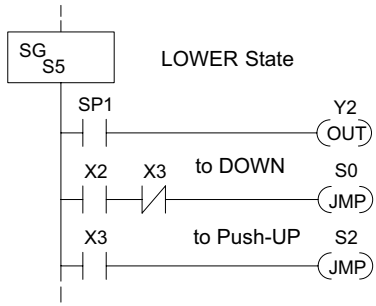
Next, we make a simple addition to the state transition diagram, shown in shaded areas in the figure below. Note the new transition path at the top of the LOWER state. If we are lowering the door and detect an obstruction (X3), we then jump to the Push-UP State. We do this instead of jumping directly to the RAISE state, to give the Lower output Y2 one scan to turn off, before the Raise output Y1 energizes.



Exclusive Transitions

It is theoretically possible that the down limit (X2) and the obstruction input (X3) could energize at the same moment. In that case, we would **jump** to the Push-UP and DOWN states simultaneously, which does not make sense.

Instead, we give priority to the obstruction by changing the transition condition to the DOWN state to [X2 and NOT X3]. This ensures the obstruction event has the priority. The modifications we must make to the LOWER Stage (S5) logic are shown to the right. The first rung remains unchanged. The second and third rungs implement the transitions we need. Note the opposite relay contact usage for X3, which ensures the stage will execute only one of the JMP instructions.

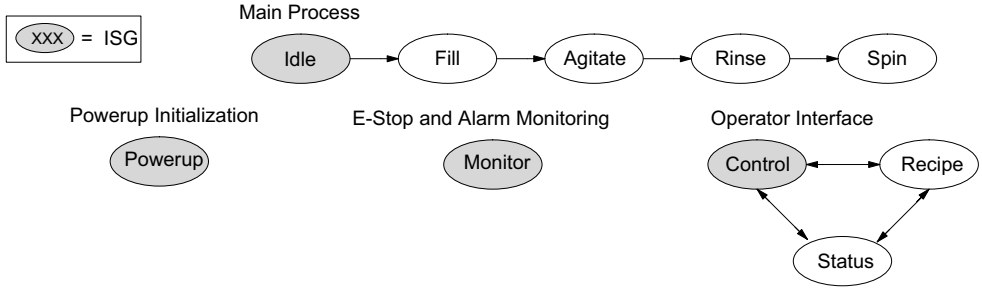


Stage Program Design Considerations

Stage Program Organization

The examples so far in this chapter used one self-contained state diagram to represent the main process. However, we can have multiple processes implemented in stages, all in the same ladder program. New stage programmers sometimes try to turn a stage on and off each scan, based on the false assumption that only one stage can be on at a time. For ladder rungs that you want to execute each scan, just put them in a stage that is always on.

The following figure shows a typical application. During operation, the primary manufacturing activity Main Process, Powerup Initialization, E-Stop and Alarm Monitoring, and Operator Interface are all running. At powerup, three initial stages shown begin operation.



In a typical application, the separate stage sequences above operate as follows:

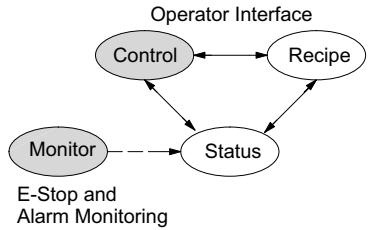
Powerup Initialization – This stage contains ladder rung tasks done just once at powerup. Its last rung resets the stage, so this stage is only active for one scan (or only as many scans that are required).

Main Process –This stage sequence controls the heart of the process or machine. One pass through the sequence represents one part cycle of the machine, or one batch in the process.

E-Stop and Alarm Monitoring –This stage is always active because it is watching for errors that could indicate an alarm condition or require an emergency stop. It is common for this stage to reset stages in the main process or elsewhere, in order to initialize them after an error condition.

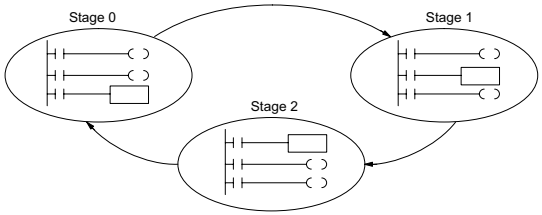
Operator Interface –This is another task that must always be active and ready to respond to an operator. It allows an operator interface to change modes, etc., independently of the current main process step.

Although we have separate processes, there can be coordination among them. For example, in an error condition, the Status Stage may want to automatically switch the operator interface to the status mode to show error information as shown to the right. The monitor stage could set the stage bit for Status and Reset the stages Control and Recipe.



How Instructions Work Inside Stages

We can think of states or stages as simply dividing up our ladder program as depicted in the figure below. Each stage contains only the ladder rungs which are needed for the corresponding state of the process. The logic for transitioning out of a stage is contained within that stage. It's easy to choose which ladder rungs are active at powerup by using an **initial** stage type (ISG).



Most all instructions work just like they do in standard RLL. You can think of a stage just like a miniature RLL program which is either active or inactive.

Output Coils – As expected, output coils in active stages will turn on or off outputs according to power flow into the coil. However, note the following:

- Outputs work as usual, provided each output reference, such as "Y3", is used in only one stage.
- An output can be referenced from more than one stage, as long as only one of the stages is active at a time.
- If an output coil is controlled by more than one stage simultaneously, the active stage nearest the bottom of the program determines the final output status during each scan. Therefore, use the OROUT instruction instead when you want multiple stages to have a logical OR control of an output.

One-Shot or PD coils – Use care if you must use a Positive Differential coil in a stage. Remember that the input to the coil must make a 0–1 transition. If the coil is already energized on the first scan when the stage becomes active, the PD coil will not work. This is because the 0–1 transition did not occur.

PD coil alternative: If there is a task which you want to do only once (on 1 scan), it can be placed in a stage which transitions to the next stage on the same scan.

Counter – In using a counter inside a stage, the stage must be active for one scan before the input to the counter makes a 0–1 transition. Otherwise, there is no real transition and the counter will not count.

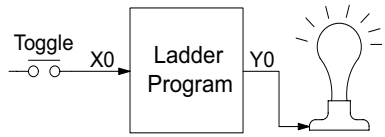
The ordinary Counter instruction does have a restriction inside stages: it may not be reset from other stages using the RST instruction for the counter bit. However, the special Stage counter provides a solution (see next paragraph).

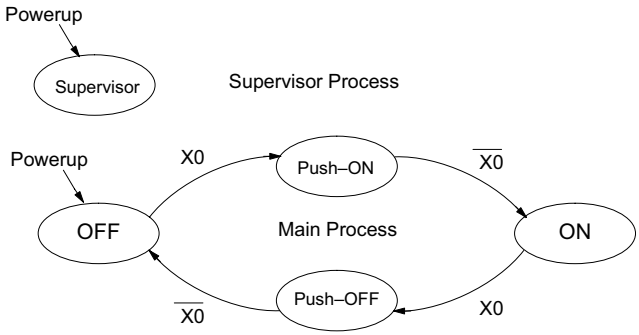
Stage Counter – The Stage Counter has the benefit that its count may be globally reset from other stages by using the RST instruction. It has a count input, but no reset input. This is the only difference from a standard counter.

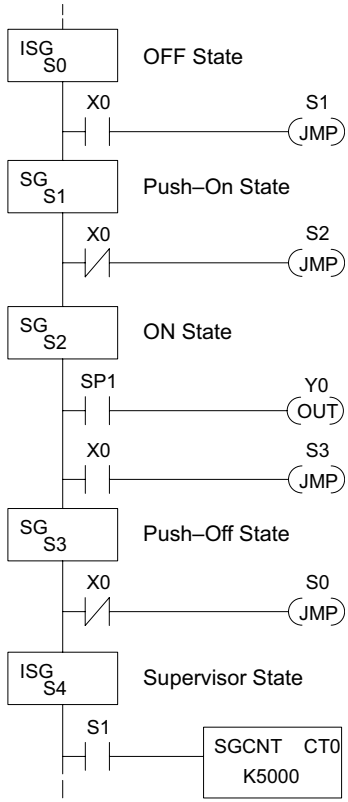
Drum – Realize that the drum sequencer is its own process, and is a different programming method than stage programming. If you need to use a drum with stages, be sure to place the drum instruction in an ISG stage that is always active.

Using a Stage as a Supervisory Process

You may recall the light bulb on-off controller example from earlier in this chapter. For the purpose of illustration, suppose we want to monitor the *productivity* of the lamp process, by counting the number of on-off cycles which occurs. This application will require the addition of a simple counter, but the key decision is in where to put the counter.







New stage programming students will typically try to place the counter inside one of the stages of the process they are trying to monitor. The problem with this approach is that the stage is active only part of the time. In order for the counter to count, the count input must transition from off to on at least one scan after its stage activates. Ensuring this requires extra logic that can be tricky.

In this case, we only need to add another supervisory stage as shown above, to *watch* the main process. The counter inside the supervisor stage uses the stage bit S1 of the main process as its count input. *Stage bits used as a contact let us monitor a process!*

***NOTE**: Both the **Supervisor** stage and the **OFF** stage are initial stages. The supervisor stage remains active indefinitely.*

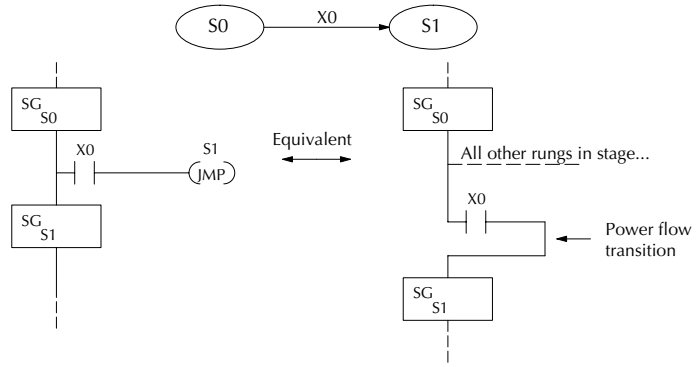
Stage Counter

The counter in the above example is a special Stage Counter. Note that it does not have a reset input. The count is reset by executing a Reset instruction, naming the counter bit (CT0 in this case). The Stage Counter has the benefit that its count may be globally reset from other stages. The standard Counter instruction does not have this global reset capability. You may still use a regular Counter instruction inside a stage... however, the reset input to the counter is the only way to reset it.

Power Flow Transition Technique

Our discussion of state transitions has shown how the Stage JMP instruction makes the current stage inactive and the next stage (named in the JMP) active. As an alternative way to enter this in *Direct*SOFT, you may use the power flow method for stage transitions.

The main requirement is that the current stage be located directly above the next (jump-to) stage in the ladder program. This arrangement is shown in the diagram below, by stages S0 and S1, respectively.

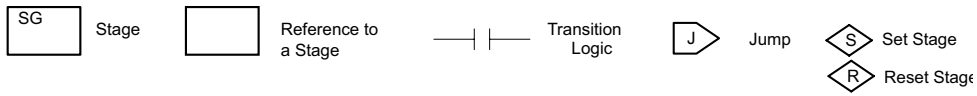


Remember that the Stage JMP instruction may occur anywhere in the current stage, and the result is the same. However, power flow transitions, as shown above, must occur as the last rung in a stage. All other rungs in the stage will precede it. The power flow transition method is also achievable on the handheld programmer, by simply following the transition condition with the Stage instruction for the next stage.

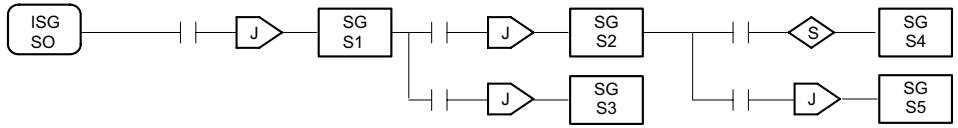
The power flow transition method does eliminate one Stage JMP instruction, its only advantage. However, it is not as easy to make program changes as using the Stage JMP. Therefore, we advise using Stage JMP transitions for most programmers.

Stage View in DirectSOFT

The Stage View option in *Direct*SOFT will let you view the ladder program as a flow chart. The figure below shows the symbol convention used in the diagrams. You may find the stage view useful as a tool to verify that your stage program has faithfully reproduced the logic of the state transition diagram you intend to realize.



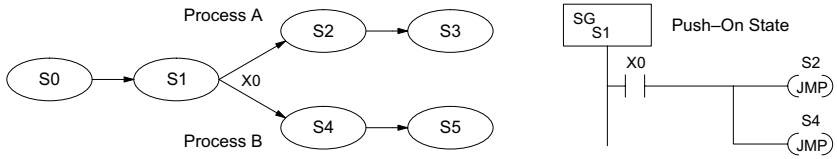
The following diagram is a typical stage view of a ladder program containing stages. Note the left-to-right direction of the flow chart.



Parallel Processing Concepts

Parallel Processes

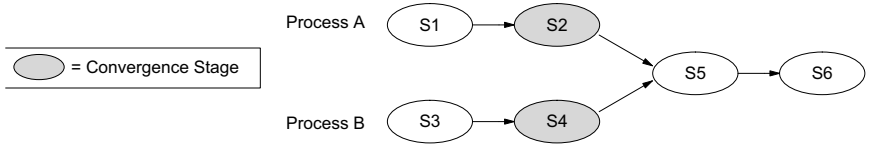
Previously in this chapter we discussed how a state may transition to either one state or another, called an *exclusive transition*. In other cases, we may need to branch simultaneously to two or more parallel processes, as shown below. It is acceptable to use all JMP instructions as shown, or we could use one JMP and a Set Stage bit instruction(s) (at least one must be a JMP, in order to leave S1). Remember that all instructions in a stage execute, even when it transitions (the JMP is not a GOTO).



Note that if we want Stages S2 and S4 to energize exactly on the same scan, both stages must be located below or above Stage S1 in the ladder program (see the explanation at the bottom of page 7–7). Overall, parallel branching is easy!

Converging Processes

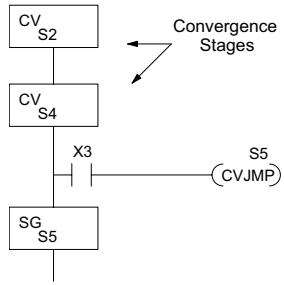
Now, we consider the opposite case of parallel branching, which is converging processes. This simply means we stop doing multiple things and continue doing one thing at a time. In the figure below, processes A and B converge when stages S2 and S4 transition to S5 at some point in time. So, S2 and S4 are *Convergence Stages.*



Convergence Stages (CV)

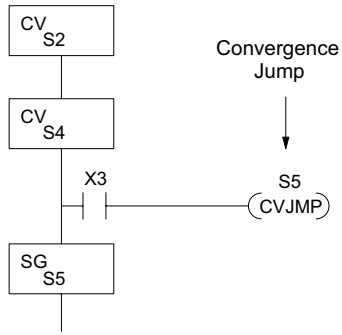
While the converging principle is simple enough, it brings a new complication. As parallel processing completes, the multiple processes almost never finish at the same time. In other words, how can we know whether Stage S2 or S4 will finish last? This is an important point, because we have to decide how to transition to Stage S5.

The solution is to coordinate the transition condition out of convergence stages. We accomplish this with a stage type designed for this purpose: the Convergence Stage (type CV). In the example to the right, convergence stages S2 and S4 are required to be grouped together as shown. **No logic is permitted between CV stages!** The transition condition (X3 in this case) must be located in the last convergence stage. The transition condition only has power flow when all convergence stages in the group are active.



Convergence Jump (CVJMP)

Remember, the last convergence stage only has power flow when all CV stages in the group are active. To complement the convergence stage, we need a new jump instruction. The Convergence Jump (CVJMP) shown to the right will transition to Stage S5 when X3 is active (as one might expect), but it also *automatically resets all convergence stages in the group*. This makes the CVJMP jump a very powerful instruction. Note that this instruction may only be used with convergence stages.



Convergence Stage Guidelines

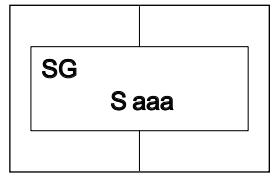
The following summarizes the requirements in the use of convergence stages, including some tips for their effective application:

- A convergence stage is to be used as the last stage of a process which is running in parallel to another process or processes. A transition to the convergence stage means that a particular process is finished and represents a waiting point until all other parallel processes also finish.
- The maximum number of convergence stages which make up one group is 16. In other words, a maximum of 16 stages can converge into one stage.
- Convergence stages of the same group must be placed together in the program, connected on the power rail without any other logic in between.
- Within a convergence group, the stages may occur in any order, top to bottom. It does not matter which stage is last in the group, because all convergence stages have to be active before the last stage has power flow.
- The last convergence stage of a group may have ladder logic within the stage. However, this logic will not execute until all convergence stages of the group are active.
- The convergence jump (CVJMP) is the intended method to be used to transition from the convergence group of stages to the next stage. The CVJMP resets all convergence stages of the group, and energizes the stage named in the jump.
- The CVJMP instruction must only be used in a convergence stage, as it is invalid in regular or initial stages.
- Convergence Stages or CVJMP instructions may not be used in subroutines or interrupt routines.

RLLPLUS (Stage) Instructions

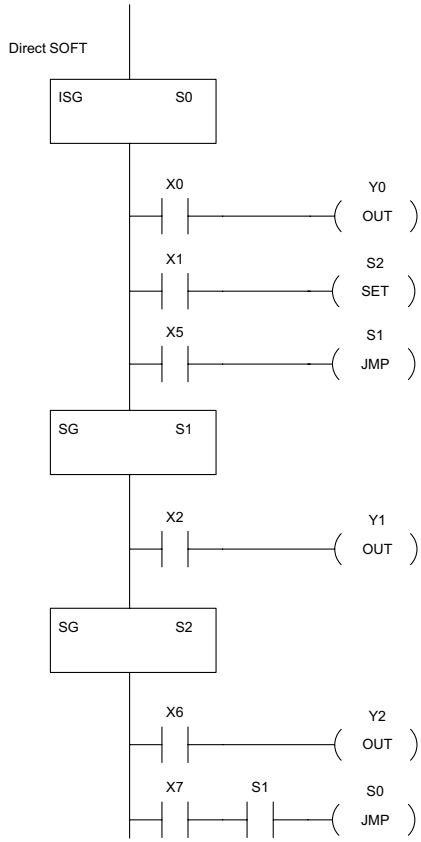
Stage (SG)

The Stage instructions are used to create structured RLL[PLUS] programs. Stages are program segments which can be activated by transitional logic, a jump or a set stage that is executed from an active stage. Stages are deactivated one scan after transitional logic, a jump, or a reset stage instruction is executed.



Operand Data Type	DL06 Range
	aaa
Stage S	0–1777

The following example is a simple RLL[PLUS] program. This program utilizes an initial stage, stages, and jump instructions to create a structured program.

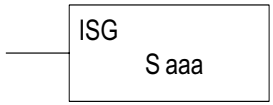


Handheld Programmer Keystrokes

U ISG	→	A 0	ENT			
$ STR	→	A 0	ENT			
GX OUT	→	A 0	ENT			
$ STR	→	B 1	ENT			
X SET	→	SHFT	S RST	C 2	ENT	
$ STR	→	F 5	ENT			
K JMP	→	B 1	ENT			
2 SG	→	B 1	ENT			
$ STR	→	C 2	ENT			
GX OUT	→	B 1	ENT			
2 SG	→	C 2	ENT			
$ STR	→	G 6	ENT			
GX OUT	→	C 2	ENT			
$ STR	→	H 7	ENT			
V AND	→	SHFT	S RST	B 1	ENT	
K JMP	→	A 0	ENT			

Initial Stage (ISG)

The Initial Stage instruction is normally used as the first segment of an RLL^{PLUS} program. Multiple Initial Stages are allowed in a program. They will be active when the CPU enters the Run mode allowing for a starting point in the program. Initial Stages are also activated by transitional logic, a jump or a set stage executed from an active stage.

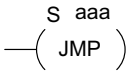


Operand Data Type		DL06 Range
		aaa
Stage	S	0–1777

NOTE: *If the ISG is within the retentive range for stages, the ISG will remain in the state it was in before power down and will NOT turn itself on during the first scan.*

Jump (JMP)

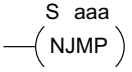
The Jump instruction allows the program to transition from an active stage containing the jump instruction to another stage (specified in the instruction). The jump occurs when the input logic is true. The active stage containing the Jump will deactivate 1 scan later.



Operand Data Type		DL06 Range
		aaa
Stage	S	0–1777

Not Jump (NJMP)

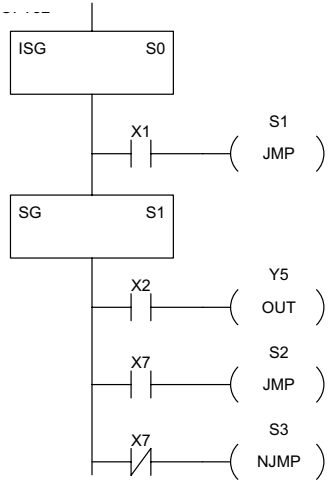
The Not Jump instruction allows the program to transition from an active stage which contains the jump instruction to another which is specified in the instruction. The jump will occur when the input logic is off. The active stage that contains the Not Jump will be deactivated 1 scan after the Not Jump instruction is executed.



Operand Data Type		DL06 Range
		aaa
Stage	S	0–1777

In the following example, only stage ISG0 will be active when program execution begins. When X1 is on, program execution will jump from Initial Stage 0 to Stage 1.

DirectSOFT



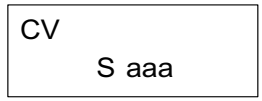
Handheld Programmer Keystrokes

U ISG	→	A 0	ENT				
$ STR	→	B 1	ENT				
K JMP	→	B 1	ENT				
2 SG	→	B 1	ENT				
$ STR	→	C 2	ENT				
GX OUT	→	F 5	ENT				
$ STR	→	H 7	ENT				
K JMP	→	C 2	ENT				
SHFT	N TMR	SHFT	K JMP	→	D 3	ENT	

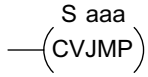
Converge Stage (CV) and Converge Jump (CVJMP)

The Converge Stage instruction is used to group certain stages together by defining them as Converge Stages.

When all of the Converge Stages within a group become active, the CVJMP instruction (and any additional logic in the final CV stage) will be executed. All preceding CV stages must be active before the final CV stage logic can be executed. All Converge Stages are deactivated one scan after the CVJMP instruction is executed.



Additional logic instructions are only allowed following the last Converge Stage instruction and before the CVJMP instruction. Multiple CVJMP instructions are allowed.

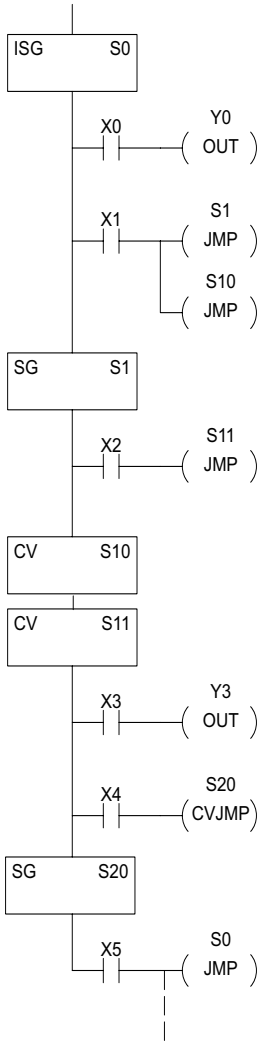


Converge Stages must be programmed in the main body of the application program. This means they cannot be programmed in Subroutines or Interrupt Routines.

Operand Data Type		DL06 Range
		aaa
Stage	S	0–1777

In the following example, when Converge Stages S10 and S11 are both active the CVJMP instruction will be executed when X4 is on. The CVJMP will deactivate S10 and S11, and activate S20. Then, if X5 is on, the program execution will jump back to the initial stage, S0.

DirectSOFT



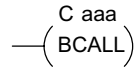
Handheld Programmer Keystrokes

U ISG	→	A 0	ENT							
$ STR	→	A 0	ENT							
GX OUT	→	A 0	ENT							
$ STR	→	B 1	ENT							
K JMP	→	B 1	ENT							
K JMP	→	B 1	A 0	ENT						
2 SG	→	B 1	ENT							
$ STR	→	C 2	ENT							
K JMP	→	B 1	B 1	ENT						
SHFT	C 2	V AND	→	B 1	A 0	ENT				
SHFT	C 2	V AND	→	B 1	B 1	ENT				
$ STR	→	D 3	ENT							
GX OUT	→	D 3	ENT							
$ STR	→	E 4	ENT							
SHFT	C 2	V AND	SHFT	K JMP	→	C 2	A 0	ENT		
2 SG	→	C 2	A 0	ENT						
$ STR	→	F 5	ENT							
K JMP	→	A 0	ENT							

Block Call (BCALL)

The stage block instructions are used to activate a block of stages. The Block Call, Block, and Block End instructions must be used together. The BCALL instruction is used to activate a stage block. There are several things you need to know about the BCALL instruction.

Uses CR Numbers — The BCALL appears as an output coil, but does not actually refer to a Stage number as you might think. Instead, the block is identified with a Control Relay (Caaa). This control relay cannot be used as an output anywhere else in the program.



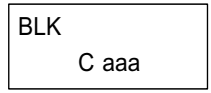
Must Remain Active — The BCALL instruction actually controls all the stages between the BLK and the BEND instructions even after the stages inside the block have started executing. The BCALL must remain active or all the stages in the block will automatically be turned off. If either the BCALL instruction, or the stage that contains the BCALL instruction goes off, then the stages in the defined block will be turned off automatically.

Activates First Block Stage — When the BCALL is executed it automatically activates the first stage following the BLK instructions.

Operand Data Type		DL06 Range
		aaa
Control Relay	S	0–1777

Block (BLK)

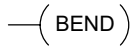
The Block instruction is a label which marks the beginning of a block of stages that can be activated as a group. A Stage instruction must immediately follow the Start Block instruction. Initial Stage instructions are not allowed in a block. The control relay (Caaa) specified in Block instruction must not be used as an output any where else in the program.



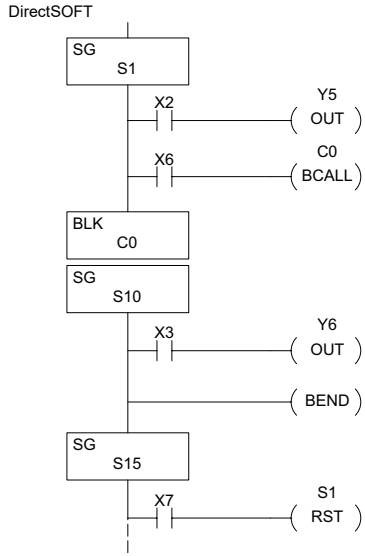
Operand Data Type		DL06 Range
		aaa
Control Relay	S	0–1777

Block End (BEND)

The Block End instruction is a label used with the Block instruction. It marks the end of a block of stages. There is no operand with this instruction. Only one Block End is allowed per Block Call.



In this example, the Block Call is executed when stage 1 is active and X6 is on. The Block Call then automatically activates stage S10, which immediately follows the Block instruction. This allows the stages between S10 and the Block End instruction to operate as programmed. If the BCALL instruction is turned off, or if the stage containing the BCALL instruction is turned off, then all stages between the BLK and BEND instructions are automatically turned off. If you examine S15, you will notice that X7 could reset Stage S1, which would disable the BCALL, thus resetting all stages within the block.

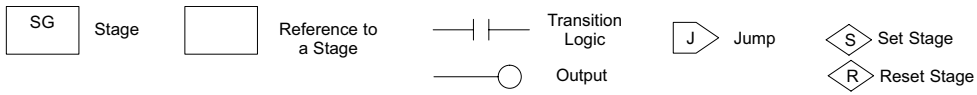


Handheld Programmer Keystrokes

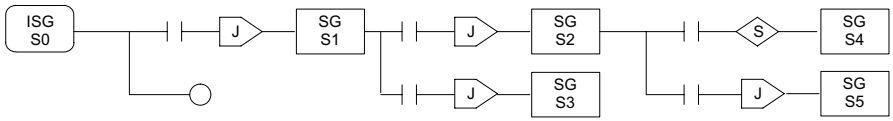
SG	→	S(SG)	1	ENT					
STR	→	X(IN)	2	ENT					
OUT	→	Y(OUT)	5	ENT					
STR	→	X(IN)	6	ENT					
SHFT	B	C	A	L	L	→	C(CR)	0	ENT
SHFT	B	L	K	→	C(CR)	0	ENT		
SG	→	S(SG)	1	0	ENT				
STR	→	X(IN)	3	ENT					
OUT	→	Y(OUT)	6	ENT					
SHFT	B	E	N	D	ENT				
SG	→	S(SG)	1	5	ENT				
STR	→	X(IN)	7	ENT					
RST	→	S(SG)	1	ENT					

Stage View in *Direct*SOFT

The Stage View option in *Direct*SOFT will let you view the ladder program as a flow chart. The figure below shows the symbol convention used in the diagrams. You may find the stage view useful as a tool to verify that your stage program has faithfully reproduced the logic of the state transition diagram you intend to realize.



The following diagram is a typical stage view of a ladder program containing stages. Note the left-to-right direction of the flow chart.



Questions and Answers about Stage Programming

We include the following commonly-asked questions about Stage Programming as an aid to new students. All question topics are covered in more detail in this chapter.

Q. What does stage programming do that I can't do with regular RLL programs?

A. Stages allow you to identify all the states of your process before you begin programming. This approach is more organized, because you divide up a ladder program into sections. As stages, these program sections are active only when they are actually needed by the process. Most processes can be organized into a sequence of stages, connected by event-based transitions.

Q. What are Stage Bits?

A. A stage bit is just a single bit in the CPU's image register, representing the active/inactive status of the stage in real time. For example, the bit for Stage 0 is referenced as "S0". If S0 = 0, then the ladder rungs in Stage 0 are bypassed (not executed) on each CPU scan. If S0 = 1, then the ladder rungs in Stage 0 are executed on each CPU scan. Stage bits, when used as contacts, allow one part of your program to monitor another part by detecting stage active/inactive status.

Q. How does a stage become active?

A. There are three ways:

- If the Stage is an initial stage (ISG), it is automatically active at powerup.
- Another stage can execute a Stage JMP instruction naming this stage, which makes it active upon its next occurrence in the program.
- A program rung can execute a Set Stage Bit instruction (such as Set S0).

Q. How does a stage become inactive?

A. There are three ways:

- Standard Stages (SG) are automatically inactive at powerup.
- A stage can execute a Stage JMP instruction, resetting its Stage Bit to 0.
- Any rung in the program can execute a Reset Stage Bit instruction (such as Reset S0).

Q. What about the power flow technique of stage transitions?

A. The power flow method of connecting adjacent stages (directly above or below in the program) actually is the same as the Stage Jump instruction executed in the stage above, naming the stage below. Power flow transitions are more difficult to edit in ***Direct***SOFT, we list them separately from two preceding questions.

Q. Can I have a stage which is active for only one scan?

A. Yes, but this is not the intended use for a stage. Instead, just make a ladder rung active for 1 scan by including a stage Jump instruction at the bottom of the rung. Then the ladder will execute on the last scan before its stage jumps to a new one.

Q. Isn't a Stage JMP just like a regular GOTO instruction used in software?

A. No, it is very different. A GOTO instruction sends the program execution immediately to the code location named by the GOTO. A Stage JMP simply resets the Stage Bit of the current stage, while setting the Stage Bit of the stage named in the JMP instruction. Stage bits are 0 or 1, determining the inactive/active status of the corresponding stages. A stage JMP has the following results:

- When the JMP is executed, the remainder of the current stage's rungs are executed, even if they reside past(under) the JMP instruction. On the following scan, that stage is not executed, because it is inactive.
- The Stage named in the Stage JMP instruction will be executed upon its next occurrence. If located past (under) the current stage, it will be executed on the same scan. If located before (above) the current stage, it will be executed on the following scan.

Q. How can I know when to use stage JMP, versus a Set Stage Bit or Reset Stage Bit?

A. These instructions are used according to the state diagram topology you have derived:

- Use a Stage JMP instruction for a state transition ... moving from one state to another.
- Use a Set Stage Bit instruction when the current state is spawning a new parallel state or stage sequence, or when a supervisory state is starting a state sequence under its command.
- Use a Reset Bit instruction when the current state is the last state in a sequence and its task is complete, or when a supervisory state is ending a state sequence under its command.

Q. What is an initial stage, and when do I use it?

A. An initial stage (ISG) is automatically active at powerup. Afterwards, it works just like any other stage. You can have multiple initial stages, if required. Use an initial stage for ladder that must always be active, or as a starting point.

Q. Can I place program ladder rungs outside of the stages, so they are always on?

A. It is possible, but it's not good software design practice. Place ladder that must always be active in an initial stage, and do not reset that stage or use a Stage JMP instruction inside it. It can start other stage sequences at the proper time by setting the appropriate Stage Bit(s).

Q. Can I have more than one active stage at a time?

A. Yes, and this is a normal occurrence for many programs. However, it is important to organize your application into separate processes, each made up of stages. And a good process design will be mostly sequential, with only one stage on at a time. However, all the processes in the program may be active simultaneously.

PID Loop Operation

In This Chapter...

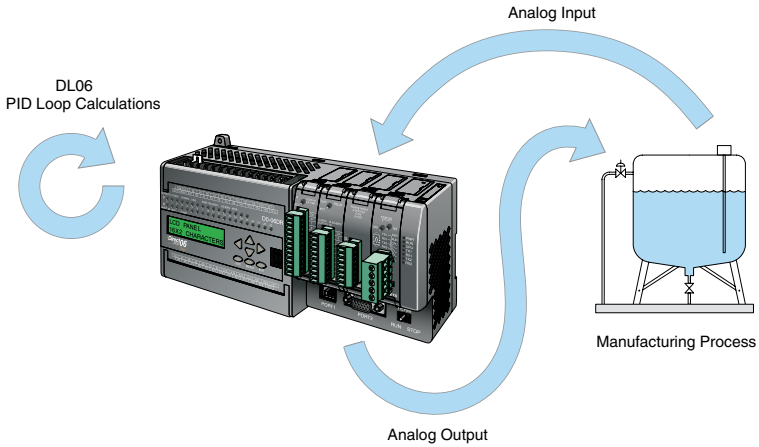
DL06 PID Control

DL06 PID Control Features

Along with control functions discussed in this manual, the DL06 PLC features PID process control capability. The DL06 PID process control loops offer the same features offered in much larger PLCs. The primary features are:

- Up to 8 PID loops, individual programmable sample rates
- Manual, Automatic and Cascade loop operation modes
- Two types of bumpless transfer available
- Full-featured alarms
- Ramp/soak generator with up to 16 segments
- Auto Tuning

The DL06 CPU has process control loop capability in addition to ladder program execution. You can select and configure up to eight loops. All sensor and actuator wiring connects directly to DL06 analog modules. All process variables, gain values, alarm levels, etc., associated with each loop reside in a Loop Variable Table in the CPU. The DL06 CPU reads process variable (PV) inputs during each scan. Then, it makes PID loop calculations during a dedicated time slice on each PLC scan, updating the control output value. The control loops use a Proportional-Integral-Derivative (PID) algorithm to generate the control output. This chapter describes how the loops operate, and how to configure and tune the loops.



*Direct*SOFT programming software, release 5, or later, is used for configuring analog control loops in the DL06. *Direct*SOFT uses dialog boxes to help you set up the individual loops. After completing the setup, you can use *Direct*SOFT's PID Trend View to tune each loop. The configuration and tuning selections you make are stored in the DL06's V-memory (RAM). The loop parameters also may be saved to disk for recall later.

PID Loop Feature	Specifications
Number of loops	Selectable, 8 maximum
CPU V-memory needed	32 words (V locations) per loop selected, 64 words if using ramp/soak
PID algorithm	Position or Velocity form of the PID equation
Control Output polarity	Selectable direct-acting or reverse-acting
Error term curves	Selectable as linear, square root of error, and error squared
Loop update rate (time between PID calculation)	0.05 to 99.99 seconds, user programmable
Minimum loop update rate	0.05 seconds for 1 to 4 loops 0.1 seconds for 5 to 8 loops
Loop modes	Automatic, Manual (operator control), or Cascade control
Ramp/Soak Generator	Up to 8 ramp/soak steps (16 segments) per loop with indication of ramp/soak step number
PV curves	Select standard linear, or square-root extract (for flow meter input)
Set Point Limits	Specify minimum and maximum setpoint values
Process Variable Limits	Specify minimum and maximum Process Variable values
Proportional Gain	Specify gains of 0.01 to 99.99
Integrator (Reset)	Specify reset time of 0.1 to 99.99 in units of seconds or minutes
Derivative (Rate)	Specify the derivative time from 0.01 to 99.99 seconds
Rate Limits	Specify derivative gain limiting from 1 to 20
Bumpless Transfer I	Automatically sets the bias equal to the control output and the setpoint equal to the process variable when control switches from manual to automatic.
Bumpless Transfer II	Automatically sets the bias equal to the control output when control switches from manual to automatic.
Step Bias	Provides proportional bias adjustment for large setpoint changes
Anti-windup (Freeze Bias)	For position form of PID, this inhibits integrator action when the control output reaches 0% or 100 % (speeds up loop recovery when output recovers from saturation)
Error Deadband	Specify a tolerance (plus and minus) for the error term (SP–PV), so that no change in control output value is made

Alarm Feature	Specifications
PV Alarm Hysteresis	Specify 1 to 200 (word/binary) does not affect all alarms, such as PV Rate-of-Change Alarm
PV Alarm Points	Select PV alarm settings for Low–low, Low, High, and High-high conditions
PV Deviation	Specify alarms for two ranges of PV deviation from the setpoint value
Rate of Change	Detect when PV exceeds a rate of change limit you specify

Introduction to PID Control

What is PID Control?

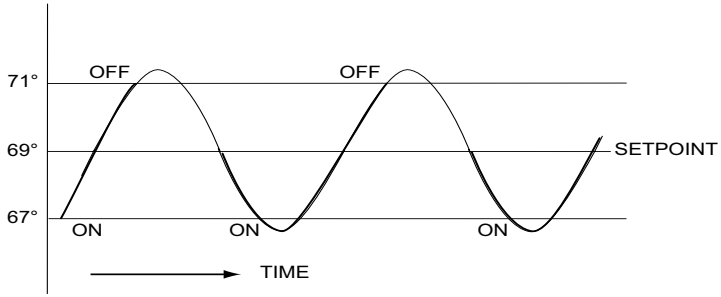
In this discussion, we will explain why PID control is used in process control instead of trying to provide control by simply using an analog input and a discrete output.

There are many types of analog controllers available, and the proper selection will depend upon the particular application. There are two types of analog controllers that are used throughout industry:

1. The ON-OFF controller, sometimes referred to as an open loop controller.
2. The PID controller, sometimes called a closed loop controller.

Regardless of type, analog controllers require input signals from electronic sensors such as pressure, differential pressure, level, flow meter or thermocouples. As an example, one of the most common analog control applications is located in your house for controlling either heat or air conditioning, the thermostat.

You wish for your house to be at a comfortable temperature so you set a thermostat to a desired temperature (setpoint). You then select the comfort mode, either heat or A/C. A temperature sensing device, normally a thermistor, is located within the thermostat. If the thermostat is set for heat and the setpoint is set for 69°, the furnace will be turned on to provide heat at, normally, 2° below the setpoint. In this case, it would turn on at 67°. When the temperature reaches 71°, 2° above setpoint, the furnace will turn off. In the opposite example, if the thermostat is set for A/C (cooling), the thermostat will turn the A/C unit on/off opposite the heat setting. For instance, if the thermostat is set to cool at 76°, the A/C unit will turn on when the sensed temperature reaches 2° above the setpoint or 78°, and turn off when the temperature reaches 74°. This would be considered to be an ON-OFF controller. The waveform below shows the action of the heating cycle. Note that there is a slight overshoot at the turn-off point, also a slight undershoot at the turn-on point.



The ON-OFF controller is used in some industrial control applications, but is not practical in the majority of industrial control processes.

The most common process controller that is used in industry is the PID controller.

The PID controller controls a continuous feedback loop that keeps the process output (control variable) flowing normally by taking corrective action whenever there is a deviation from the desired value (setpoint) of the process variable (PV) such as, rate of flow, temperature, voltage, etc. An *error* occurs when an operator manually changes the setpoint or when an event (valve opened, closed, etc.) or a disturbance (cold water, wind, etc.) changes the load, thus causing a change in the process variable.

The PID controller receives signals from sensors and computes corrective action to the actuator from a computation based on the error (Proportional), the sum of all previous errors (Integral) and the rate of change of the error (Derivative).

We can look at the PID controller in more simple terms. Take the cruise control on an automobile as an example. Let's say that we are cruising on an interstate highway in a car equipped with cruise control. The driver decides to engage the cruise control by turning it ON, then he manually brings the car to the desired cruising speed, say 70 miles per hour. Once the cruise speed is reached, the SET button is pushed fixing the speed at 70mph, the setpoint. Now, the car is cruising at a steady 70 mph until it comes to a hill to go up. As the car goes up the hill, it tends to slow down. The speed sensor senses this and causes the throttle to increase the fuel to the engine. The vehicle speeds up to maintain 70mph without jerking the car and it reaches the top at the set speed. When the car levels out after reaching the top of the hill it will speed up. The speed sensor senses this and signals the throttle to provide less fuel to the engine, thus, the engine slows down allowing the car to maintain the 70mph speed. How does this application apply to PID control? Lets look at the function of P, I and D terms:

- **Proportional** - is commonly referred to as Proportional Gain. The proportional term is the corrective action which is proportional to the error, that is, the change of the manipulated variable is equal to the proportional gain multiplied by the error (the activating signal). In mathematical terms:

  Proportional action = proportional gain X error
  Error = Setpoint (SP) - Process Variable (PV)

Applying this to the cruise control, the speed was set at 70mph which is the Setpoint. The speed sensor senses the actual speed of the car and sends this signal to the cruise controller as the Process Variable (PV). When the car is on a level highway, the speed is maintained at 70mph, thus, no error since the error would be SP - PV = 0. When the car goes up the hill, the speed sensor detected a slow down of the car, SP-PV = error. The proportional gain would cause the output of the speed controller to bring the car back to the setpoint of 70mph. This would be the Controlled Output.

- **Integral** - this term is often referred to as Reset action. It provides additional compensation to the control output, which causes a change in proportion to the value of the error over a period of time. In other words, the reset term is the integral sum of the error values over a period of time.
- **Derivative** - this term is referred to as rate. The Rate action adds compensation to the control output, which causes a change in proportion to the rate of change of error. Its job is to anticipate the probable growth of the error and generate a contribution to the output in advance.

Introducing DL06 PID Control

The DL06 is capable of controlling a process variable such as those already mentioned. As previously mentioned, the control of a variable, such as temperature, at a given level (setpoint) as long as there are no disturbances (cold water) in the process.

The DL06 PLC has the ability to directly accept signals from electronic sensors, such as thermocouples, pressure, VFDs, etc. These signals may be used in mathematically derived control systems.

In addition, the DL06 has built-in PID control algorithms that can be implemented. The basic function of PID closed loop process control is to maintain certain process characteristics at desired setpoints. As a rule, the process deviates from the desired setpoint reference as a result of load material changes and interaction with other processes. During this control, the actual condition of the process characteristics (liquid level, temperature, motor control, etc.) is measured as a *process variable* (PV) and compared with the target setpoint (SP). When deviations occur, an error is generated by the difference between the process variable (actual value) and the setpoint (desired value). Once an error is detected, the function of the control loop is to modify the control output in order to force the error to zero.

The DL06 PID control provides feedback loops using the PID algorithm. The control output is computed from the measured process variable as follows:

Let:

- K_c = proportional gain
- T_i = Reset or integral time
- T_d = Derivative time or rate
- SP = Setpoint
- PV(t) = Process Variable at time "t"
- e(t) = SP-PV(t) = PV deviation from setpoint at time "t" or PV error.

Then:

- M(t) = Control output at time "t"

$$M(t) = Kc \left[e(t) + 1/T_i \int_0^t e(x)\, dx + T_d\, d/dt\, e(t) \right] + M_o$$

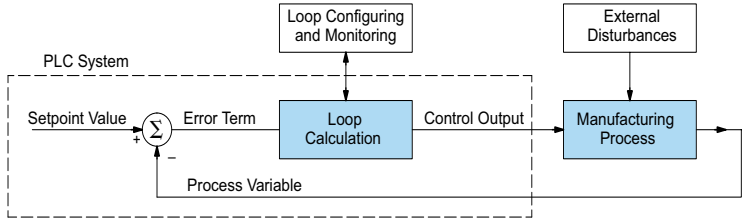
The analog input module receives the process variable in analog form along with an operator entered setpoint; the CPU computes the error. The error is used in the algorithm computation to provide corrective action at the control output. The function of the control action is based on an output control, which is proportional to the instantaneous error value. The *integral control action* (reset action) provides additional compensation to the control output, which causes a change in proportion to the value of the change of error over a period of time. The *derivative control action* (rate change) adds compensation to the control output, which causes a change in proportion to the rate of change of error. These three modes are used to provide the desired control action in Proportional (P), Proportional-Integral (PI), or Proportional-Integral-Derivative (PID) control fashion.

Standard DL05/06 analog input modules are used to interface to field transmitters to obtain the PV. These transmitters normally provide a 4–20 mA current or an analog voltage of various ranges for the control loop.

For temperature control, thermocouple or RTD can be connected directly to the appropriate module. The PID control algorithm, residing in the CPU memory, receives information from the user program, primarily control parameters and setpoints. Once the CPU makes the PID calculation, the result may be used to directly control an actuator connected to a 4–20 mA current output module to control a valve.

With ***Direct***SOFT, additional ladder logic programming, both time proportioning (e.g., heaters for temperature control) and position actuator (e.g., reversible motor on a valve) type of control schemes can be easily implemented. This chapter will explain how to set up the PID control loop, how to implement the software and how to tune the loop.

The following block diagram shows the key parts of a PID control loop. The path from the PLC to the manufacturing process and back to the PLC is the closed loop control.



Process Control Definitions

Manufacturing Process – the set of actions that adds value to raw materials. The process can involve physical changes and/or chemical changes to the material. The changes render the material more useful for a particular purpose, ultimately used in a final product.

Process Variable – The controlled variable part of the process that you wish to control. It may be temperature, pressure, level, flow, composition, density, the ratio of two streams, etc. Also known as the actual value.

Setpoint – This is the target for the process variable. When all conditions of the process are correct, the process variable will equal the setpoint.

Control Output – The result of the loop calculation, which becomes a command for the process (such as the heater level in an oven). This is sometimes referred to as control variable.

Error Term – The algebraic difference between the process variable and the setpoint. This is the control loop error, and is equal to zero when the process variable is equal to the setpoint (desired) value. A well-behaved control loop is able to maintain a small error term magnitude.

Manipulated Variable – This is what is used to effect the controlled variable. For example, the fuel used in a furnace might be manipulated in order to control the temperature.

Disturbance – Something in the system that changes such that corrective action is required. For instance, when controlling a flow and the upstream pressure drops, the control valve must open wider in order to keep flow constant. The drop in upstream pressure is the disturbance.

Final Control Element – This is the physical device used to control the manipulated variable. Valves are probably the most widely used final control element.

Lag Time – The time it takes for the process to respond to a change in manipulated variable. This is also known as the capacitance of the system. When you're in the shower and you turn up the hot water a little, the time it takes before the water gets hot is the lag time.

Dead Time – The time it takes for a change in the process to be recognized. Composition analyzers and quality control are usually sources of significant dead time.

Loop Configuring – Operator-initiated selections which set up and optimize the performance of a control loop. The loop calculation function uses the configuration parameters in real time to adjust gains, offsets, etc.

Loop Monitoring – The function which allows an operator to observe the status and performance of a control loop. This is used in conjunction with the loop configuring to optimize the performance of a loop (minimize the error term).

PID Loop Operation

The Proportional–Integral–Derivative (PID) algorithm is widely used in process control. The PID method of control adapts well to electronic solutions, whether implemented in analog or digital (CPU) components. The DL06 CPU implements the PID equations digitally by solving the basic equations in software. I/O modules serve only to convert electronic signals into digital form (or vice versa).

The DL06 uses two types of PID controls: "position" and "velocity". These terms usually refer to motion control situations, but here we use them in a different sense:

- PID *Position* Algorithm – The control output is calculated so it responds to the displacement (position) of the PV from the SP (error term).
- PID *Velocity* Algorithm – The control output is calculated to represent the rate of change (velocity) for the PV to become equal to the SP.

Position Form of the PID Equation

Referring to the control output equation on page 8-6, the DL06 CPU approximates the output M(t) using a discrete position form of the PID equation.

- **Let:**

  Ts = Sample rate
  Kc = Proportional gain
  Ki = Kc * (Ts/Ti) = Coefficient of integral term
  Kr = Kc * (Td/Ts) = Coefficient of derivative term
  Ti = Reset or integral time
  Td = Derivative time or rate
  SP = Setpoint
  PV_n = Process variable at n^{th} sample
  en = SP – PV_n = Error at n^{th} sample
  M_o = Value to which the controller output has been initialized

- **Then:**

  M_n = Control output at n^{th} sample

$$\mathbf{M_n = Kc * e_n + Ki \sum_{i=1}^{n} e_i + Kr\ (e_n - e_{n-1}) + M_o}$$

This form of the PID equation is referred to as the position form since the actual actuator position is computed. The velocity form of the PID equation computes the change in actuator position. The CPU modifies the standard equation slightly to use the derivative of the process variable instead of the error as follows:

$$\mathbf{M_n = Kc * e_n + Ki \sum_{i=1}^{n} e_i + Kr\ (PV_n - PV_{n-1}) + M_o}$$

These two forms are equivalent unless the setpoint is changed. In the original equation, a large step change in the setpoint will cause a correspondingly large change in the error resulting in a bump to the process due to derivative action. This bump is not present in the second form of the equation.

The DL06 also combines the integral sum and the initial output into a single term called the bias (Mx). This results in the following set of equations:

$$Mx_o = M_o$$

$$Mx = Ki * e_n + Mx_{n-1}$$

$$M_n = Kc * e_n - Kr(PV_n - PV_{n-1}) + Mx_n$$

The DL06 by default will keep the normalized output M in the range of 0.0 to 1.0. This is done by clamping M to the nearer of 0.0 or 1.0 whenever the calculated output falls outside this range. The DL06 also allows you to specify the minimum and maximum output limit values (within the range 0 to 4095 in BCD if using 12 bit unipolar).

NOTE: *The equations and algorithms, or parts of, in this chapter, are only for references. Analysis of these equations can be found in most good text books about process control.*

Reset Windup Protection

Reset windup can occur if reset action (integral term) is specified and the computation of the bias term Mx is:

$$Mx = Ki * e_n + Mx_{n-1}$$

For example, assume the output is controlling a valve and the PV remains at some value greater than the setpoint. The negative error (e_n) will cause the bias term (Mx) to constantly decrease until the output M goes to 0 closing the valve. However, since the error term is still negative, the bias will continue to decrease becoming ever more negative. When the PV finally does come back down below the SP, the valve will stay closed until the error is positive for long enough to cause the bias to become positive again. This will cause the process variable to undershoot.

One way to solve the problem is to simply clamp the normalized bias between 0.0 and 1.0. The DL06 CPU does this. However, if this is the only thing that is done, then the output will not move off 0.0 (thus opening the valve) until the PV has become less than the SP. This will also cause the process variable to undershoot.

The DL06 CPU is programmed to solve the overshoot problem by either freezing the bias term, or by adjusting the bias term.

Freeze Bias

If the "Freeze Bias" option is selected when setting up the PID loop (discussed later) then the CPU simply stops changing the bias (Mx) whenever the computed normalized output (M) goes outside the interval 0.0 to 1.0.

$$Mx = Ki * e_n + Mx_{n-1}$$

$$M = Kc * e_n - Kr(PV_n - PV_{n-1}) + Mx$$

$$M_n = 0 \quad \text{if } M < 0$$
$$M_n = M \quad \text{if } 0 \le M \le 1$$
$$M_n = 1 \quad \text{if } M > 1$$

$$Mx_n = Mx \quad \text{if } 0 \le M \le 1$$
$$Mx_n = Mx_{n-1} \quad \text{otherwise}$$

Thus in this example, the bias will probably not go all the way to zero so that, when the PV does begin to come down, the loop will begin to open the valve sooner than it would have if the bias had been allowed to go all the way to zero. This action has the effect of reducing the amount of overshoot.

Adjusting the Bias

The normal action of the CPU is to adjust the bias term when the output goes out of range as shown below.

$$Mx = Ki * e_n + Mx_{n-1}$$

$$M = Kc * e_n - Kr(PV_n - PV_{n-1}) + Mx$$

$$M_n = 0 \quad \text{if } M < 0$$
$$M_n = M \quad \text{if } 0 \le M \le 1$$
$$M_n = 1 \quad \text{if } M > 1$$

$$Mx_n = Mx \quad \text{if } 0 \le M \le 1$$
$$Mx_n = M_n - Kc * e_n - Kr(PV_n - PV_{n-1}) \quad \text{otherwise}$$

By adjusting the bias, the valve will begin to open as soon as the PV begins to come down. If the loop is properly tuned, overshoot can be eliminated entirely. If the output went out of range due to a setpoint change, then the loop probably will oscillate because we must wait for the bias term to stabilize again.

The choice of whether to use the default loop action or to freeze the bias is dependent on the application. **If large, step changes to the setpoint are anticipated, then it is probably better to select the freeze bias option** (see page 8-34).

Step Bias Proportional to Step Change in SP

This feature reduces oscillation caused by a step change in setpoint when the adjusting bias feature is used.

$Mx = Mx * SP_n / SP_{n-1}$ if the loop is direct acting

$Mx = Mx * SP_{n-1} / SP_n$ if the loop is reverse acting

$Mx_n = 0$ if $Mx < 0$
$Mx_n = Mx$ if $0 \le Mx \le 1$
$Mx_n = 1$ if $M > 1$

Eliminating Proportional, Integral or Derivative Action

It is not always necessary to run a full three mode PID control loop. Most loops require only the PI terms or just the P term. Parts of the PID equation may be eliminated by choosing appropriate values for the gain (Kc), reset (Ti) and rate (Td) yielding a P, PI, PD, I and even an ID and a D loop.

Eliminating Integral Action The effect of integral action on the output may be eliminated by setting Ti = 9999. When this is done, the user may then manually control the bias term (Mx) to eliminate any steady-state offset.

Eliminating Derivative Action The effect of derivative action on the output may be eliminated by setting Td = 0 (most loops do not require a D parameter; it may make the loop unstable).

Eliminating Proportional Action Although rarely done, the effect of proportional term on the output may be eliminated by setting Kc = 0. Since Kc is also normally a multiplier of the integral coefficient (Ki) and the derivative coefficient (Kr), the CPU makes the computation of these values conditional on the value of Kc as follows:

$Ki = Kc * (Ts / Ti)$ if $Kc \neq 0$
$Ki = Ts / Ti$ if $Kc = 0$ (I or ID only)
$Kr = Kc * (Td / Ts)$ if $Kc \neq 0$
$Kr = Td / Ts$ if $Kc = 0$ (ID or D only)

Velocity Form of the PID Equation

The standard position form of the PID equation computes the actual actuator position. An alternative form of the PID equation computes the change in actuator position. This form of the equation is referred to as the velocity PID equation and is obtained by subtracting the equation at time "n" from the equation at time "n-1".

The velocity equation is given by:

$\Delta M_n = M - M_{n-1}$

$\Delta M_n = Kc * (e_n - e_{n-1}) + Ki * (PV_n - 2 * PV_{n-1} + PV_{n-2})$

Bumpless Transfer

The DL06 loop controller provides for bumpless mode changes. A bumpless transfer from manual mode to automatic mode is achieved by preventing the control output from changing immediately after the mode change.

When a loop is switched from Manual mode to Automatic mode, the setpoint and Bias are initialized as follows:

- **Position PID Algorithm**
  SP = PV
  Mx = M

  Velocity PID Algorithm
  SP = PV

The bumpless transfer feature of the DL06 is available in two types: Bumpless I and Bumpless II (see page 8-26). The transfer type is selected when the loop is set up.

Loop Alarms

The DL06 allows the user to specify alarm conditions that are to be monitored for each loop. Alarm conditions are reported to the CPU by setting up the alarms in ***Direct***SOFT using the PID setup alarm dialog when the loop is setup. The alarm features for each loop are:

- **PV Limit** – Specify up to four PV alarm points.

High-High	PV rises above the programmed High-High Alarm Limit.
High	PV rises above the programmed High Alarm Limit.
Low	PV fails below the Low Alarm Limit.
Low-Low	PV fails below the Low-Low Limit.

- **PV Deviation Alarm** – Specify an alarm for High and Low PV deviation from the setpoint (Yellow Deviation). An alarm for High-High and Low-Low PV deviation from the setpoint (Orange Deviation) may also be specified. When the PV is further from the setpoint than the programmed Yellow or Orange Deviation Limit the corresponding alarm bit is activated.

- **Rate of Change** – This alarm is set when the PV changes faster than a specified rate-of-change limit.

- **PV Alarm Hysteresis** – The PV Limit Alarms and PV Deviation Alarms are programmed using threshold values. When the absolute value or deviation exceeds the threshold, the alarm status becomes true. Real-world PV signals have some noise on them, which can cause some fluctuation in the PV value in the CPU. As the PV value crosses an alarm threshold, its fluctuations will cause the alarm to be intermittent and annoy process operators. The solution is to use the PV Alarm Hysteresis feature.

Loop Operating Modes

The DL06 loop controller operates in one of three modes, either *Manual , Automatic* or *Cascade*.

Manual

In manual mode, the control output is determined by the operator, not the loop controller. While in manual mode, the loop controller will still monitor all of the alarms including High-High, High, Low, Low-Low, Yellow deviation, Orange deviation and Rate-of-Change.

Automatic

In automatic mode, the loop controller computes the control output based on the programmed parameters stored in V-memory. All alarms are monitored while in automatic.

Cascade

Cascade mode is an option with the DL06 PLC and is used in special control applications. If the cascade feature is used, the loop will operate as it would if in automatic mode except for the fact that a cascaded loop has a setpoint which is the control output from another loop.

Special Loop Calculations

Reverse Acting Loop

Although the PID algorithm is used in a direct, or forward, acting loop controller, there are times when a reverse acting control output is needed. The DL06 loop controller allows a loop to operate as reverse acting. With a reverse acting loop, the output is driven in the opposite direction of the error. For example, if SP > PV, then a reverse acting controller will decrease the output to increase the PV.

$$Mx = -Ki * e_n + Mx_{n-1}$$

$$M = -Kc * e_n + Kr(PV_n - PV_{n-1}) + Mx_n$$

Square Root of the Process Variable

Square root is selected whenever the PV is from a device such as an orifice meter which requires this calculation.

Error Squared Control

Whenever error squared control is selected, the error is calculated as:

$$e_n = (SP - PV_n) * ABS(SP - PV_n)$$

A loop using the error squared is less responsive than a loop using just the error, however, it will respond faster with a large error. The smaller the error, the less responsive the loop. Error squared control would typically be used in a PH control application.

Error Deadband Control

With error deadband control, no control action is taken if the PV is within the specified deadband area around the setpoint. The error deadband is the same above and below the setpoint.

Once the PV is outside of the error deadband around the setpoint, the entire error is used in the loop calculation.

$e_n = 0$ SP - Deadband_Below_SP < PV < SP - Deadband_Above_SP
$e_n = P - PV_n$ otherwise

The error will be squared first if both Error Squared and Error Deadband is selected.

Derivative Gain Limiting

When the coefficient of the derivative term, Kr, is a large value, noise introduced into the PV can result in erratic loop output. This problem is corrected by specifying a derivative gain limiting coefficient, Kd. Derivative gain limiting is a first order filter applied to the derivative term computation, Y_n, as shown below.

$$Y_n = Y_{n-1} + \frac{Ts}{Ts + (\frac{Td}{Kd})} * (PV_n - Y_{n-1})$$

Position Algorithm

$$Mx = Ki * e_n + Mx_{n-1}$$
$$M = Kc * e_n - Kr * (Y_n - Y_{n-1}) + Mx$$

Velocity Algorithm

$$\Delta M = Kc * (e_n - e_{n-1}) + Ki * e_n - Kr * (Y_n - 2 * Y_{n-1} + Y_{n-2})$$

Ten Steps to Successful Process Control

Controllers such as the DL06 PLC provide sophisticated process control features. Automated control systems can be difficult to debug, because a given symptom can have many possible causes. We recommend a careful, step-by-step approach to bringing new control loops online:

Step 1: Know the Recipe

The most important is – how to produce your product. This knowledge is the foundation for designing an effective control system. A good process *recipe* will do the following:

- Identify all relevant Process Variables, such as temperature, pressure, or flow rates, etc., which need precise control.
- Plot the desired Setpoint values for each process variables for the duration of one process cycle.

Step 2: Plan Loop Control Strategy

This simply means choosing the method the machine will use to maintain control over the Process Variables to follow their Setpoints. This involves many issues and trade-offs, such as energy efficiency, equipment costs, ability to service the machine during production, and more. You must also determine how to generate the Setpoint value during the process, and whether a machine operator can change the SP.

Step 3: Size and Scale Loop Components

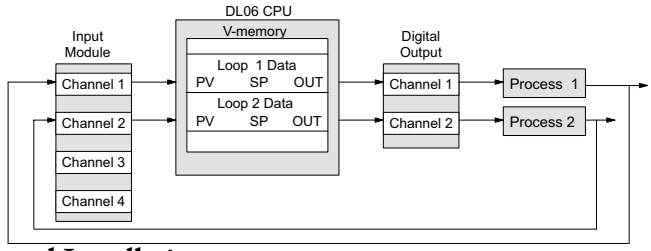
Assuming the control strategy is sound, it is still crucial to *properly size the actuator and properly scale the sensors.*

- Choose an actuator (heater, pump. etc.) which matches the size of the load. An oversized actuator will have an overwhelming effect on your process after a SP change. However, an undersized actuator will allow the PV to lag or drift away from the SP after a SP change or process disturbance.
- Choose a PV sensor which matches the range of interest (and control) for our process. Decide the resolution of control you need for the PV (such as within 2°C), and make sure the sensor input value provides the loop with at least 5 times that resolution (at LSB level). However, an over-sensitive sensor can cause control oscillations, etc. The DL06 provides 12–bit and 15–bit unipolar and bipolar data format options, and a 16–bit unipolar option. This selection affects SP, PV, Control Output and Integrator sum.

Step 4: Select I/O Modules

After deciding the number of loops, PV variables to measure, and SP values, you can choose the appropriate I/O module. Refer to the figure on the next page. In many cases, you will be able to share input or output modules, or use an analog I/O combination module, among several control loops. The example shown sends the PV and Control Output signals for two loops through the same set of modules.

Automationdirect offers DL06 analog input modules with 4 channels per module that accept 0 – 20mA or 4 – 20mA signals. Also, analog input and output combination modules are now available. Thermocouple and RTD modules can also be used to maintain temperatures to a 10th of a degree. Refer to the sales catalog for further information on these modules, or find the modules on our website, **www.automationdirect.com.**



Step 5: Wiring and Installation

- After selection and procurement of all loop components and I/O module(s), you can perform the wiring and installation. Refer to the wiring guidelines in Chapter 2 of this manual, and to the **D0–OPTIONS–M** manual. The most common wiring errors when installing PID loop controls are:
- Reversing the polarity of sensor or actuator wiring connections.
- Incorrect signal ground connections between loop components.

Step 6: Loop Parameters

After wiring and installation, choose the loop setup parameters. The easiest method for programming the loop tables is using ***Direct***SOFT (5.0 or later). This software provides PID Setup using dialog boxes to simplify the task. **Note: It is important to understand the meaning of all loop parameters mentioned in this chapter before choosing values to enter.**

Step 7: Check Open Loop Performance

With the sensor and actuator wiring done, and loop parameters entered, we must manually and carefully check out the new control system using the Manual mode.

- Verify that the PV value from the sensor is correct.
- If it is safe to do so, gradually increase the control output up above 0%, and see if the PV responds (and moves in the correct direction!).

Step 8: Loop Tuning

If the Open Loop Test (page 8–40) shows the PV reading is correct and the control output has the proper effect on the process; you can follow the closed loop tuning procedure (see page 8–45). In this step, the loop is tuned so the PV automatically follows the SP.

Step 9: Run Process Cycle

If the closed loop test shows the PV will follow small changes in the SP, consider running an actual process cycle. You will need to have completed the programming which will generate the desired SP in real time. In this step, you may want to run a small test batch of product through the machine, watching the SP change according to the recipe.

WARNING: Be sure the Emergency Stop and power-down provision is readily accessible, in case the process goes out of control. Damage to equipment and/or serious injury to personnel can result from loss of control of some processes.

Step 10: Save Parameters

When the loop tests and tuning sessions are complete, be sure to save all loop setup parameters to disk.

PID Loop Setup

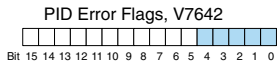
Some Things to Do and Know Before Starting

Have your analog module installed and operational before beginning the loop setup (refer to the DL05/06 Option Modules User Manual, D0-OPTIONS-M). The DL06 PLC gets its PID loop processing instructions from V-memory tables. There isn't a PID instruction that can be used in RLL, such as a block, to setup the PID loop control. Instead, the CPU reads the setup parameters from system V-memory locations. These locations are shown in the table below for reference only; they can be used in a RLL program if needed.

Address	Setup Parameter	Data type	Ranges	Read/Write
V7640	Loop Parameter Table Pointer	Octal	V1200 – V7340 V10000-V17740	Write
V7641	Number of Loops	BCD	1 – 8	Write
V7642	Loop Error Flags	BITS	0 or 1	Read

NOTE: *The V-memory data is stored in SRAM memory. If power is removed from the CPU for an extended period of time, the PID Setup Parameters will be lost. It is recommended to use the optional battery backup to retain the memory in SRAM. Another option is to use the MOV instruction, which places the data in non-volatile memory, when setting up the parameters in the ladder program.*

PID Error Flags



The CPU reports any programming errors of the setup parameters in V7640 and V7641. It does this by setting the appropriate bits in V7642 on program-to-run mode transitions.

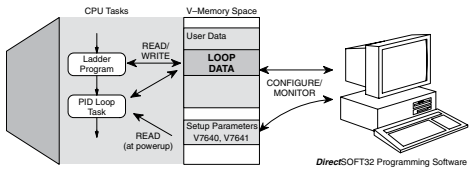
If you use the *Direct*SOFT loop setup dialog box, its automatic range checking prohibits possible setup errors. However, the setup parameters may be written using other methods such as RLL, so the error flag register may be helpful in those cases. The following table lists the errors reported in V7642.

As a quick check, if the CPU is in Run mode and V7642=0000, there are no programming errors.

Bit	Error Description (0 = no error, 1 = error)
0	The starting address (in V7640) is out of the lower V-memory range.
1	The starting address (in V7640) is out of the upper V-memory range.
2	The number of loops selected (in V7641) is greater than 8.
3	The loop table extends past (straddles) the boundary at V7377. Use an address closer to V1200.

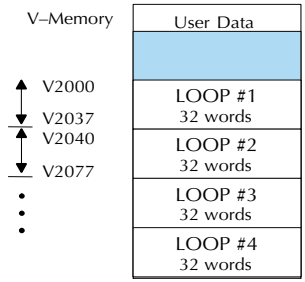
Establishing the Loop Table Size and Location

On a PROGRAM-to-RUN mode transition, the CPU reads the loop setup parameters as pictured below. At that moment, the CPU learns the location of the loop table and the number of loops it configures. Then during the ladder program scan, the PID Loop task uses the loop data to perform calculations, generate alarms, and so on. There are some loop table parameters the CPU will read or write on every loop calculation.



***NOTE**: The DL06 CPU's PID algorithm requires **Direct**SOFT Version 5.0 (or later) and firmware version 2.1 (or later). See our website for more information: **www.automationdirect.com**.*

The Loop Table contains data for only the number of loops that are selected. The address for the table is stored in V7641. Each loop configuration occupies 32 words (0 to 37 octal) in the loop table.



For example, consider an application with 4 loops, and V2000 has been chosen as the starting location. The Loop Parameter will occupy V2000 – V2037 for loop 1, V2040 – V2077 for loop 2 and so on. Loop 4 occupies V2140 - V2177.

Determine the block of V-memory to be used for each PID loop. Besides being the beginning of the PID parameter memory block, the first address will be the start of loop 1 parameters. Remember, there are 32 words (0 to 37 octal) needed for each loop. Once you have determined the beginning V-memory address to be used, you can setup and store the PID parameters either directly in your RLL program or by the using PID Setup in *Direct*SOFT.

NOTE:** Whether one or more loops are being setup, this block of V-memory will only be used for the PID loop parameters, **do not use this block of memory for anything else in your program.

Using *Direct*SOFT is the simplest way to setup the parameters. To setup the PID parameters, the DL06 must be powered up and connected to the programming computer. The parameters can only be entered in PID setup when the PLC is in the Program mode. Once the parameters have been entered and saved for each loop, changes made through the PID setup can be made, but only in Program Mode. You can type the beginning address in the PID Table Address dialog found when the PID Setup is opened in *Direct*SOFT. This can be seen in the diagram below. After the address has been entered, the memory range will appear. Also, entering the number of PID loops (1 to 8) will set the total V-memory range for the number of loops The Loop Table contains data for only the number of loops that are selected. The address for the table is stored in V7641. Each loop configuration occupies 32 words (0 to 37 octal) in the loop table.

***NOTE:** Have an edited program open, then click on PLC > Setup > PID to access the Setup PID dialog.*

Loop Table Word Definitions

These are the loop parameters associated with each of the four loops available in the DL06. The parameters are listed in the following table. The address offset is in octal, to help you locate specific parameters in the loop table. For example, if a table begins at V2000, then the location of the reset (integral) term is Addr+11, or V2011. Do not use the Word # (in the first column) to calculate addresses.

Word #	Address+Offset	Description	Format	Read on-the-fly***
1	Addr + 0	PID Loop Mode Setting 1	bits	Yes
2	Addr + 1	PID Loop Mode Setting 2	bits	Yes
3	Addr + 2	Setpoint Value (SP)	word/binary	Yes
4	Addr + 3	Process Variable (PV)	word/binary	Yes
5	Addr + 4	Bias (Integrator) Value	word/binary	Yes
6	Addr + 5	Control Output Value	word/binary	Yes
7	Addr + 6	Loop Mode and Alarm Status	bits	–
8	Addr + 7	Sample Rate Setting	word/BCD	Yes
9	Addr + 10	Gain (Proportional) Setting	word/BCD	Yes
10	Addr + 11	Reset (Integral) Time Setting	word/BCD	Yes
11	Addr + 12	Rate (Derivative) Time Setting	word/BCD	Yes
12	Addr + 13	PV Value, Low-low Alarm	word/binary	No*
13	Addr + 14	PV Value, Low Alarm	word/binary	No*
14	Addr + 15	PV Value, High Alarm	word/binary	No*
15	Addr + 16	PV Value, High-high Alarm	word/binary	No*
16	Addr + 17	PV Value, deviation alarm (YELLOW)	word/binary	No*
17	Addr + 20	PV Value, deviation alarm (RED)	word/binary	No*
18	Addr + 21	PV Value, rate-of-change alarm	word/binary	No*
19	Addr + 22	PV Value, alarm hysteresis setting	word/binary	No*
20	Addr + 23	PV Value, error deadband setting	word/binary	Yes
21	Addr + 24	PV low-pass filter constant	word/BCD	Yes
22	Addr + 25	Loop derivative gain limiting factor setting	word/BCD	No**
23	Addr + 26	SP value lower limit setting	word/binary	Yes
24	Addr + 27	SP value upper limit setting	word/binary	Yes
25	Addr + 30	Control output value lower limit setting	word/binary	No**
26	Addr + 31	Control output value upper limit setting	word/binary	No**
27	Addr + 32	Remote SP Value V-Memory Address Pointer	word/hex	Yes
28	Addr + 33	Ramp/Soak Setting Flag	bit	Yes
29	Addr + 34	Ramp/Soak Programming Table Starting Address	word/hex	No**
30	Addr + 35	Ramp/Soak Programming Table Error Flags	bits	No**
31	Addr + 36	PV auto transfer, channel number	word/hex	Yes
32	Addr + 37	Control output auto transfer, channel number	word/hex	Yes

* Read data only when alarm enable bit transitions from 0 to 1.
** Read data only on PLC Mode change.
*** Read on-the-fly means that the content of V-memory can be changed while the PID loop is in operation.

PID Mode Setting 1 Bit Descriptions (Addr + 00)

The individual bit definitions of the PID Mode Setting 1 word (Addr+00) are listed in the following table.

Bit	PID Mode Setting 1 Description	Read/Write	Bit=0	Bit=1
0	Manual Mode Loop Operation request	Write	–	01 request
1	Automatic Mode Loop Operation request	Write	–	01 request
2	Cascade Mode Loop Operation request	Write	–	01 request
3	Bumpless Transfer select	Write	Mode I	Mode II
4	Direct or Reverse-Acting Loop select	Write	Direct	Reverse
5	Position / Velocity Algorithm select	Write	Position	Velocity
6	PV Linear / Square Root Extract select	Write	Linear	Sq. root
7	Error Term Linear / Squared select	Write	Linear	Squared
8	Error Deadband enable	Write	Disable	Enable
9	Derivative Gain Limit select	Write	Off	On
10	Bias (Integrator) Freeze select	Write	Off	On
11	Ramp/Soak Operation select	Write	Off	On
12	PV Alarm Monitor select	Write	Off	On
13	PV Deviation alarm select	Write	Off	On
14	PV rate-of-change alarm select	Write	Off	On
15	Loop mode is independent from CPU mode when set	Write	Loop with CPU mode	Loop Independent of CPU mode

PID Mode Setting 2 Bit Descriptions (Addr + 01)

The individual bit definitions of the PID Mode Setting 2 word (Addr+01) are listed in the following table.

Bit	PID Mode 2 Word Description	Read/Write	Bit=0	Bit=1
0	Input (PV) and Control Output Range Unipolar/Bipolar select (See Notes 2 and 3)	Write	Unipolar	Bipolar
1	Input/Output Data Format select (See Notes 2 and 3)	Write	12-bit	15-bit
2	Analog Input filter	Write	Off	On
3	SP Input limit enable	Write	Disable	Enable
4	Integral Gain (Reset) units select	Write	Seconds	Minutes
5	Select Auto tune PID algorithm	Write	Closed loop	Open loop
6	Auto tune selection	Write	PID	PI only (rate = 0)
7	Auto tune start (See Note 1)	Read/Write	Auto tune cancel/done	Force start
8	PID Scan Clock (internal use)	Read	–	–
9	Input/Output Data Format 16-bit select (See Notes 1 and 2)	Write	Not 16-bit	Select 16-bit
10	Select separate data format for input and output (See Notes 2, and 3)	Write	Same format	Separate formats
11	Control Output Range Unipolar/Bipolar select See Notes 2, and 3)	Write	Unipolar	Bipolar
12	Output Data Format select (See Notes 2, and 3)	Write	12-bit	15-bit
13	Output data format 16-bit select (See Notes 2, and 3)	Write	Not 16-bit	Select 16-bit
14–15	Reserved for future use	–	–	–

NOTE 1: *Bit 7 can be used to cancel Autotune mode by setting it to 0.*

NOTE 2: *If the value in bit 9 is 0, then the values in bits 0 and 1 are read. If the value in bit 9 is 1, then the values in bits 0 and 1 are not read, and bit 9 defines the data format (the range is automatically unipolar).*

NOTE 3: *If the value in bit 10 is 0, then the values in bits 0, 1 and 9 define the input and output ranges and data formats (the values in bits 11, 12, and 13 are not read). If the value in bit 10 is 1, then the values in bits 0, 1, and 9 define only the input range and data format, and bits 11, 12, and 13 are read and define the output range and data format.*

NOTE 4: *If bit 10 has a value of 1 and bit 13 has a value of 0, then bits 11 and 12 are read and define the output range and data format. If bit 10 and bit 13 each have a value of 1, then bits 11 and 12 are not read, and bit 13 defines the data format, (the output range is automatically unipolar).*

Mode/Alarm Monitoring Word (Addr + 06)

The individual bit definitions of the Mode / Alarm monitoring (Addr+06) word are listed in the following table.

Bit	Mode/Alarm Bit Description	Read/Write	Bit=0	Bit=1
0	Manual Mode Indication	Read	–	Manual
1	Automatic Mode Indication	Read	–	Auto
2	Cascade Mode Indication	Read	–	Cascade
3	PV Input LOW–LOW Alarm	Read	Off	On
4	PV Input LOW Alarm	Read	Off	On
5	PV Input HIGH Alarm	Read	Off	On
6	PV Input HIGH–HIGH Alarm	Read	Off	On
7	PV Input YELLOW Deviation Alarm	Read	Off	On
8	PV Input RED Deviation Alarm	Read	Off	On
9	PV Input Rate-of-Change Alarm	Read	Off	On
10	Alarm Value Programming Error	Read	–	Error
11	Loop Calculation Overflow/Underflow	Read	–	Error
12	Loop in Auto-Tune indication	Read	Off	On
13	Auto-Tune error indication	Read	–	Error
14–15	Reserved for Future Use	–	–	–

Ramp/Soak Table Flags (Addr + 33)

The individual bit definitions of the Ramp/Soak Table Flag (Addr+33) word are listed in the following table.

Bit	Ramp/Soak Flag Bit Description	Read/Write	Bit=0	Bit=1
0	Start Ramp / Soak Profile	Write	–	01 Start
1	Hold Ramp / Soak Profile	Write	–	01 Hold
2	Resume Ramp / soak Profile	Write	–	01 Resume
3	Jog Ramp / Soak Profile	Write	–	01 Jog
4	Ramp / Soak Profile Complete	Read	–	Complete
5	PV Input Ramp / Soak Deviation	Read	Off	On
6	Ramp / Soak Profile in Hold	Read	Off	On
7	Reserved	Read	–	–
8–15	Current Step in R/S Profile	Read	Decode as byte (hex)	

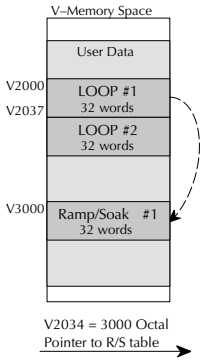
Bits 8–15 must be read as a byte to indicate the current segment number of the Ramp/Soak generator in the profile. This byte will have the values 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, and 10, which represent segments 1 to 16 respectively. If the byte=0, then the Ramp/Soak table is not active.

Ramp/Soak Table Location (Addr + 34)

Each loop that you configure has the option of using a built-in Ramp/Soak generator dedicated to that loop. This feature generates SP values that follow a profile. To use the Ramp Soak feature, you must program a separate table of 32 words with appropriate values. The *Direct*SOFT dialog box makes this easy to do.

In the loop table, the Ramp/Soak Table Pointer at Addr+34 must point to the start of the ramp/soak data for that loop. This may be anywhere in user memory, and does not have to adjoin to the Loop Parameter table, as shown to the left. Each R/S table requires 32 words, regardless of the number of segments programmed.

The ramp/soak table parameters are defined in the table below. Further details are in the section on Ramp/Soak Operation in this chapter.



Addr Offset	Step	Description	Addr Offset	Step	Description
+ 00	1	Ramp End SP Value	+ 20	9	Ramp End SP Value
+ 01	1	Ramp Slope	+ 21	9	Ramp Slope
+ 02	2	Soak Duration	+ 22	10	Soak Duration
+ 03	2	Soak PV Deviation	+ 23	10	Soak PV Deviation
+ 04	3	Ramp End SP Value	+ 24	11	Ramp End SP Value
+ 05	3	Ramp Slope	+ 25	11	Ramp Slope
+ 06	4	Soak Duration	+ 26	12	Soak Duration
+ 07	4	Soak PV Deviation	+ 27	12	Soak PV Deviation
+ 10	5	Ramp End SP Value	+ 30	13	Ramp End SP Value
+ 11	5	Ramp Slope	+ 31	13	Ramp Slope
+ 12	6	Soak Duration	+ 32	14	Soak Duration
+ 13	6	Soak PV Deviation	+ 33	14	Soak PV Deviation
+ 14	7	Ramp End SP Value	+ 34	15	Ramp End SP Value
+ 15	7	Ramp Slope	+ 35	15	Ramp Slope
+ 16	8	Soak Duration	+ 36	16	Soak Duration
+ 17	8	Soak PV Deviation	+ 37	16	Soak PV Deviation

Ramp/Soak Table Programming Error Flags (Addr + 35)

The individual bit definitions of the Ramp/Soak Table **Programming Error Flags** word (Addr+35) are listed in the following table. Further details are given in the PID Loop Mode section and in the PV Alarm section later in chapter 8.

Bit	R/S Error Flag Bit Description	Read/Write	Bit=0	Bit=1
0	Starting Addr out of lower V-memory range	Read	–	Error
1	Starting Addr out of upper V-memory range	Read	–	Error
2–3	Reserved for Future Use	–	–	–
4	Starting Addr in System Parameter V-memory Range	Read	–	Error
5–15	Reserved for Future Use	–	–	–

Configure the PID Loop

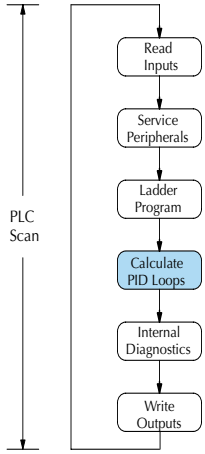
Once the PID table is established in V-memory, configuring the PID loop continues with the *Direct*SOFT PID setup configuration dialog. You will need to check and fill in the data required to control the PID loop. Select Configure and the following dialog will appear for this process.

Select the Algorithm Type

Chose either *Position* or *Velocity*. The default algorithm is Position. This is the choice for most applications which include heating and cooling loops as well as most position and level control loops. A typical velocity control will consist of a process variable such as a flow totalizer in a flow control loop.

Enter the Sample Rate

The main tasks of the CPU fall into categories as shown to the right. The list represents the tasks done when the CPU is in Run Mode, on each PLC scan. Note that PID loop calculations occur after the ladder logic task.



The *sample rate* of a control loop is simply the frequency of the PID calculation. Each calculation generates a new control output value. With the DL06 CPU, you can set the sample rate of a loop from 50 ms to 99.99 seconds. Most loops do not require a fresh PID calculation on every PLC scan. Some loops may need to be calculated only once in 1000 scans.

Enter 0.05 sec., or the sample rate of your choice, for each loop, and the CPU automatically schedules and executes PID calculations on the appropriate scans.

NOTE: *If more than 4 loops are programmed, enter a minimum of 0.1 second.*

Select Forward/Reverse

It is important to know which direction the control output will respond to the error (SP-PV), either *forward* or *reverse.* A forward (direct) acting control loop means that whenever the control output increases, the process variable will also increase. The control outputs of most PID loops are forward acting, such as a heating control loop. An increase in heat applied will increase the PV (temperature).

A reverse acting control loop is one where an increase in the control output results in a decrease in the PV. A common example of this would be a refrigeration system, where an increase in the cooling input causes a decrease in the PV (temperature).

The Transfer Mode

Choose either Bumpless I or Bumpless II to provide a smooth transition of the control output from Manual Mode to Auto Mode. Choosing Bumpless I will set the SP equal to the PV when the control output is switched from Manual to Auto. If this is not desired, choose BumplessII.

The characteristics of Bumpless I and II transfer types are listed in the chart below. Note that their operation also depends on which PID algorithm you are using, the position or velocity form of the PID equation. Note that you must use Bumpless Transfer type I when using the velocity form of the PID algorithm.

Transfer Type	Transfer Select Bit 3	PID Algorithm	Manual-to-Auto Transfer Action	Auto-to-Cascade Transfer Action
Bumpless Transfer I	0	Position	Forces Bias = Control Output Forces SP = PV	Forces Major Loop Output = Minor Loop PV
		Velocity	Forces SP = PV	Forces Major Loop Output = Minor Loop PV
Bumpless Transfer II	1	Position	Forces Bias = Control Output	none
		Velocity	none	none

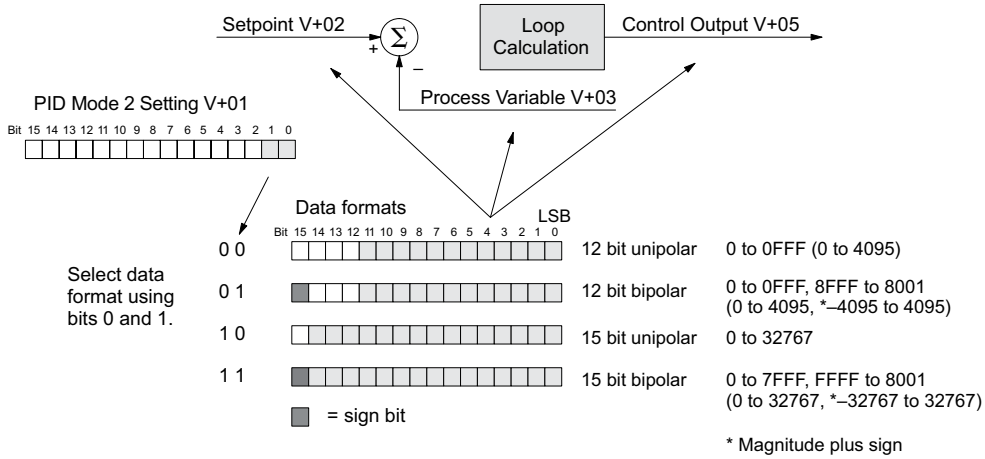
The transfer type can also be selected in an RLL program by setting bit 3 of PID Mode 1, V+00 setting as shown.

SP/PV & Output Format

This block allows you to select either *Common format* or *Independent format*. Common format is the default and is most commonly used. With this format, both SP/PV and Output will have the same data structure. Both will have the same number of bits and either bipolar or unipolar. If Independent format is selected, the data structure selections will be grayed out. The reason for this is that they become independently selectable in the *SP/PV* and the *Output* dialogs.

Common Data Format

Select either *Unipolar data format* (which is positive data only) in 12 bit (0 to 4095), 15 bit (0 to 32767), or 16 bit (0 to 65535) format, or *Bipolar data format,* which ranges from negative to positive (-4095 to 4095 or -32767 to 32767) and requires a sign bit. Bipolar selection displays input/output as magnitude plus sign, not two's complement. The bipolar selection is not available when 16-bit data format is selected.



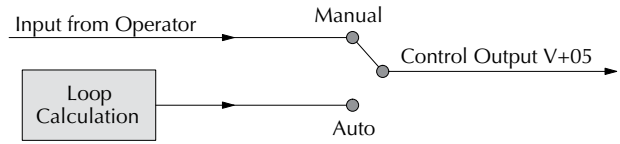
The data format determines the numerical interface between the PID loop and the PV sensor and the control output device. This selects the data format for both the SP and the PV.

Loop Mode

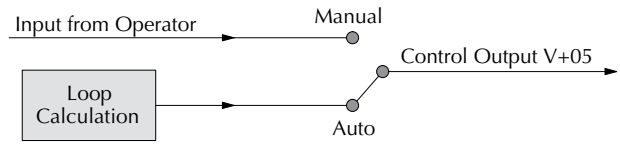
Loop Mode is a special feature that allows the PID loop controller to perform closed-loop control while the CPU is in the Program Mode. Careful thought must be taken before using this feature called *Independent of CPU mode* in the dialog. Before continuing with the PID setup, a knowledge of the three PID loop modes will be helpful.

The DL06 provides the three standard control modes: *Manual, Automatic,* and *Cascade*. The sources of the three basic variables SP, PV and control output are different for each mode.

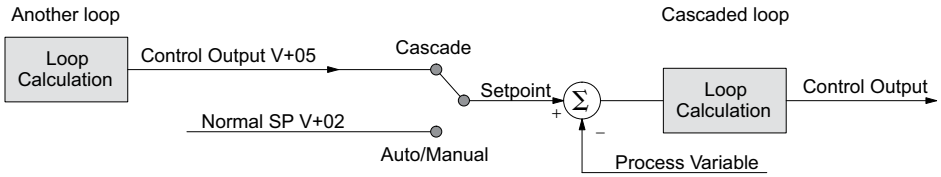
In Manual Mode, the loop is not executing PID calculations (however, loop alarms are still active). With regard to the loop table, the CPU stops writing values to location V+05 (control output) for that loop. *It is expected that an operator or other intelligent source* is manually controlling the output by observing the PV and writing data to the control output as necessary to keep the process under control. The drawing below shows the equivalent schematic diagram of manual mode operation.



In Automatic Mode, the loop operates normally and generates new control output values. It calculates the PID equation and writes the result in location V+05 every sample period of that loop. The equivalent schematic diagram is shown below.



In Cascade Mode, the loop operates as it does in Automatic Mode, with one important difference. The data source for the SP changes from its normal location at V+02 to using the control output value, V+05, from another loop. So in Auto or Manual modes, the loop calculation uses the data at V+02. In Cascade Mode, the loop calculation reads the control

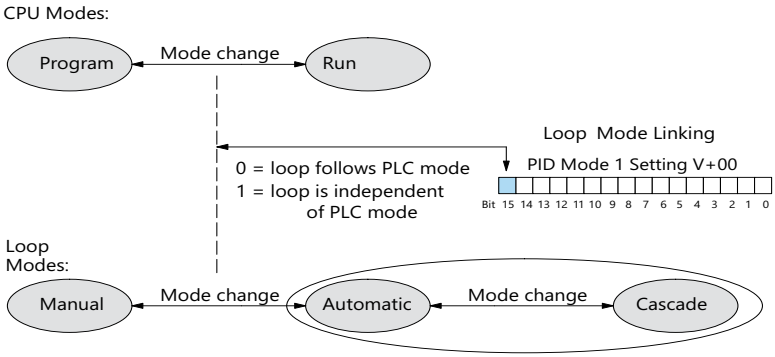


output from another loop's parameter table, V+05.

As pictured below, a loop can be changed from one mode to another, but *cannot go from*

Manual Mode directly to Cascade, or vice versa. This mode change is prohibited because a loop would be changing two data sources at the same time, and could cause a loss of control.

Once the CPU is operating in the Run Mode, the normal operation of the PID loop controller is to read the loop data and perform calculations on each scan of the RLL program. When the CPU is placed in the Program Mode, the RLL program halts operation and all PID loops are automatically put into the Manual Mode. The PID parameters can then be changed if desired.



Similarly, by placing the CPU in the Run mode, the PID loops are returned to the operational mode which they were previously in, i.e., Manual, Automatic and Cascade. With this selection you automatically affect the modes by changing the CPU mode.

If bit 15 is set to one, then the loops will run independently of the CPU mode. It is like having two independent processors in the CPU... one is running the RLL program and the other is running the process loops.

Having the ability to run loops independently of the RLL program makes it feasible to make a ladder logic change while the process is still running. This is especially beneficial for large-mass continuous processes that are difficult or costly to interrupt. The independent of CPU is the feature used for this.

If you need to operate the PID loops while the RLL program is halted, in Program Mode, either select the Independent of CPU mode in the dialog or edit your program to set and reset bit 15 of PID Mode 1 word (V+00) in your RLL program. If the bit is set to a zero, the loop will follow the CPU mode; then, when the CPU is placed in the Program Mode, all loops will be forced into the Manual Mode.

When Independent of CPU mode is used, you should also set the PV to be read directly from an analog input module. This can easily be done in the PID setup dialog, SP/PV.

The SP/PV dialog has a block entitled *Process Variable*. There is a block within this block called *Auto Transfer From* (from analog input) with the information grayed out. Checking the box to the left of the Auto Transfer From will highlight the information. Select *I/O Module* then enter the slot number in which the input module resides. Next, select the analog input channel of your choice.

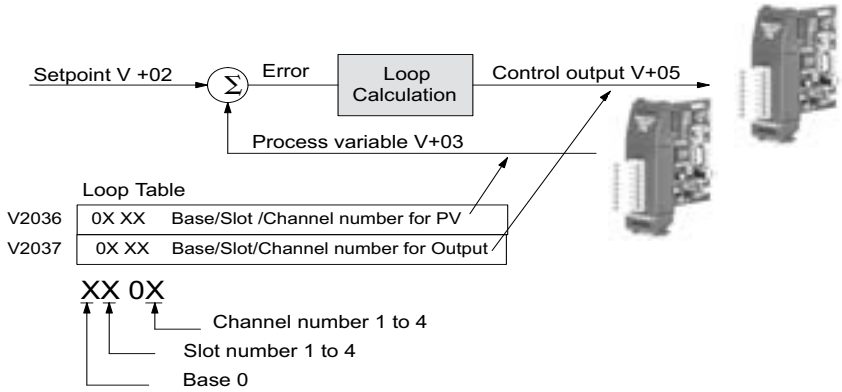
The second choice is *V-Memory*. When this is selected, the V-memory address from where the PV is transferred must be specified.

Whichever method of auto transfer is used, it is recommended to check the *Enable Filter Factor* (a low pass filter) and specify the coefficient.

You should also select the analog output for the control output to be transferred to. This is done in the PID setup *Output* dialog shown here. The block of information in this dialog is grayed-out until the box next to *Auto transfer to I/O module* is checked. Once checked, enter the slot number where the output module is residing and then enter the analog output channel number.

NOTE: *To make changes to any loop table parameters, the PID loop must be in Manual mode and the PLC must be stopped. If you have selected to operate the PID loop independent of the CPU mode, then you must take certain steps to make it possible to make loop parameter changes. You can temporarily make the loops follow the CPU mode by changing bit 15 to 0. Then, you will be able to place the loop into Manual Mode using* ***Direct****SOFT. After you change the loop's parameter settings, restore bit 15 to a value of 1 to re-establish PID operation independent of CPU.*

You may optionally configure each loop to access its analog I/O (PV and control output) by placing proper values in the associated loop table registers in your RLL program. The following figure shows the loop table parameters at V+36 and V+37 and their auto transfer role to access the analog values directly.

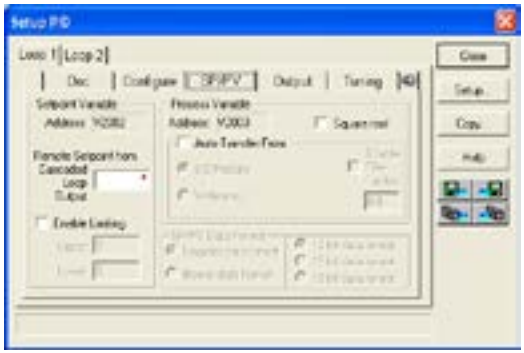


When these loop table parameters are programmed directly, a value of **0102** in register V2036 directs the loop controller to read the PV data from channel 1 of the analog input. A value of **0000** in either register tells the loop controller not to access the corresponding analog value directly. In that case, ladder logic must be used to transfer the value between the analog input and the loop table.

NOTE: *When auto transfer to/from I/O is used, the analog data for all of the channels on the analog module cannot be accessed by any other method, i.e., pointer or multiplex.*

SP/PV Addresses

An SP/PV dialog will be made available to setup how the setpoint (SP) and the process variable (PV) will be used in the loop. If this loop is the minor loop of a cascaded pair, enter that control output address in the *Remote SP from Cascaded Loop Output* area. It is sometimes desirable to limit the range of setpoint values allowed to be entered. To activate this feature, check the box next to *Enable Limiting*. This will activate the *Upper* and *Lower* fields for the values to be



entered. Set the limits around the SP value to prevent an operator from entering a setpoint value outside of a safe range. The *Square root* box is only checked for certain PID loops, such as a flow control loop. If the *Auto transfer from I/O module* is selected, a first-order low-pass filter can be used by checking the *Enable Filter* box. The filter coefficient is user specified. The use of this filter is recommended during closed loop auto-tuning. If the Independent format had been checked previously, make the Data format selections here.

NOTE: *The SP/PV dialog can be left as it first appears for basic PID operation.*

Set Control Output Limits

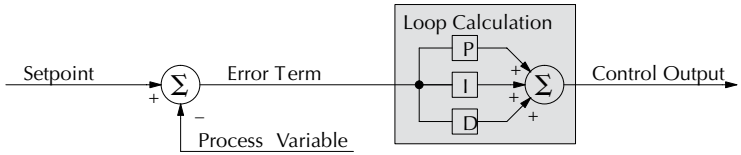
Another dialog that will be available in the PID setup will be the Output dialog. The control output address, V+05, (determined by the PID loop table beginning address) will be in view. Enter the output range limits, *Upper Limit* and *Lower Limit,* that will meet the requirement of the process and which will agree with the data format that has been selected. For a basic PID operation using a 12-bit output module, set the Upper Limit to 4095 and leave the Lower Limit set to 0. Check the box next for *Auto transfer to I/O module* if there is a need to send the control output to a certain analog output module, as in the case of using the Loop Mode independent of CPU Mode; otherwise, the PID output signal cannot control the analog output when the PLC is not in RUN Mode. If the *Auto transfer to I/O module* feature is checked, all channels of the module must be used for PID control outputs. If Independent format has been previously chosen, the *Output Data Format* will need to be setup here, that is, select Unipolar or Bipolar format and the bit structure. This area is not available and is grayed out if *Common format* has been chosen (see page 8-26).

WARNING: If the Upper Limit is set to zero, the output will never get above zero. In effect, there will be no control output.

Enter PID Parameters

Another PID setup dialog, Tuning, is for entering the PID parameters shown as: Gain (Proportional Gain), Reset (Integral Gain) and Rate (Derivative Gain).

Recall the position and velocity forms of the PID loop equations which were introduced earlier. The equations basically show the three components of the PID calculation: Proportional Gain (P), Integral Gain (I) and Derivative Gain (D). The following diagram shows a form of the PID calculation in which the control output is the sum of the proportional gain, integral gain and derivative gain. With each calculation of the loop, each term receives the same error signal value.



The P, I and D gains are 4-digit BCD numbers with values from 0000 to 9999. They contain an implied decimal point in the middle, so the values are actually 00.00 to 99.99. Some gain values have units – Proportional gain has no unit, Integral gain may be selected in seconds or in minutes, and Derivative gain is in seconds.

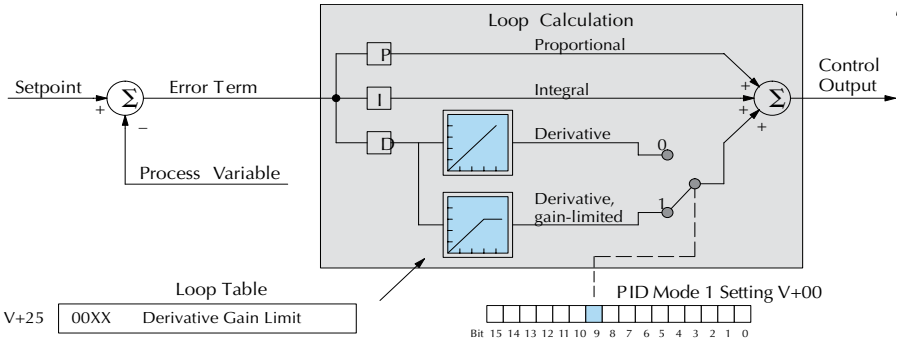
Gain (Proportional Gain) – This is the most basic gain of the three. Values range from 0000 to 9999, but they are used internally as xx.xx. An entry of "0000" effectively removes the proportional term from the PID equation. This accommodates applications which need integral-only loops.

Reset (Integral Gain) – Values range from 0001 to 9998, but they are used internally as xx.xx. An entry of "0000" or "9999" causes the integral gain to be "infinity", effectively removing the integrator term from the PID equation. This accommodates applications which need proportional-only loops. The units of integral gain may be either seconds or minutes, as shown in the above dialog.

Rate (Derivative Gain) – Values which can be entered range from 0001 to 9999, but they are used internally as XX.XX. An entry of "0000" allows removal of the derivative term from the PID equation (a common practice). This accommodates applications which require only proportional and/or integral loops. Most control loops will operate as a PI loop.

NOTE: *You may elect to leave the tuning dialog blank and enter the tuning parameters in the* ***Direct****SOFT PID View.*

Derivative Gain Limiting



The derivative gain (rate) has an optional gain-limiting feature. This is provided because the derivative gain reacts badly to PV signal noise or other causes of sudden PV fluctuations. The function of the gain-limiting is shown in the diagram below.

The gain limit can be particularly useful during loop tuning. Most loops can tolerate only a little derivative gain without going into uncontrolled oscillations.

If this option is checked, a *Limit* from 0 to 20 must also be entered.

NOTE*: When first configuring a loop, it's best to use the standard error term until after the loop is tuned. Once the loop is tuned, you will be able to tell if these functions will enhance control. The Error Squared and/or Enable Deadband can be selected later in the PID setup. Also, values are not required to be entered in the Tuning dialog, but they can set later in the* ***Direct****SOFT PID View.*

Error Term Selection

The error term is internal to the CPUs PID loop controller, and is generated again in each PID calculation. Although its data is not directly accessible, you can easily calculate it by subtracting: Error = (SP–PV). If the PV square-root extract is enabled, then: Error = √PV In any case, the size of the error and algebraic sign determine the next change of the control output for each PID calculation.

Error Squared – When selected, the squared error function simply squares the error term (but preserves the original algebraic sign), which is used in the calculation. This affects the Control Output by diminishing its response to smaller error values, but maintaining its response to larger errors. Some situations in which the error squared term might be useful:

- Noisy PV signal – using a squared error term can reduce the effect of low-frequency electrical noise on the PV, which will make the control system jittery. A squared error maintains the response to larger errors.
- Non-linear process – some processes (such as chemical pH control) require non-linear controllers for best results. Another application is surge tank control, where the Control Output signal must be smooth.

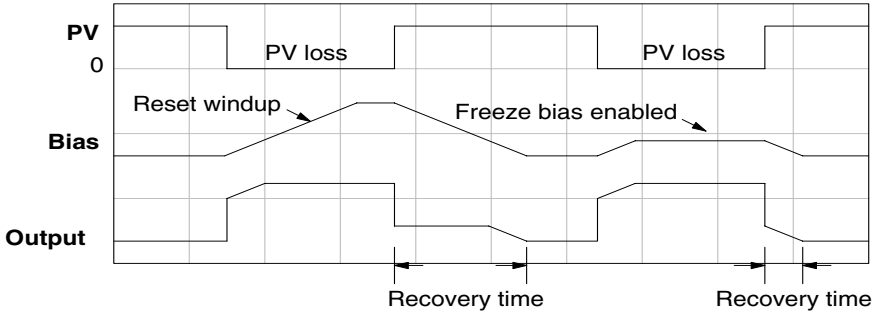
Enable Deadband – When selected, the enable deadband function takes a range of small error

values near zero, and simply substitutes zero as the value of the error. If the error is larger than the deadband range, then the error value is used normally.

Freeze Bias

The term *reset windup* refers to an undesirable characteristic of integrator behavior which occurs naturally under certain conditions. Refer to the figure below. Suppose the PV signal becomes disconnected, and the PV value goes to zero. While this is a serious loop fault, it is made worse by reset windup. Notice the bias (reset) term keeps integrating normally during the PV disconnect, until its upper limit is reached. When the PV signal returns, the bias value is saturated (windup) and takes a long time to return to normal. The loop output consequently has an extended recovery time. Until recovery, the output level is wrong and causes further problems.

In the second PV signal loss episode in the figure, the freeze bias feature is enabled. It causes



the bias value to freeze when the control output goes out of bounds. Much of the reset windup is thus avoided, and the output recovery time is much less.

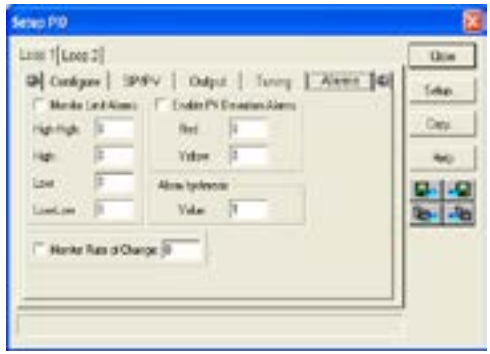
For most applications, the freeze bias feature will work with the loop as described above. It is suggested to enable this feature by selecting it in the dialog. Bit 10 of PID Mode 1 Setting (V+00) word can also be set in RLL.

NOTE: *The freeze bias feature stops the bias term from changing when the control output reaches the end of the data range. If you have set limits on the control output other than the range (i.e, 0–4095 for a unipolar/12-bit loop), the bias term still uses the end of range for the stopping point and bias freeze will not work.*

Setup the PID Alarms

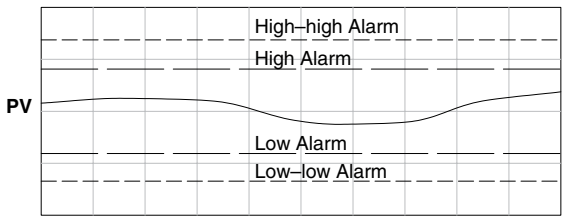
Although the setup of the PID alarms is optional, you surely would not want to operate a process without monitoring it. The performance of a process control loop may generally be measured by how closely the process variable matches the setpoint. Most process control loops in industry operate continuously, and will eventually lose control of the PV due to an error condition. Process alarms are vital in early discovery of a loop error condition and can alert plant personnel to manually control a loop or take other measures until the error condition has been repaired.

The alarm thresholds are fully programmable, and each type of alarm may be independently enabled and monitored. The following diagram shows the Alarm dialog in the PID setup which simplifies the alarm setup.



Monitor Limit Alarms

Checking this box will allow all of the PV limit alarms to be monitored once the limits are entered. The PV absolute value alarms are organized as two upper and two lower alarms. The alarm status is false as long as the PV value remains in the region between the upper and lower alarms, as shown below. The alarms nearest the safe zone are named *High Alarm* and *Low Alarm.* If the loop loses control, the PV will cross one of these thresholds first. Therefore, you can program the appropriate alarm threshold values in the loop table locations shown below to the right. The data format is the same as the PV and SP (12-bit or 15-bit). The threshold values for these alarms should be set to give an operator an early warning if the process loses control.



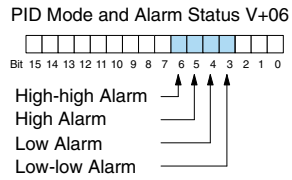
Loop Table

V+16	XXXX	High-high Alarm
V+15	XXXX	High Alarm
V+14	XXXX	Low Alarm
V+13	XXXX	Low-low Alarm

NOTE: *The Alarm dialog can be left as it first appears, without alarm entries. The alarms can then be setup in the* ***Direct****SOFT PID View.*

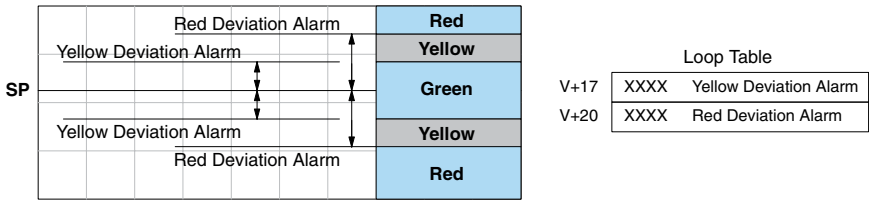
If the process remains out of control for some time, the PV will eventually cross one of the outer alarm thresholds, named High-high alarm and Low-low alarm. Their threshold values are programmed using the loop table registers listed above. A High-high or Low-low alarm indicates a serious condition exists, and needs the immediate attention of the operator.

The PV Absolute Value Alarms are reported in the four bits in t PID Mode and Alarm Status word in the loop table, as shown the right. We highly recommend using ladder logic to monit these bits. The bit-of-word instructions make this easy to d Additionally, you can monitor PID alarms using *Direct*SOFT.



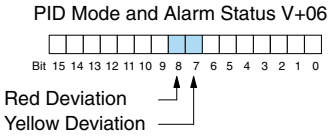
PV Deviation Alarms

The PV Deviation Alarms monitor the PV deviation with respect to the SP value. The deviation alarm has two programmable thresholds, and each threshold is applied equally above and below the current SP value. In the figure below, the smaller deviation alarm is called the **Yellow Deviation**, indicating a cautionary condition for the loop. The larger deviation alarm is called the **Red Deviation**, indicating a strong error condition for the loop. The threshold values use the loop parameter table locations V+17 and V+20 as shown.



The thresholds define zones, which fluctuate with the SP value. The green zone which surrounds the SP value represents a safe (no alarm) condition. The yellow zones lie outside the green zone, and the red zones are beyond those.

The PV Deviation Alarms are reported in the two bits in the PID Mode and Alarm Status word in the loop table, as shown to the right. We highly recommend using ladder logic to monitor these bits. The bit-of-word instructions make this easy to do. Additionally, you can monitor PID alarms using *Direct*SOFT.



The PV Deviation Alarm can be independently enabled and disabled from the other PV alarms, using bit 13 of the PID Mode 1 Setting V+00 word.

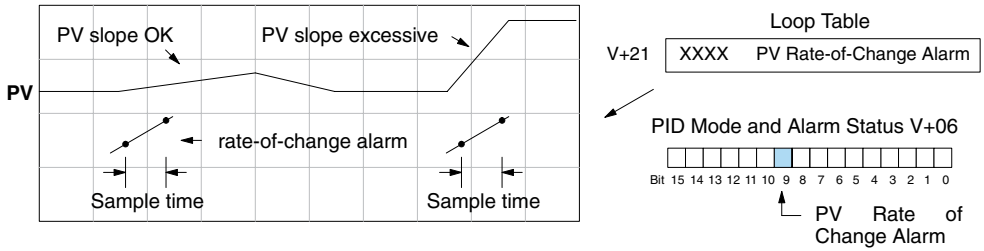
Remember the alarm hysteresis feature works in conjunction with both the deviation and absolute value alarms, and is discussed at the end of this section.

NOTE: *PID deviation alarm only work in Auto mode.*

PV Rate-of-Change Alarm

An excellent way to get an early warning of a process fault is to monitor the *rate-of-change* of the PV. Most batch processes have large masses and slowly-changing PV values. A relatively fast-changing PV will result from a broken signal wire for either the PV or control output, a SP value error, or other causes. If the operator responds to a PV Rate-of-Change Alarm quickly and effectively, the PV absolute value will not reach the point where the material in process would be ruined.

The DL06 loop controller provides a programmable PV Rate-of-Change Alarm, as shown below. The rate-of-change is specified in PV units change per loop sample time. This value is programmed into the loop table location V+21.



As an example, suppose the PV is the temperature for your process, and you want an alarm whenever the temperature changes faster than 15 degrees/minute. The PV counts per degree and the loop sample rate must be known. Then, suppose the PV value (in V+03 location) represents 10 counts per degree, and the loop sample rate is 2 seconds. Use the formula below to convert our engineering units to counts/sample period:

$$\text{Alarm Rate-of-Change} = \frac{15 \text{ degrees}}{1 \text{ minute}} \times \frac{10 \text{ counts / degree}}{30 \text{ loop samples / min.}} = \frac{150}{30} = 5 \text{ counts / sample period}$$

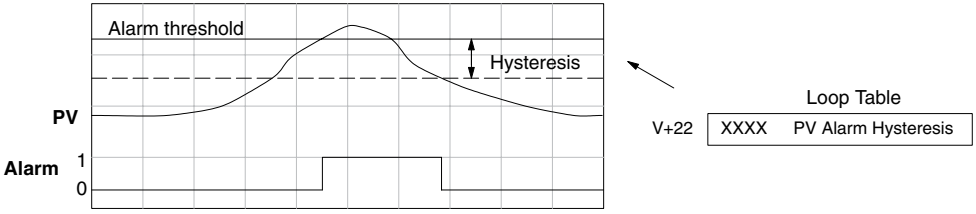
From the calculation result, you would program the value 5 in the loop table for the rate-of-change. The PV Rate-of-Change Alarm can be independently enabled and disabled from the other PV alarms, using bit 14 of the PID Mode 1 Setting V+00 word.

The alarm hysteresis feature (discussed next) does not affect the Rate-of-Change Alarm.

PV Alarm Hysteresis

The PV Absolute Value Alarm and PV Deviation Alarm are programmed using threshold values. When the absolute value or deviation exceeds the threshold, the alarm status becomes true. Real-world PV signals have some noise on them, which can cause some fluctuation in the PV value in the CPU. As the PV value crosses an alarm threshold, its fluctuations cause the alarm to be intermittent and annoy process operators. The solution is to use the PV Alarm Hysteresis feature.

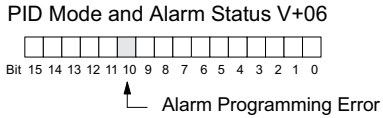
The PV Alarm Hysteresis amount is programmable from 1 to 200 (binary/decimal). When using the PV Deviation Alarm, the programmed hysteresis amount must be less than the programmed deviation amount. The figure below shows how the hysteresis is applied when the PV value goes past a threshold and descends back through it.



The hysteresis amount is applied after the threshold is crossed, and toward the safe zone. In this way, the alarm activates immediately above the programmed threshold value. It delays turning off until the PV value has returned through the threshold by the hysteresis amount.

Alarm Programming Error

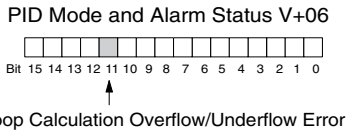
The PV Alarm threshold values must have certain mathematical relationships to be valid. The requirements are listed below. If not met, the Alarm Programming Error bit will be set, as indicated to the right.



- PV Absolute Alarm value requirements:
  Low-low < Low < High < High-high
- PV Deviation Alarm requirements:
  Yellow < Red

Loop Calculation Overflow/Underflow Error

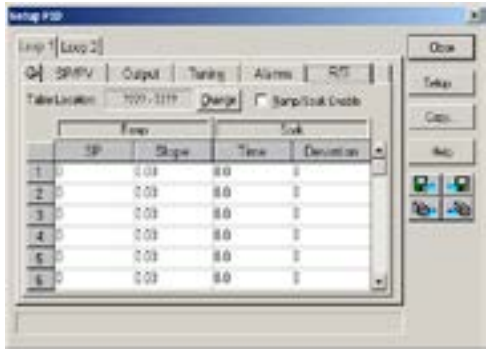
This error occurs whenever the output reaches its upper or lower limit and the PV does not reach the setpoint. A typical example might be when a valve is stuck, the output is at its limit, but the PV has not reached setpoint.



NOTE: *Overflow/underflow can be alarmed in PID View. The optional C-more operator interface panel (see the automationdirect.com website) can also be setup to read these error bits using the PID Faceplate templates.*

Ramp/Soak

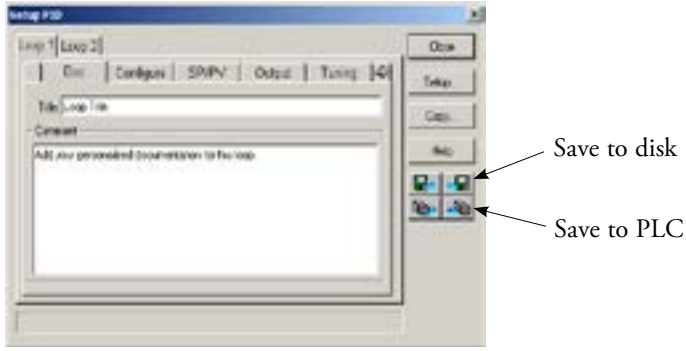
R/S (Ramp/Soak) is the last dialog available in the PID setup. The basic PID does not require any entries to be made in order to operate the PID loop. Ramp/Soak will be discussed in another section.



Complete the PID Setup

Once you have filled in the necessary information for the basic PID setup, the configuration should be saved. The icons on the Setup PID dialog will allow you to save the configuration to the PLC and to disk. The save to icons have the arrow pointing to the PLC and disk. The read from icons have the arrows pointing away from the PLC and disk.

An optional feature is available with the Doc tab in the Setup PID window. You enter a name and description for the loop. This is useful if there is more than one PID loop in your application.



NOTE: *It is good practice to save your project after setting up the PID loop by selecting* ***File*** *from the menu toolbar, then* ***Save project > to disk****. In addition to saving your entire project, all the PID parameters are also saved.*

PID Loop Tuning

Once you have set up a PID loop, it must be tuned in order for it to work. The goal of loop tuning is to adjust the loop gains so the loop has optimal performance in dynamic conditions. The quality of a loop's performance may generally be judged by how well the PV follows the SP after an SP step change. It is important to keep in mind that understanding the process is fundamental to getting a well designed control loop. Sensors must be in appropriate locations and valves must be sized correctly with appropriate trim. **PID control does not have *typical* values**. There isn't one control process that is identical to another.

Manual Tuning vs. Auto Tuning

You may enter the PID gain values to tune your loops (manual tuning), or you can rely on the PID processing engine in the CPU to automatically calculate the gain values (auto tuning). Most experienced process engineers will have a favorite method; the DL06 will accommodate either preference. The use of auto tuning can eliminate much of the trial-and-error of the manual tuning approach, especially if you do not have a lot of loop tuning experience. However, performing the auto tuning procedure will get the gains close to optimal values, but additional manual tuning can get the gain values to their optimal values.

WARNING: Only authorized personnel fully familiar with all aspects of the process should make changes that affect the loop tuning constants. Using the loop auto tune procedures will affect the process, including inducing large changes in the control output value. Make sure you thoroughly consider the impact of any changes to minimize the risk of injury to personnel or damage to equipment. The auto tune in the DL06 is not intended to be used as a replacement for your process knowledge.

Open-Loop Test

Whether you use manual or auto tuning, it is very important to verify basic characteristics of a newly-installed process before attempting to tune it. With the loop in Manual Mode, verify the following items for each new loop.

- **Setpoint** – verify that the SP source can generate a setpoint. Put the PLC in Run Mode and leave the loop in Manual Mode, then monitor the loop table location V+02 to see the SP value(s). (If you are using the ramp/soak generator, test it now).
- **Process Variable** – verify that the PV value is an accurate measurement, and the PV data arriving in the loop table location V+03 is correct. If the PV signal is very noisy, consider filtering the input either through hardware (RC low-pass filter), or using the filter in this chapter.
- **Control Output** – if it is safe to do so, manually change the output a small amount (perhaps 10%) and observe its affect on the process variable. Verify the process is direct-acting or reverse acting, and check the setting for the control output (inverted or non-inverted). Make sure the control output upper and lower limits are not equal to each other.
- **Sample Rate** – while operating open-loop, this is a good time to find the ideal sample rate (see Configure the PID Loop on page 8-25). However, if you are going to use auto tuning, the auto tuning procedure will automatically calculate the sample rate in addition to the PID gains.

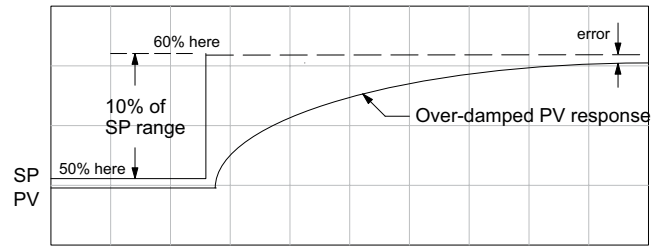
Manual Tuning Procedure

It is not necessary to try to obtain the best values for the P, I and D parameters in the PID loop by trial and error. Following is a typical procedure for tuning a temperature control loop which you may use to tune your loop.

Monitor the values of SP, PV and CV with a loop trending instrument or use the PID View feature in *Direct*SOFT (see page 8-49).

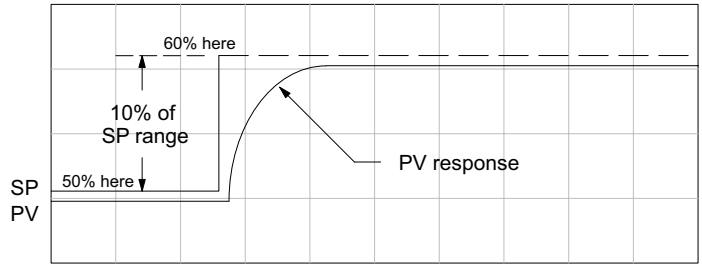
NOTE: *We recommend using the PID View Tuning and Trending window to select manual for the vertical scale feature, for both SP/PV area and Bias/Control Output areas. The auto scaling feature would otherwise change the vertical scale on the process parameters and add confusion to the loop tuning process.*

- Adjust the gains so the Proportional Gain = 0.5 or 1.0 (1.0 is a good value based on experience), Integral Gain = 9999 (this basically eliminates reset) and Derivative Gain = 0000. This disables the integrator and derivative terms, and provides some proportional gain.
- Check the bias value in the PID View and set it to zero.
- Set the SP to a value equal to 50% of the full range.
- Now, select Auto Mode. If the loop will not stay in Auto Mode, check the troubleshooting tips at the end of this chapter. Allow the PV to stabilize around the 50% point of the range.

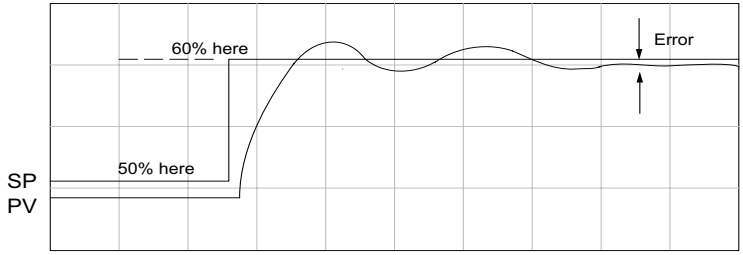


- Change the SP to the 60% point of the range.

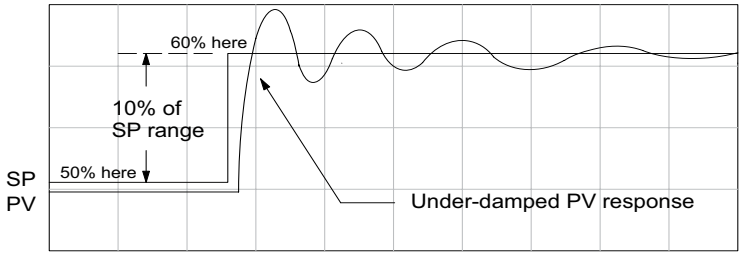
The response may take awhile, but you will see that there isn't any oscillation. This response is not desirable since it takes a long time to correct the error; also, there is a difference between the SP and the PV.



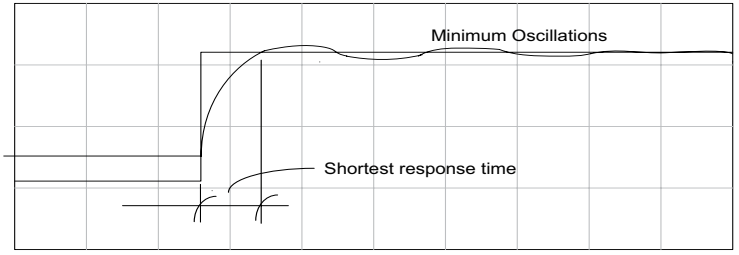
- Increase the Proportional gain, for example to 2.0. The control output will be greater and the response time will be quicker. The trend should resemble the figure below.
- Increase the Proportional gain in small increments, such as 4, 6, 7, etc., until the control output response begins to oscillate. This is the Proportional gain that should be recorded.



- Now, return the Proportional gain to the stable response; for example, 9.7. The error, SP-PV, should be small, but not at zero.
- Next, add a small amount of Integral gain (reset) in order for the error to reach zero. Begin by using 80 seconds (adjust in minutes if necessary). The error should get smaller.
- Set the Integral gain to a lower value, such as 50 for a different response. If there is no response, continue to decrease the reset value until the response becomes unstable. See the figure below.



- For discussion, let us say that a reset value of 35 made the control output unstable. Return the reset value to the stable value, such as 38. **Be careful with this adjustment since the oscillation can destroy the process.**
- The control output response should be optimal now, without a Derivative gain. The example recorded values are: Proportional gain = 9.7 and Integral gain = 38 seconds. Note that the error has been minimized.



- The foregone method is the most common method used to tune a PID loop. Derivative gain is almost never used in a temperature control loop. This method can also be used for other control loops, but other parameters may need to be added for a stable control output.

Test your loop for a high PV of 80% and again for a low PV of 20%, and correct the values if necessary. Small adjustments of the parameters can make the control output more precise or more unstable. It is sometimes acceptable to have a small overshoot to make the control output react quicker.

The derivative gain can be helpful for those control loops which are not controlling temperature. For these loops, try adding a value of 0.5 for the derivative gain and see if this improves the control output. If there is little or no response, increase the derivative by increments of 0.5 until there is an improvement to the output trend. Recall that the derivative gain reacts with a rate of change of the error.

Alternative Manual Tuning Procedures by Others

The following tuning procedures have been extracted from various publications about PID process control. These procedures are for comparison to the procedure in this manual.

Tuning PID Controllers

Two-Mode Simple Method - – for P-I controllers

1. Turn off reset and set the gain to a small value (0.5 - 1.0).
2. Increase gain until cycling starts, then decrease gain slightly.
3. Make setpoint changes to observe offset (error).
4. Increase reset to eliminate offset (error).
5. Repeat steps 2 through 4 until you obtain the largest gain and reset consistent with the criteria of the control desired, i.e., offset, overshoot, stability.

Zeigler-Nichols Method– "Quarter amplitude decay"

1. Turn off reset and rate; set the proportional gain to a fairly large value.
2. Make a small setpoint change and observe how the controlled variable cycles.
3. Adjust the gain until the cycle is self-sustaining, and of constant amplitude; this value is the ultimate gain (Gu).
4. Measure the period of cycling in minutes. This is the ultimate period (Pu).
5. Calculate the controller adjustments as follows:

P only:	G = Gu/2	
P & I :	G = Gu/2.2	
	Ti = 1.2/Pu	(repeats/minute)
P-I-D:	G = Gu/1.6	
	Ti = 2.0/Pu	(repeats/minute)
	Td = Pu/8.0	(minutes)

Pessen Method

1. Follow the procedure described above (Zeigler-Nichols) to determine the ultimate gain and ultimate period.
2. Apply the formulas below.

   - For no overshoot during startup:

     G = Gu/5.0
     Ti = 3/Pu (repeats/minute)
     Td = Pu/2 (minutes)

   - For some overshoot, but better response to disturbances:

     G = Gu/3
     Ti = 3/Pu (repeats/minute)
     Td = Pu/3 (minutes)

Auto Tuning Procedure

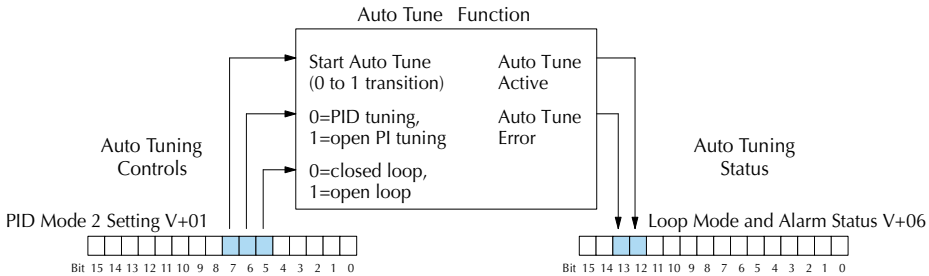
The auto tuning feature for the DL06 loop controller will only run once each time it is enabled in the PID table. Therefore, auto tuning does not run continuously during operation (this would be *adaptive* control). Whenever there is a substantial change in loop dynamics, such as mass of process, size of actuator, etc., the tuning process will need to be repeated in order to derive new gains required for optimal control.

WARNING: Only authorized personnel fully familiar with all aspects of the process should make changes that affect the loop tuning constants. Using the loop auto tuning procedures will affect the process, including inducing large changes in the control output value. Make sure you thoroughly consider the impact of any changes to minimize the risk of injury to personnel or damage to equipment. The auto tune in the DL06 is not intended to be used as a replacement for your process knowledge.

Once the physical loop components are connected to the PLC, auto tuning can be initiated within ***Direct***SOFT (see the ***Direct***SOFT Programming Software Manual), and it can be used to establish initial PID parameter values. Auto tuning is the best "guess" the CPU can do after some trial tests.

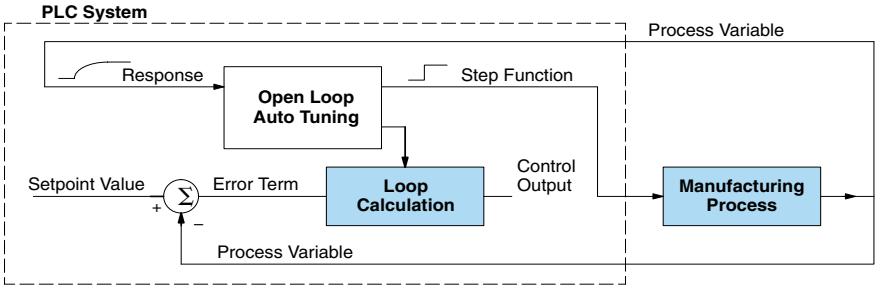
The loop controller offers both closed-loop and open-loop methods. The following sections describe how to use the auto tuning feature, and what occurs in open and closed-loop auto tuning.

The controls for the auto tuning function use three bits in the PID Mode 2 word V+01, as shown below. ***Direct***SOFT will manipulate these bits automatically when you use the auto tune feature within ***Direct***SOFT. Or, you may have your ladder logic access these bits directly for allowing control from another source such as a dedicated operator interface. The individual control bits allow you to start the auto tune procedure, select PID or PI tuning and select closed-loop or open-loop tuning. If you select PI tuning, the auto tune procedure leaves the derivative gain at 0. The Loop Mode and Alarm Status word V+06 reports the auto tune status as shown. Bit 12 will be on (1) during the auto tune cycle, automatically returning to off (0) when done.



Open-Loop Auto Tuning

During an open-loop auto tuning cycle, the loop controller operates as shown in the diagram below. Before starting this procedure, place the loop in Manual Mode and ensure the PV and control output values are in the middle of their ranges (away from the end points).

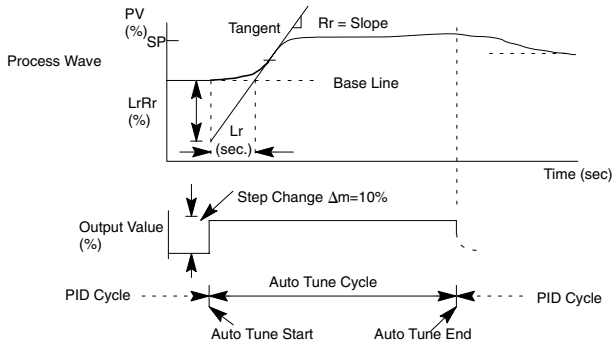


***NOTE**: In theory, the SP value does not matter in this case, because the loop is not closed. However, the requirement of the firmware is that the SP value must be more than 5% of the PV range from the actual PV before starting the auto tune cycle (for the DL06, 12 bit PV should be 205 counts or more below the SP for forward-acting loops, or 205 counts or more above the SP for reverse-acting loops).*

When auto tuning, the loop controller induces a step change on the output and simply observes the response of the PV. From the PV response, the auto tune function calculates the gains and the sample time. It automatically places the results in the corresponding registers in the loop table.

The following timing diagram shows the events which occur in the open-loop auto tuning cycle. The auto tune function takes control of the control output and induces a 10%-of-span step change. If the PV change which the loop controller observes is less than 2%, then the step change on the output is increased to 20%-of-span.

- When Auto Tune starts, step change output m=10%
- During Auto Tune, the controller output reached the full scale positive limit. Auto Tune stopped and the Auto Tune Error bit in the Alarm word bit turned on.
- When PV change is under 2%, output is changed at 20%. Open Loop Auto Tune Cycle Wave: Step Response Method



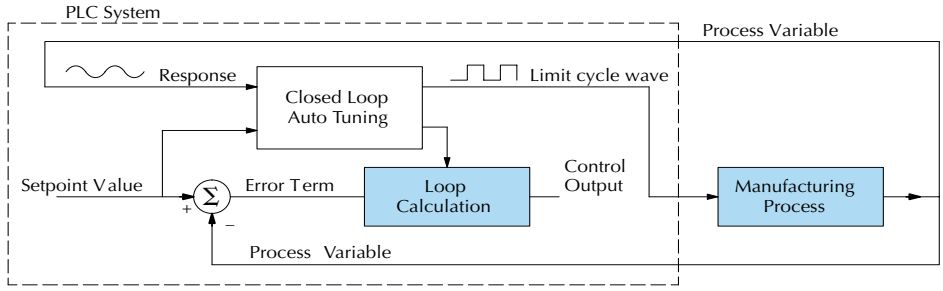
When the loop tuning observations are complete, the loop controller computes Rr (maximum slope in %/sec.) and Lr (dead time in sec). The auto tune function computes the gains according to the Zeigler-Nichols equations, shown below:

PID Tuning	PI Tuning
P=1.2*Δm/LrRr	P=0.9*Δm/LrRr
I=2.0* Lr	I=3.33* Lr
D=0.5* Lr	D=0
Sample Rate = 0.056* Lr	Sample Rate = 0.12*Lr
Δm = Output step change (10% = 0.1, 20% = 0.2)	

We highly recommend using *Direct*SOFT for the auto tuning interface. The duration of each auto tuning cycle will depend on the mass of the process. A slowly-changing PV will result in a longer auto tune cycle time. When the auto tuning is complete, the proportional, integral, and derivative gain values are automatically updated in loop table locations V+10, V+11, and V+12 respectively. The sample time in V+07 is also updated automatically. You can test the validity of the values the auto tuning procedure yields by measuring the closed-loop response of the PV to a step change in the output. The instructions on how to do this are in the section on the manual tuning procedure (located prior to this auto tuning section).

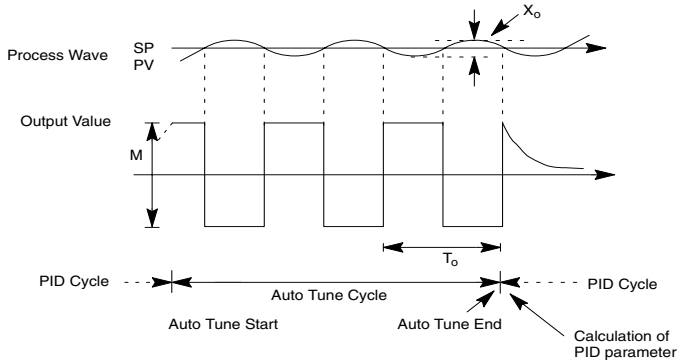
Closed-Loop Auto Tuning

During a closed-loop auto tuning cycle the loop controller operates as shown in the diagram below.



When auto tuning, the loop controller imposes a square wave on the output. Each transition of the output occurs when the PV value crosses over/under the SP value. Therefore, the frequency of the limit cycle is roughly proportional to the mass of the process. From the PV response, the auto tune function calculates the gains and the sample time. It automatically places the results in the corresponding registers in the loop table.

The following timing diagram shows the events which occur in the closed-loop auto tuning cycle. The auto tune function examines the direction of the offset of the PV from the SP. The auto tune function then takes control of the control output and induces a full-span step change in the opposite direction. Each time the sign of the error (SP – PV) changes, the output changes full-span in the opposite direction. This proceeds through three full cycles.



*Mmax = Output Value upper limit setting.

*Mmin = Output Value lower limit setting.

* This example is direct–acting.

When set to reverse–acting, the output will be inverted. When the loop tuning observations are complete, the loop controller computes To (bump period) and Xo (amplitude of the PV). Then it uses these values to compute Kpc (sensitive limit) and Tpc (period limit). From these values, the loop controller auto tune function computes the PID gains and the sample rate according to the Zeigler-Nichols equations shown below:

Kpc = 4M / (π *X0) Tpc = 0	
M = Amplitude of output	
PID Tuning	**PI Tuning**
P = 0.45*Kpc	P = 0.30*Kpc
I = 0.60*Tpc	I = 1.00*Tpc
D = 0.10*Tpc	D = 0
Sample Rate = 0.014*Tpc	Sample Rate = 0.03*Tpc

Auto tuning error

In open-loop tuning, if the auto tune error bit (bit 13 of loop Mode/Alarm status word V+06) is on, please verify the PV and SP values are within 5% of full scale difference, as required by the auto tune function.

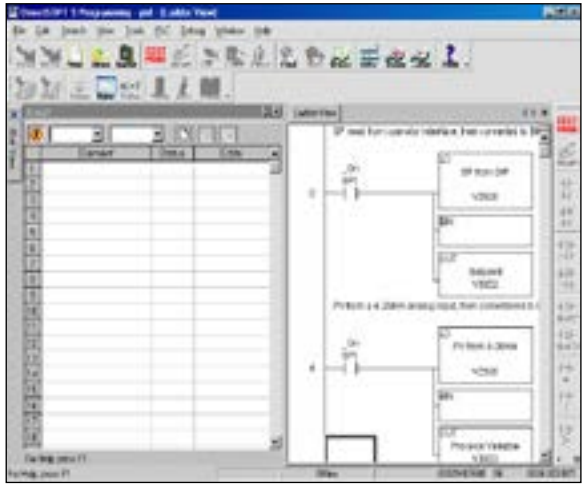
***NOTE**: If your PV fluctuates rapidly, you probably need to use the built-in analog filter (see page 8–55) or create a filter in ladder logic (see example on page 8–56).*

Use DirectSOFT Data View with PID View

The **Data View** window is a very useful tool which can be used to help tune your PID loop. You can compare the variables in the **PID View** with the actual values in the V-memory location with Data View.

Open a New Data View Window

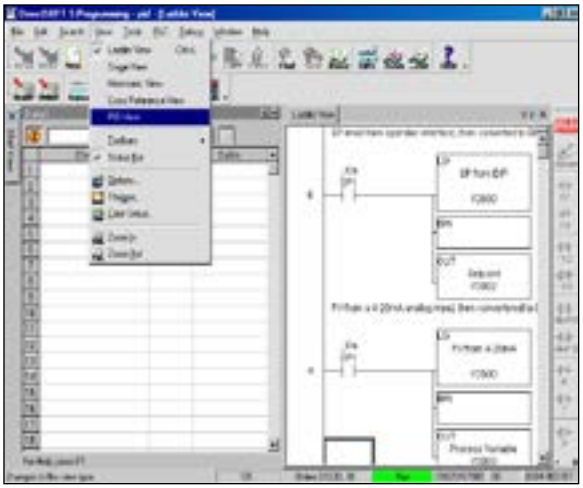
A new Data View window can be opened in any one of three ways; the menu bar **Debug > Data View > New**, the keyboard shortcut **Ctrl + Shift + F3** or the **Data** button on the Status toolbar. By default, the Data View window is assigned Data1 as the default name. This name can be changed for the current view using the Options dialog. The following diagram is an example of a newly opened Data View. The window will open next to the Ladder View by default.



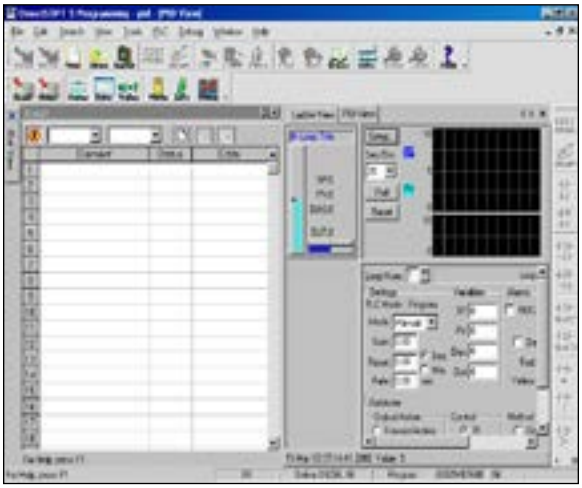
The Data View window can be used just as it is shown above for troubleshooting your PID logic, and it can be most useful when tuning the PID loop.

Open PID View

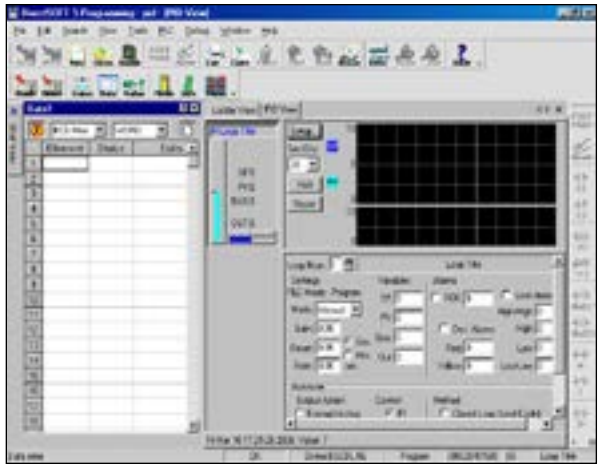
The PID View can only be opened after a loop has been setup in your ladder program. PID View is opened by selecting it from the View submenu on the Menu bar, **View > PID View**. The PID View can also be opened by clicking on the PID View button from the PLC Setup toolbar if it is in view.



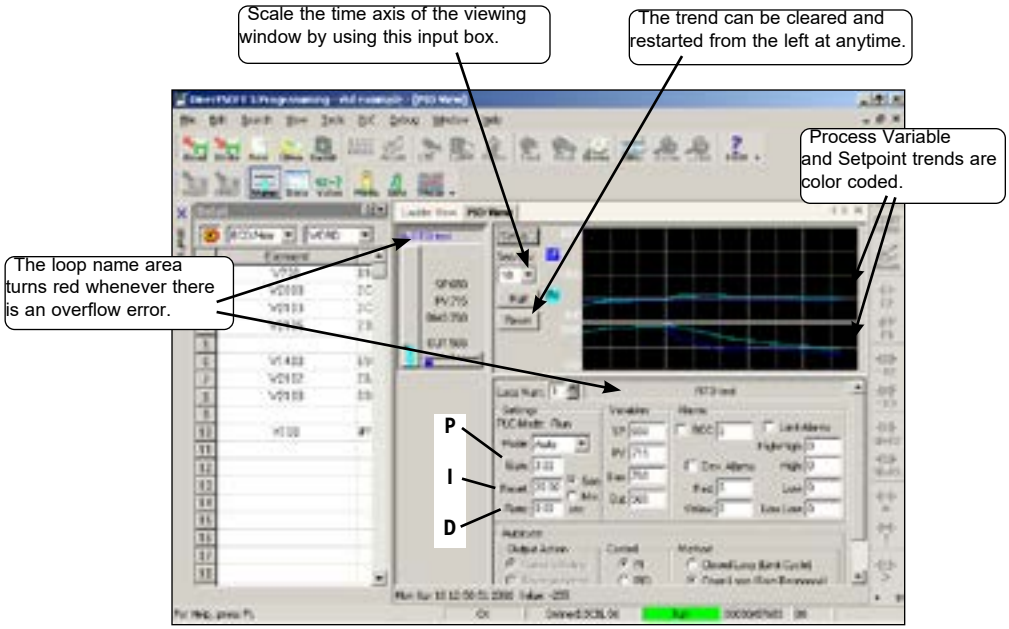
The PID View will open and appear over the Ladder View which can be brought into view by clicking on its tab. When using the Data View and the PID View together, each view can be sized for better use as shown on the facing page.



The two views are now ready to be used to tune your loop. You will be able to see where the PID values have been set and see the process that it is controlling.



The diagram below illustrates how the to use the views to see the current SP, PV and Output values, along with the other PID addresses. Refer to the Loop Table Definitions page 8-20 for details of each word in the table. This is also a good data type reference for each word in the table.

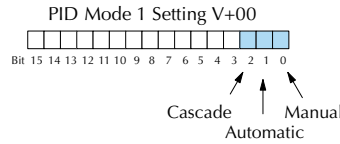


With both windows positioned in this manner, you are able to see where the PID values have been set and see the process that it is controlling. In the diagram below, you can see the current SP, PV and Output values, along with the other PID addresses. Refer to the Loop Table Word Definitions (page 8-20) for details for each word in the table. This is also a good data type reference for each word in the table.

Using the Special PID Features

It's a good idea to understand the special features of the DL06 and how to use them. You may want to incorporate some of these features for your PID.

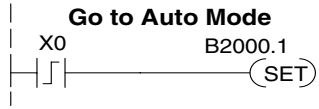
How to Change Loop Modes



The first three bits of the PID Mode 1 word (V+00) request the operating mode of the corresponding loop. Note: these bits are mode change *requests*, not commands (certain conditions can prohibit a particular mode change – see next page).

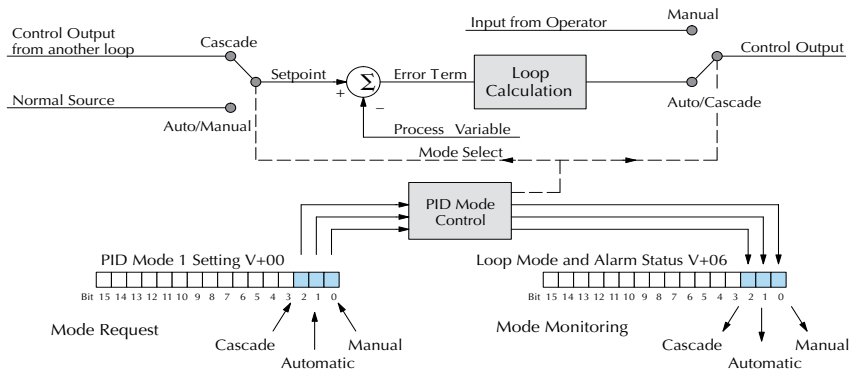
The normal state of these mode request bits is "000." To request a mode change, you must SET the corresponding bit to a "**1**" using a one-shot. The PID loop controller automatically resets the bits back to "000" after it reads the mode change request. Methods of requesting mode changes are:

- ***Direct***SOFT's PID View – this is the easiest method. Use the pull-down menu, or click on one of the radio buttons if using older ***Direct***SOFT versions, and the appropriate bit will get set.
- Ladder program– ladder logic can request any loop mode when the PLC is in Run Mode. This will be necessary after application startup if mode changes are part of the application.
- Use the program shown to the right to SET the mode bit (do not use an OUT coil). On a 0–1 transition of X0, the rung sets the Auto bit equal to 1. The loop controller resets it.



- Operator panel – interface the operator's panel to ladder logic using standard methods, then use the logic to the right to set the mode bit.

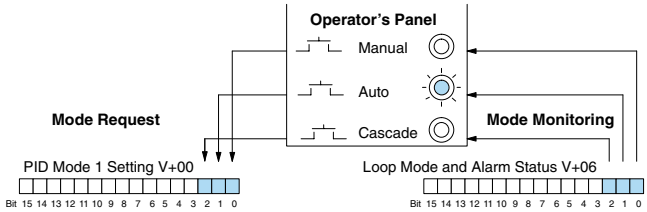
Since mode changes can only be *requested*, the PID loop controller will decide when to permit mode changes and provide the loop mode status. It reports the current mode on bits 0, 1, and 2 of the Loop Mode/Alarm Status word, location V+06 in the loop table. The parallel request/monitoring functions are shown in the figure below. The figure also shows the two possible mode-dependent SP sources, and the two possible Control Output sources.



Operator Panel Control of PID Modes

Since the modes Manual, Auto and Cascade are the most fundamental and important PID loop controls, you may want to hard-wire mode control switches to an operator's panel. Most applications will need only Manual and Auto selections (Cascade is used in special applications). Remember that mode controls are really *mode request* bits, and the actual loop mode is indicated elsewhere.

The figure at right shows an operator's panel using momentary push-buttons to request PID mode changes. The panel's mode indicators do not connect to the switches, but interface to the corresponding data locations.



PLC Modes Effect on Loop Modes

If you have selected the option for the loops to follow the PLC mode, the PLC modes (Program, Run) interact with the loops as a group. The following summarizes this interaction:

- When the PLC is in Program Mode, all loops are placed in Manual Mode and no loop calculations occur. However, note that output modules (including analog outputs) turn off in PLC Program Mode. So, actual manual control is not possible when the PLC is in Program Mode.
- The only time the CPU will allow a loop mode change is during PLC Run Mode operation. As such, the CPU records the modes of all 4 loops as the desired mode of operation. If power failure and restoration occurs during PLC Run Mode, the CPU returns all loops to their prior mode (which could be Manual, Auto, or Cascade).
- On a Program-to-Run mode transition, the CPU forces each loop to return to its prior mode recorded during the last PLC Run Mode.
- You can add and configure new loops only when the PLC is in Program Mode. New loops automatically begin in Manual Mode.

Loop Mode Override

In normal conditions the mode of a loop is determined by the request to V+00, bits 0, 1, and 2. However, some conditions exist which will prevent a requested mode change from occurring:

- A loop that is not set independent of PLC mode cannot change modes when the PLC is in Program mode.
- A major loop of a cascaded pair of loops cannot go from Manual to Auto until its minor loop is in Cascade mode.

In other situations, the PID loop controller will automatically change the mode of the loop to ensure safe operation:

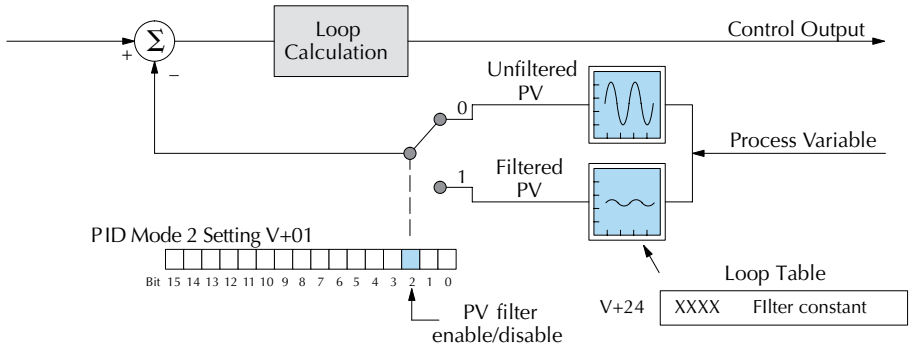
- A loop which develops an error condition automatically goes to Manual.
- If the minor loop of a cascaded pair of loops leaves Cascade Mode for any reason, its major loop automatically goes to Manual Mode.

PV Analog Filter

A noisy PV signal can make tuning difficult and can cause the control output to be more extreme than necessary, as the output tries to respond to the peaks and valleys of the PV. There are two equivalent methods of filtering the PV input to make the loop more stable. The first method is accomplished using the DL06's built-in filter. The second method achieves a similar result using ladder logic.

The DL06 Built-in Analog Filter

The DL06 provides a selectable first-order low-pass PV input filter. **We only recommend the use of a filter during auto tuning or PID control if there is noise on the input signal.** You may disable the filter after auto tuning is complete, or continue to use it if the PV input signal is noisy.



Bit 2 of PID Mode Setting 2 provides the enable/disable control for the low-pass PV filter (0=disable, 1=enable). The roll-off frequency of the single-pole low-pass filter is controlled by using register V+24 in the loop parameter table, the filter constant. The data format of the filter constant value is BCD, with an implied decimal point 00X.X, as follows:

- The filter constant has a valid range of 000.1 to 001.0. The smaller the filter value, the greater the filtering performed; for example, the value 001.0 provides no filtering.
- DirectSOFT converts values above the valid range to 001.0 and values below this range to 000.1
- Values close to 001.0 result in higher roll-off frequencies, while values closer to 000.1 result in lower roll-off frequencies.

We highly recommend using ***Direct***SOFT for the auto tuning interface. The duration of each auto tuning cycle will depend on the mass of your process. A slowly-changing PV will result in a longer auto tune cycle time.

When the auto tuning is complete, the proportional and integral gain values are automatically updated in loop table locations V+10 and V+11 respectively. The derivative is calculated if you autotune for PID and updated in loop table location V+12. The sample time in V+07 is also updated automatically. You can test the validity of the values the auto tuning procedure yields by measuring the closed-loop response of the PV to a step change in the output. The instructions on how to do this are in the section on the manual tuning procedure.

The algorithm which the built-in filter follows is:

$$yi = k\,(xi - yi{-}1) + yi{-}1$$

yi is the current output of the filter

xi is the current input to the filter

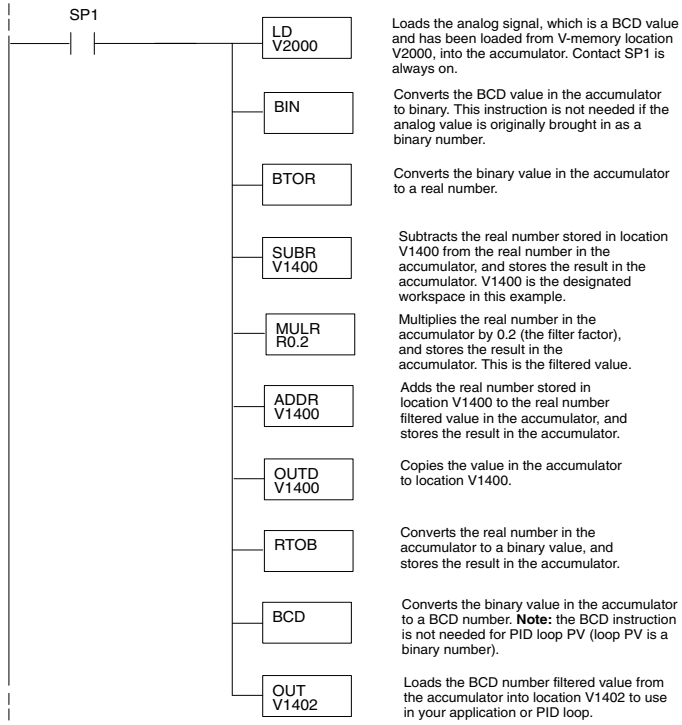
yi–1 is the previous output of the filter

k is the PV Analog Input Filter Factor

Creating an Analog Filter in Ladder Logic

A similar algorithm can be built in your ladder program. Your analog inputs can be filtered effectively using either method. The following programming example describes the ladder logic you will need. Be sure to change the example memory locations to those that fit your application.

Filtering can induce a 1 part in 1000 error in your output because of "rounding." If your process cannot tolerate a 1 part in 1000 error, do not use filtering. Because of the rounding error, you should not use zero or full scale as alarm points. Additionally, the smaller the filter constant the greater the smoothing effect, but the slower the response time. Be sure a slower response is acceptable in controlling your process.



Use the DirectSOFT 5 Filter Intelligent Box Instruction

For those who are using *Direct*SOFT 5, you have the opportunity to use Intelligent Box (IBox) instruction IB-402, Filter Over Time in Binary (decimal). This IBox will perform a first-order filter on the Raw Data on a defined time interval. The equation is,

New = Old + [(Raw - Old) / FDC] where

- New =New Filtered Value
- Old = Old Filtered Value
- FDC = Filter Divisor Constant
- Raw = Raw Data

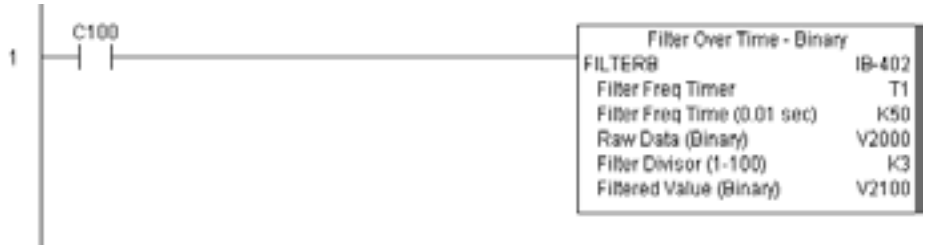
The Filter Divisor Constant is an integer in the range K1 to K100, such that if it equaled K1, then no filtering is performed.

The rate at which the calculation is performed is specified by time in hundredths of a second (0.01 seconds) as the Filter Freq Time parameter. Note that this Timer instruction is embedded in the IBox and must NOT be used any other place in your program. Power flow controls whether the calculation is enabled. If it is disabled, the Filter Value is not updated. On the first scan from Program to Run mode, the Filter Value is initialized to 0 to give the calculation a consistent starting point.

Since the following binary filter example does not write directly to the PID PV location, the BCD filter could be used with BCD values and then converted to BIN.

FilterB Example

Following is an example of how the FilterB IBox is used in a ladder program. The instruction is used to filter a binary value that is in V2000. Timer (T1) is set to 0.5 seconds, the rate at which the filter calculation will be performed. The filter constant is set to 3.0. A larger value will increase the smoothing effect of the filter. A value of 1 results with no filtering. The filtered value will be placed in V2100.

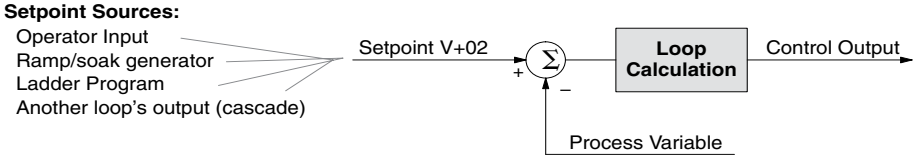


NOTE: *See Chapter 5, page 242, for more detailed information.*

Ramp/Soak Generator

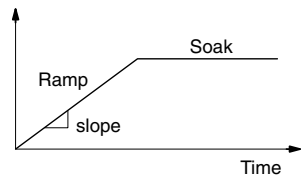
Introduction

Our discussion of basic loop operation noted the setpoint for a loop will be generated in various ways, depending on the loop operating mode and programming preferences. In the figure below, the ramp/soak generator is one of the ways the SP may be generated. It is the responsibility of your ladder program to ensure only one source attempts to write the SP value at V+02 at any particular time.



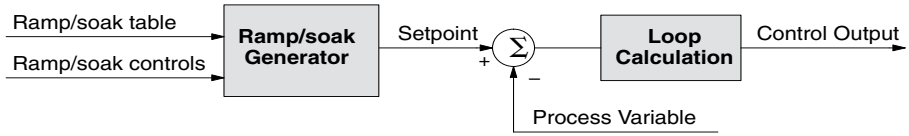
If the SP for your process rarely changes or can tolerate step changes, you probably will not need to use the ramp/soak generator. However, some processes require precisely-controlled SP value changes. *The ramp/soak generator can greatly reduce the amount of programming required for these applications.*

The terms **ramp** and **soak** have special meanings in the process control industry, and refer to desired setpoint (SP) values in temperature control applications. In the figure to the right, the setpoint increases during the ramp segment. It remains steady at one value during the soak segment.



Complex SP profiles can be generated by specifying a series of ramp/soak segments. The ramp segments are specified in SP units per second time. The soak time is also programmable in minutes.

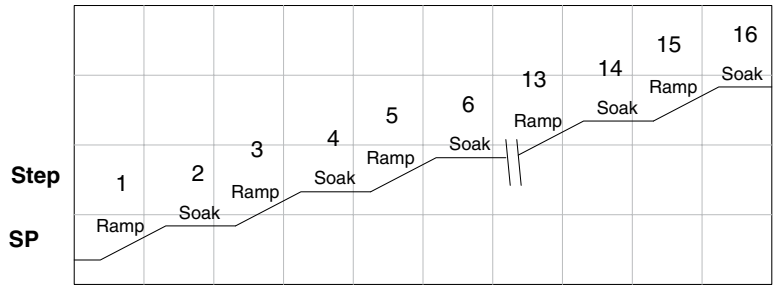
It is instructive to view the ramp/soak generator as a dedicated function to generate SP values, as shown below. It has two categories of inputs which determine the SP values generated. The ramp/soak table must be programmed in advance, containing the values that will define the ramp/soak profile. The loop reads from the table during each PID calculation as necessary. The ramp/soak controls are bits in a special loop table word that control the real-time start/stop functionality of the ramp/soak generator. The ladder program can monitor the status of the ramp soak profile (current ramp/segment number).



Now that we have described the general ramp/soak generator operation, we list its specific features:

- Each loop has its own ramp/soak generator (use is optional).
- You may specify up to eight ramp/soak steps (16 segments).
- The ramp soak generator can run any time the PLC is in Run mode. Its operation is independent of the loop mode (Manual or Auto).
- Ramp/soak real-time controls include Start, Hold, Resume, and Jog.
- Ramp/soak monitoring includes Profile Complete, Soak Deviation (SP minus PV), and current ramp/soak step number.

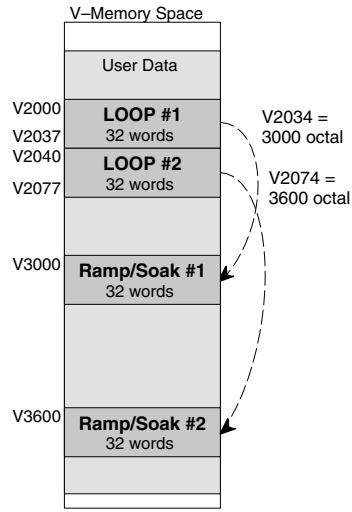
The following figure shows an SP profile consisting of ramp/soak segment pairs. The segments are individually numbered as steps from 1 to 16. The slope of each of the ramp segments may be either increasing or decreasing. The ramp/soak generator automatically knows whether to increase or decrease the SP based on the relative values of a ramp's end points. These values come from the ramp/soak table.



Ramp/Soak Table

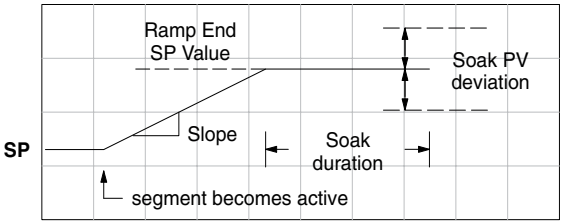
The parameters which define the ramp/soak profile for a loop are in a ramp/soak table. Each loop may have its own ramp/soak table, but it is optional. Recall the Loop Parameter table consists of a 32-word block of memory for each loop, and together they occupy one contiguous memory area. However, the ramp/soak table for a loop is individually located, because it is optional for each loop. An address pointer in location V+34 in the loop table specifies the starting location of the ramp/soak table.

In the example to the right, the loop parameter tables for Loop #1 and #2 occupy contiguous 32-word blocks as shown. Each has a pointer to its ramp/soak table, independently located elsewhere in user V-memory. Of course, you may locate all the tables in one group, as long as they do not overlap.



The parameters in the ramp/soak table must be user-defined. the most convenient way is to use *Direct*SOFT, which features a special editor for this table. Four parameters are required to define a ramp and soak segment pair, as pictured below.

- **Ramp End Value** – specifies the destination SP value for the end of the ramp. Use the same data format for this number as you use for the SP. It may be above or below the beginning SP value, so the slope could be up or down (we don't have to know the starting SP value for ramp #1).
- **Ramp Slope** – specifies the SP increase in counts (units) per second. It is a BCD number from 00.00 to 99.99 (uses implied decimal point).
- **Soak Duration** – specifies the time for the soak segment in minutes, ranging from 000.1 to 999.9 minutes in BCD (implied decimal point).
- **Soak PV Deviation** – (optional) specifies an allowable PV deviation above and below the SP value during the soak period. A PV deviation alarm status bit is generated by the ramp/soak generator.



Ramp/Soak Table

V+00	XXXX	Ramp End SP Value
V+01	XXXX	Ramp Slope
V+02	XXXX	Soak Duration
V+03	XXXX	Soak PV Deviation

The ramp segment becomes active when the previous soak segment ends. If the ramp is the first segment, it becomes active when the ramp/soak generator is started, and automatically assumes the present SP as the starting SP.

Offset	Step	Description	Offset	Step	Description
+ 00	1	Ramp End SP Value	+ 20	9	Ramp End SP Value
+ 01	1	Ramp Slope	+ 21	9	Ramp Slope
+ 02	2	Soak Duration	+ 22	10	Soak Duration
+ 03	2	Soak PV Deviation	+ 23	10	Soak PV Deviation
+ 04	3	Ramp End SP Value	+ 24	11	Ramp End SP Value
+ 05	3	Ramp Slope	+ 25	11	Ramp Slope
+ 06	4	Soak Duration	+ 26	12	Soak Duration
+ 07	4	Soak PV Deviation	+ 27	12	Soak PV Deviation
+ 10	5	Ramp End SP Value	+ 30	13	Ramp End SP Value
+ 11	5	Ramp Slope	+ 31	13	Ramp Slope
+ 12	6	Soak Duration	+ 32	14	Soak Duration
+ 13	6	Soak PV Deviation	+ 33	14	Soak PV Deviation
+ 14	7	Ramp End SP Value	+ 34	15	Ramp End SP Value
+ 15	7	Ramp Slope	+ 35	15	Ramp Slope
+ 16	8	Soak Duration	+ 36	16	Soak Duration
+ 17	8	Soak PV Deviation	+ 37	16	Soak PV Deviation

Many applications do not require all 16 R/S steps. Use all zeros in the table for unused steps. The R/S generator ends the profile when it finds ramp slope = 0.

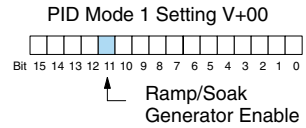
Ramp/Soak Table Flags

The individual bit definitions of the Ramp/Soak Table Flag (Addr+33) word is listed in the following table.

Bit	Ramp/Soak Flag Bit Description	Read/Write	Bit=0	Bit=1
0	Start Ramp / Soak Profile	Write	–	01 Start
1	Hold Ramp / Soak Profile	Write	–	01 Hold
2	Resume Ramp / Soak Profile	Write	–	01 Resume
3	Jog Ramp / Soak Profile	Write	–	01 Jog
4	Ramp / Soak Profile Complete	Read	–	Complete
5	PV Input Ramp / Soak Deviation	Read	Off	On
6	Ramp / Soak Profile in Hold	Read	Off	On
7	Reserved	Read	Off	On
8–15	Current Step in R/S Profile	Read	Decode as byte (hex)	

Ramp/Soak Generator Enable

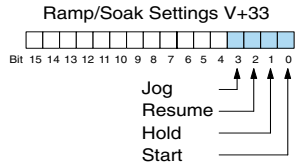
The main enable control to permit ramp/soak generation of the SP value is accomplished with bit 11 in the PID Mode 1 Setting V+00 word, as shown to the right. The other ramp/soak controls in V+33 shown in the table above will not operate unless this bit=1 during the entire ramp/soak process.



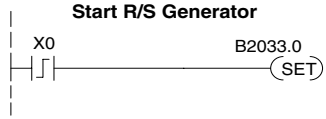
Ramp/Soak Controls

The four main controls for the ramp/soak generator are in bits 0 to 3 of the ramp/soak settings word in the loop parameter table. *Direct*SOFT controls these bits directly from the ramp/soak settings dialog. However, you must use ladder logic to control these bits during program execution. We recommend using the bit-of-word instructions.

Ladder logic must set a control bit to a 1 to command the corresponding function. When the loop controller reads the ramp/soak value, it automatically turns off the bit for you. Therefore, a reset of the bit is not required, when the CPU is in Run Mode.



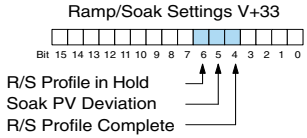
The example program rung to the right shows how an external switch X0 can turn on, and the PD contact uses the leading edge to set the proper control bit to start the ramp soak profile. This uses the Set Bit-of-word instruction.



The normal state for the ramp/soak control bits is all zeros. Ladder logic must set only one control bit at a time.

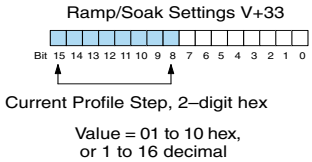
- **Start** – A 0 to 1 transition will start the ramp soak profile. The CPU must be in Run Mode, and the loop can be in Manual or Auto Mode. If the profile is not interrupted by a Hold or Jog command, it finishes normally.
- **Hold** – A 0 to 1 transition will stop the ramp/soak profile in its current state, and the SP value will be frozen.
- **Resume** – A 0 to 1 transition cause the ramp/soak generator to resume operation if it is in the hold state. The SP values will resume from their previous value.
- **Jog** – A 0 to 1 transition will cause the ramp/soak generator to truncate the current segment (step), and go to the next segment.

Ramp/Soak Profile Monitoring



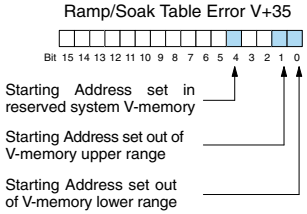
You can monitor the Ramp/Soak profile status using other bits in the Ramp/Soak Settings V+33 word, shown to the right.

- **R/S Profile Complete** – =1 when the last programmed step is done.
- **Soak PV Deviation** – =1 when the error (SP–PV) exceeds the specified deviation in the R/S table.
- **R/S Profile in Hold** – =1 when the profile was active but is now in hold. Ramp/Soak Settings V+33



The number of the current step is available in the upper 8 bits of the Ramp/Soak Settings V+33 word. The bits represent a 2-digit hex number, ranging from 1 to 10. Ladder logic can monitor these to synchronize other parts of the program with the ramp/soak profile. Load this word to the accumulator and shift right 8 bits, and you have the step number.

Ramp/Soak Programming Errors



The starting address for the ramp/soak table must be a valid location. If the address points outside the range of user V-memory, one of the bits to the right will turn on when the ramp/soak generator is started. We recommend using *Direct*SOFT to configure the ramp/soak table. It automatically range checks the addresses for you.

Testing Your Ramp/Soak Profile

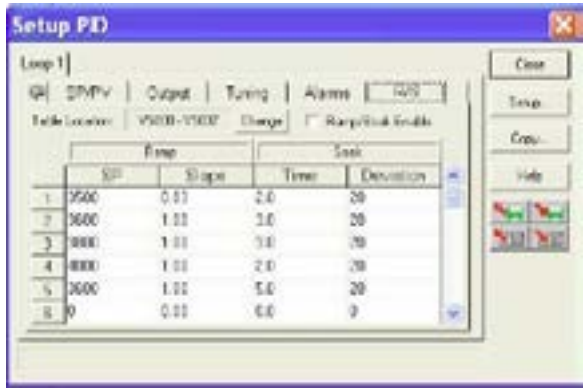
It's a good idea to test your ramp/soak profile before using it to control the process. This is easy to do, because the ramp/soak generator will run even when the loop is in Manual Mode. Using *Direct*SOFT's PID View will be a real time-saver, because it will draw the profile on-screen for you. Be sure to set the trending timebase slow enough to display completed ramp-soak segment pairs in the waveform window.

DirectSOFT Ramp/Soak Example

The following example will step you through the Ramp/Soak setup.

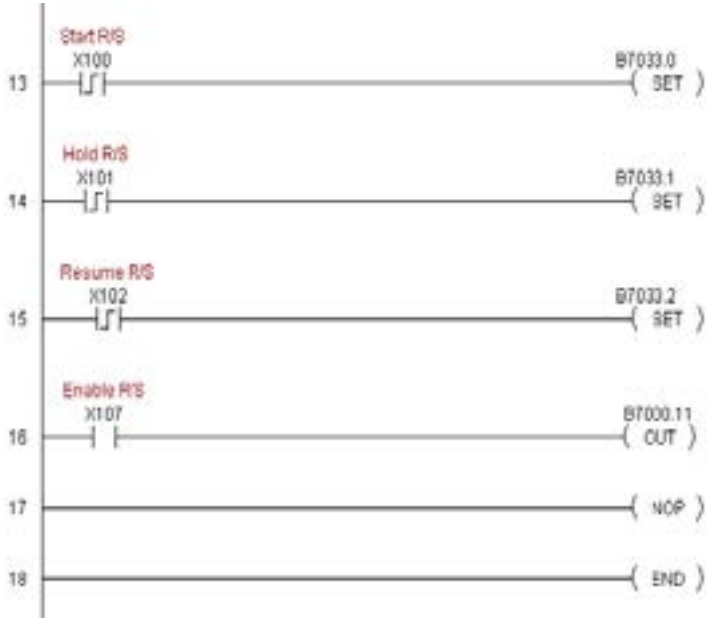
Setup the Profile in PID Setup

The first step is to use Setup PID in *Direct*SOFT to set the profile of your process. Open the Setup PID window and select the R/S tab, and then enter the Ramp and Soak data.



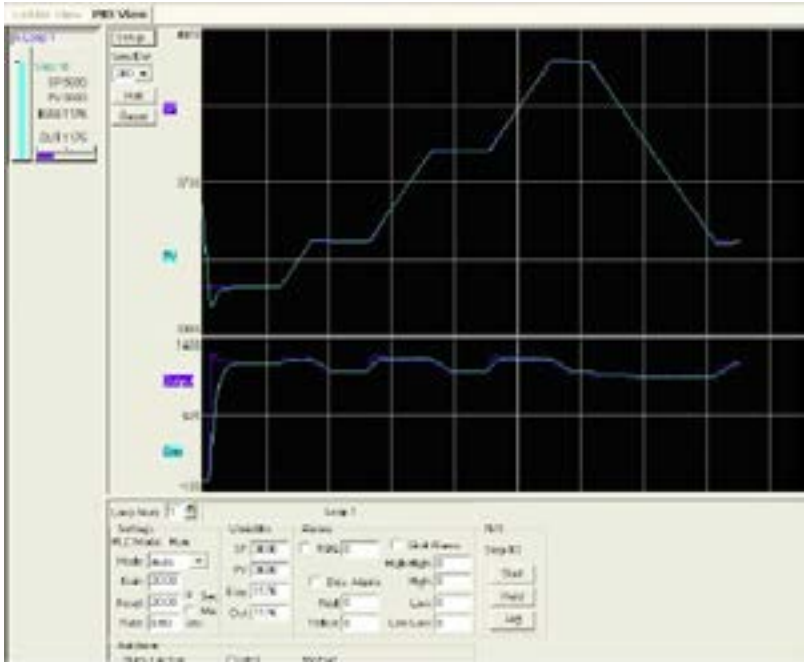
Program the Ramp/Soak Control in Relay Ladder

Refer to the Ramp/Soak Flag Bit Description table on page 8-60 when adding the control rungs to your program similar to the ladder rungs below.



Test the Profile

Test your profile using PID View.



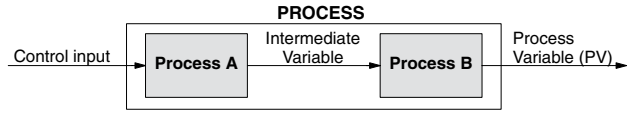
Cascade Control

Introduction

Using cascaded loops is an advanced control technique, superior to individual loop control in certain situations. As the name implies, cascade means that one loop is connected to another loop. In addition to Manual (open loop) and Auto (closed loop) Modes, the DL06 also provides Cascaded Mode.

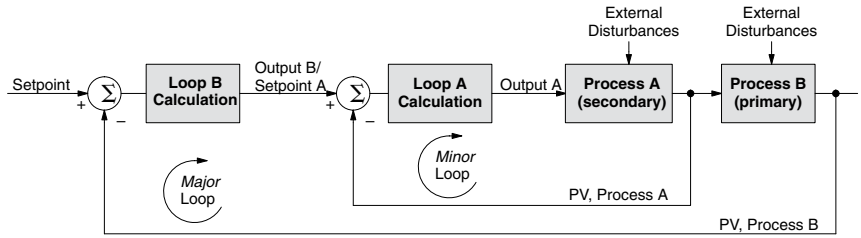
***NOTE**: Using cascaded loops is an advanced process control technique; therefore, we recommend their use only for experienced process control engineers.*

When a manufacturing process is complex and contains a lag time from control input to process variable output, even the most perfectly tuned single loop around the process may yield slow and inaccurate control. It may be the actuator operates on one physical property, which eventually affects the process variable, measured by a different physical property. Identifying the intermediate variable allows us to divide the process into two parts as shown in the following figure.



The principle of cascaded loops is simply that we add another process loop to more precisely control the intermediate variable! This separates the source of the control lag into two parts, as well.

The diagram below shows a cascade control system, showing that it is simply one loop nested inside another. The inside loop is called the minor loop, and the outside loop is called the major loop. For overall stability, the minor loop must be the fastest responding loop of the two (try a factor of 10 for a better response time). We do have to add the additional sensor to measure the intermediate variable (PV for process A). Notice the setpoint for the minor loop is automatically generated for us, by using the output of the major loop. Once the cascaded control is programmed and debugged, we only need to deal with the original setpoint and process variable at the system level. The cascaded loops behave as one loop, but with improved performance over the previous single-loop solution.



One of the benefits to cascade control can be seen by examining its response to external disturbances. Remember the minor loop is faster acting than the major loop. Therefore, if a disturbance affects process A in the minor loop, the Loop A PID calculation can correct the resulting error before the major loop sees the effect.

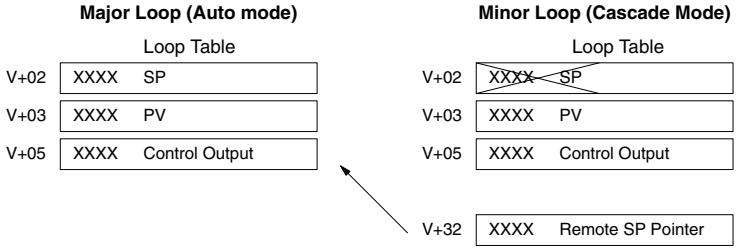
Cascaded Loops in the DL06 CPU

In the use of the term cascaded loops, we must make an important distinction. Only the minor loop will actually be in the Cascade Mode. In normal operation, the major loop must be in Auto Mode. If you have more than two loops cascaded together, the outer-most (major) loop must be in Auto Mode during normal operation, and all inner loops in Cascade Mode.

***NOTE**: Technically, both major and minor loops are cascaded in strict process control terminology. Unfortunately, we are unable to retain this convention when controlling loop modes. Remember that all minor loops will be in Cascade Mode, and only the outer-most (major) loop will be in Auto Mode.*

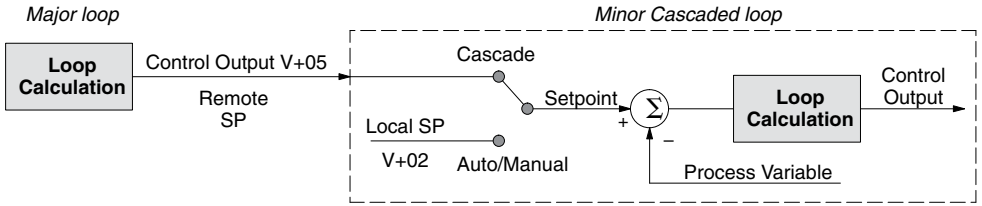
You can cascade together as many loops as necessary on the DL06, and you may have multiple groups of cascaded loops. For proper operation on cascaded loops you must use the same data range (12/15 bit) and unipolar/bipolar settings on the major and minor loop.

To prepare a loop for Cascade Mode operation as a minor loop, you must program its remote Setpoint Pointer in its loop parameter table location V+32, as shown below. The pointer must be the address of the V+05 location (control output) of the major loop. In Cascade Mode, the minor loop will ignore the its local SP register (V+02), and read the major loop's control output as its SP instead.



When using *Direct*SOFT's PID View to watch the SP value of the minor loop, *Direct*SOFT automatically reads the major loop's control output and displays it for the minor loop's SP. The minor loop's normal SP location, V+02, remains unchanged.

Now, we use the loop parameter arrangement above and draw its equivalent loop diagram, shown below.



Remember that a major loop goes to Manual Mode automatically if its minor loop is taken out of Cascade Mode.

Tuning Cascaded Loops

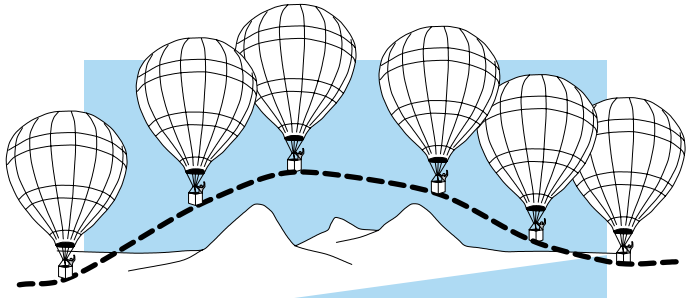
In tuning cascaded loops, you will need to de-couple the cascade relationship and tune the loops individually, using one of the loop tuning procedures previously covered.

1. If you are not using auto tuning, then find the loop sample rate for the minor loop, using the method discussed earlier in this chapter. Then set the sample rate of the major loop slower than the minor loop by a factor of 10. Use this as a starting point.
2. Tune the minor loop first. Leave the major loop in Manual Mode, and you will need to generate SP changes for the minor loop manually as described in the loop tuning procedure.
3. Verify the minor loop gives a critically-damped response to a 10% SP change while in Auto Mode. Then we are finished tuning the minor loop.
4. In this step, you will need to get the minor loop in Cascade Mode, and then the Major loop in Auto Mode. We will be tuning the major loop with the minor loop treated as a series component its overall process. Therefore, do not go back and tune the minor loop again while tuning the major loop.
5. Tune the major loop, following the standard loop tuning procedure in this section. The response of the major loop PV is actually the overall response of the cascaded loops together.

Time-Proportioning Control

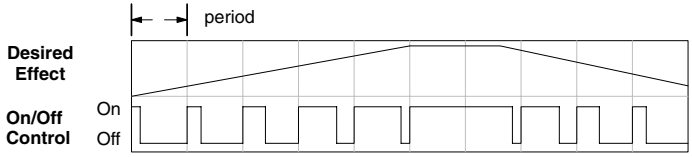
The PID loop controller in the DL06 CPU generates a smooth control output signal across a numerical range. The control output value is suitable to drive an analog output module, which connects to the process. In the process control field, this is called *continuous control*, because the output is on (at some level) continuously.

While continuous control can be smooth and robust, the cost of the loop components (such as actuator, heater amplifiers) can be expensive. A simpler form of control is called *time-proportioning control.* This method uses actuators which are either on or off (no in-between). Loop components for on/off-based control systems are lower cost than their continuous control counterparts.



In this section, we will show you how to convert the control output of a loop to time-proportioning control for the applications that need it. Let's take a moment to review how alternately turning a load on and off can control a process. The diagram below shows a hot-air balloon following a path across some mountains. The desired path is the *setpoint*. The balloon pilot turns the burner on and off alternately, which is his *control output*. The large mass of air in the balloon effectively averages the effect of the burner, converting the bursts of heat into a continuous effect: slowly changing balloon temperature and ultimately the altitude, which is the *process variable*.

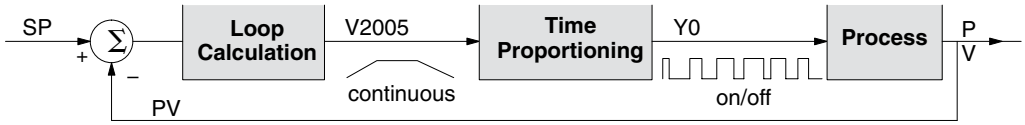
Time-proportioning control approximates continuous control by virtue of its duty-cycle – the ratio of ON time to OFF time. The following figure shows an example of how duty-cycle approximates a continuous level when it is averaged by a large process mass.



If we were to plot the on/off times of the burner in the hot-air balloon, we would probably see a very similar relationship to its effect on balloon temperature and altitude.

On/Off Control Program Example

The following ladder segment provides a time proportioned on/off control output. It converts the continuous output in V2005 to on/off control using the output coil, Y0.

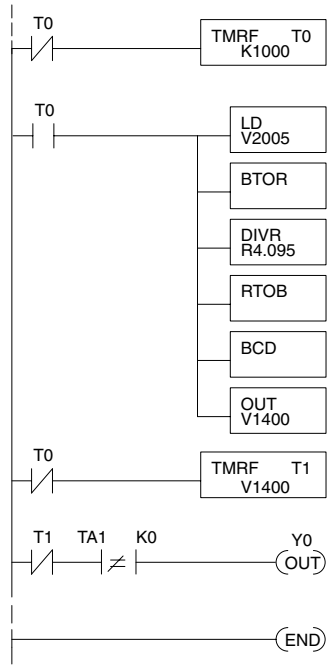


The example program uses two timers to generate On/Off control. It makes the following **assumptions**, which you can alter to fit your application:

- The loop table starts at V2000, so the control output is at V2005.
- The data format of the control output is 12-bit, unipolar (0 – FFF).
- The time base (one full cycle) for the On/Off waveform is 10 seconds. We use a fast timer (0.01 sec/tick), counting to 1000 ticks (10 seconds).
- The On/Off control output is Y0.

The time proportioning program must match the resolution of the output (1 part in 1000) to the resolution of the time base of T0 (also 1 part in 1000).

***NOTE**: Some processes change too fast for time proportioning control. Consider the speed of your process when you choose this control method. Use continuous control for processes that change too fast for time proportioning control. Also, consider using a solid state switch for a longer switch life instead of a relay.*



A fast timer (0.01 sec. timebase) establishes the primary time interval. The constant, K1000, sets the preset at 10 seconds (1,000 ticks). The N.C. enabling contact, T0, makes the timer self-resetting. T0 is on for one scan each 10 seconds, when it resets itself and T1.

At the end of the 10 second period, T0 turns on, and loads the control output value (binary) from the loop table V+05 location (V2005).

The BTOR instruction changes the number in the accumulator to a real number.

Dividing the control output by 4.095, converts the 0 – 4095 range to 0 – 1000, which "matchs" the number of ticks in the 10 second timer range.

This instruction converts the real number back to binary. This step prepares the number for conversion to BCD. There is no real-to-BCD instruction.

Convert the number in the accumulator to BCD format. This satisfies the timer preset format requirement.

Output the result to V1400. In our example, this is the location of the timer preset for the second timer.

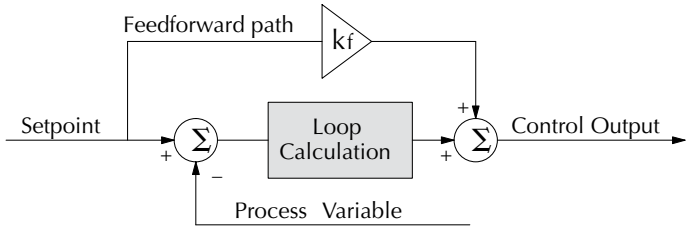
The second fast timer also counts in increments of .01 seconds, so its range is variable from 0 to a maximum of 1000 ticks, or 10 seconds. This timer's output, T1, turns off the output coil, Y0, when the preset is reached.

The N.C. T1 contact, inverts the T1 timer output. The control output is on at the beginning of the 10-second time interval. Y0 turns off when T1 times out. The STRNE contact prevents Y0 from energizing during the one scan when T0 resets T1. Y0 is the actual control output.

END coil marks the end of the main program.

Feedforward Control

Feedforward control is an enhancement to standard closed-loop control. It is most useful for diminishing the effects of a *quantifiable and predictable* loop disturbance or sudden change in setpoint. Use of this feature is an option available to you on the DL06. However, it's best to implement and tune a loop without feedforward, and adding it only if better loop performance is still needed. The term *feedforward* refers to the control technique involved, shown in the diagram below. The incoming setpoint value is fed forward around the PID equation, and summed with the output.



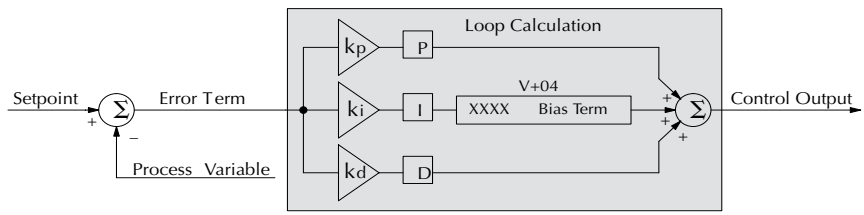
In the previous section on the bias term, we said that "the bias term value establishes a working region or operating point for the control output. *When the error fluctuates around its zero point, the output fluctuates around the bias value.*" Now, when there is a change in setpoint, an error is generated and the output must change to a new operating point. This also happens if a disturbance introduces a new offset in the loop. The loop does not really know its way to the new operating point... the integrator (bias) must increment/decrement until the error disappears, and then the bias has found the new operating point.

Suppose that we are able to know a sudden setpoint change is about to occur (common in some applications). We can avoid much of the resulting error in the first place, if we can quickly change the output to the new operating point. If we know (from previous testing) what the operating point (bias value) will be after the setpoint change, we can artificially change the output directly (which is feedforward). The benefits of using feedforward are:

- The SP–PV error is reduced during predictable setpoint changes or loop offset disturbances.
- Proper use of feedforward will allow us to reduce the integrator gain. Reducing integrator gain gives us an even more stable control system.

Feedforward is very easy to use in the DL06 loop controller, as shown below. The bias term has been made available to the user in a special read/write location, at PID Parameter Table location V+04.

Parameter Table location V+04.

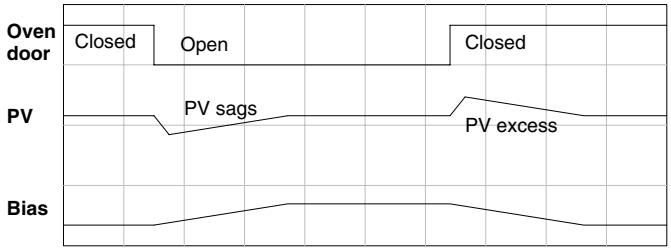


To change the bias (operating point), ladder logic only has to write the desired value to V+04. The PID loop calculation first reads the bias value from V+04 and modifies the value based on the current integrator calculation. Then it writes the result back to location V+04. This arrangement creates a sort of transparent bias term. All you have to do to implement feed forward control is write the correct value to the bias term at the right time (see the following example).

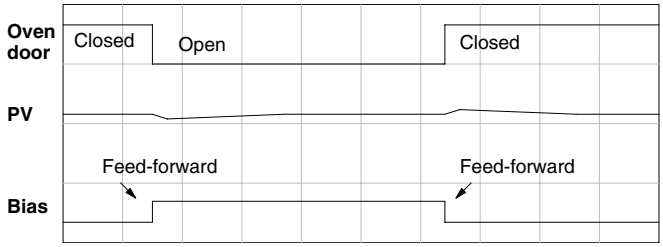
***NOTE**: When writing the bias term, one must be careful to design ladder logic to write the value only once, at the moment when the new bias operating point is to occur. If ladder logic writes the bias value on every scan, the loop's integrator is effectively disabled.*

Feedforward Example

How do we know when to write to the bias term, and what value to write? Suppose we have an oven temperature control loop, and we have already tuned the loop for optimal performance. Refer to the figure below. We notice that when the operator opens the oven door, the temperature sags a bit while the loop bias adjusts to the heat loss. Then, when the door closes, the temperature rises above the SP until the loop adjusts again. Feedforward control can help diminish this effect.



First, we record the amount of bias change the loop controller generates when the door opens or closes. Then, we write a ladder program to monitor the position of an oven door limit switch. When the door opens, our ladder program reads the current bias value from V+04, adds the desired change amount, and writes it back to V+04. When the door closes, we duplicate the procedure, but subtracting desired change amount instead. The following figure shows the results.



The step changes in the bias are the result of our two feed-forward writes to the bias term. We can see the PV variations are greatly reduced. The same technique may be applied for changes in setpoint.

PID Example Program

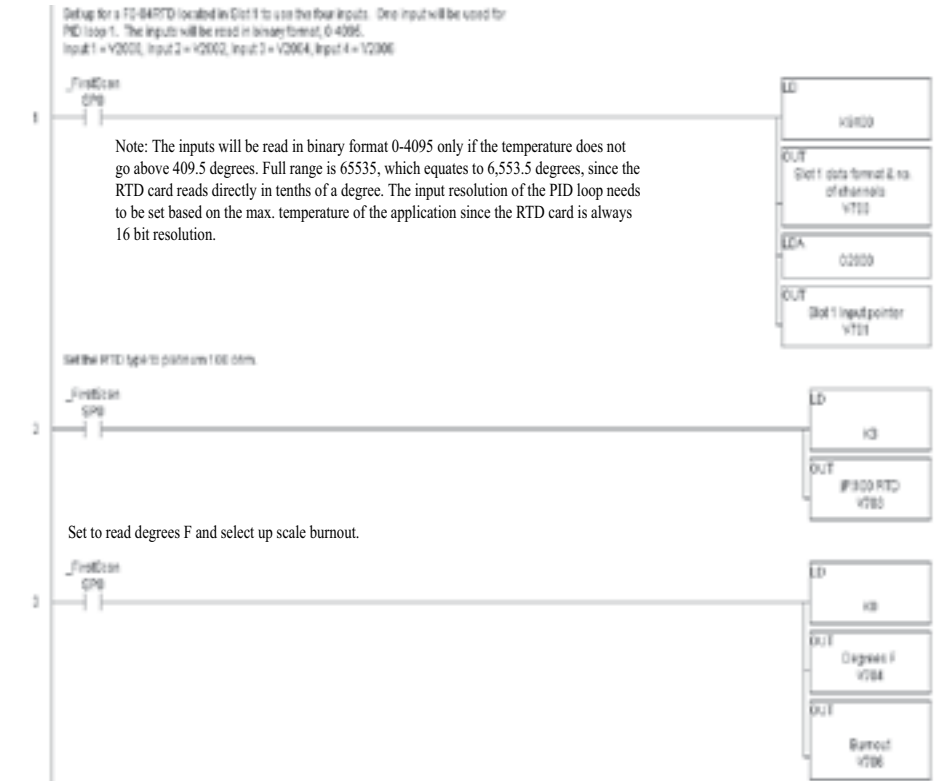
Program Setup for the PID Loop

After setting up the PID loop or loops, with *Direct*SOFT, you will need to edit your RLL program to include the rungs needed to setup the analog I/O module to be used by the PID loop(s).

The following example program shows how an RTD module, F0-04RTD, and an analog combination module, F0-2AD2DA-2, are used and setup for a PID loop. This example assumes that the PID table for loop 1 has a beginning address of V2100.

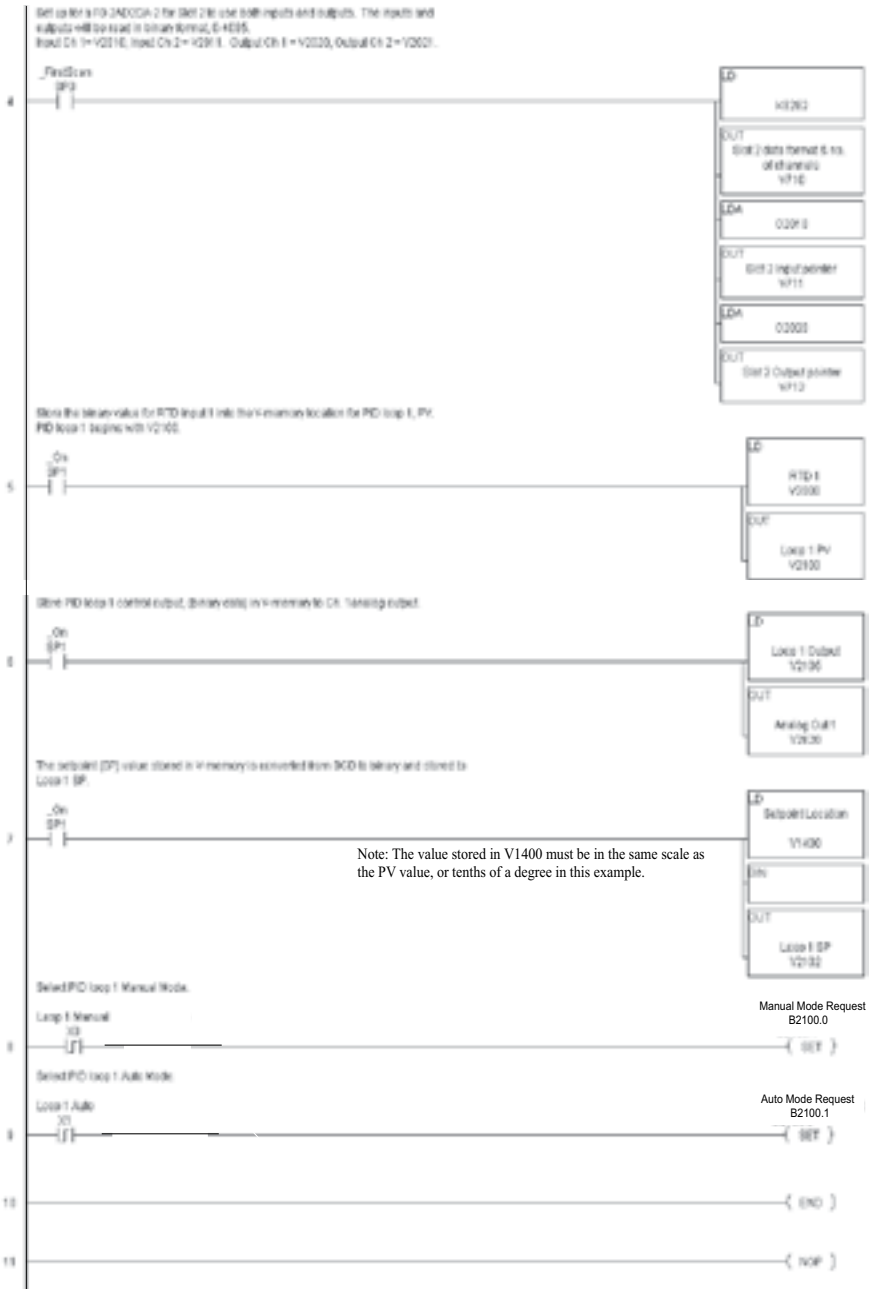
All of the analog I/O modules used with the DL06 are setup in a similar manner. Refer to the DL05/DL06 Options Manual for the setup information for the particular module that you will be using.

***Direct*SOFT**



Program continued on next page

Example program continued



Note that the modules used in the PID loop example program were set up for binary format. They could have been set up for BCD format. In the latter case, the BCD data would have to be converted to binary format before being stored to the setpoint and process variable, and the control output would have to be converted from BCD to binary before being stored to the analog output.

By following the steps outlined in this chapter, you should be able to set up workable PID control loops. The ***Direct***SOFT Programming Software Manual provides more information for the use of PID View.

For a step-by-step tutorial, go to the Technical Support section located on our website, **www.automationdirect.com**. Once you are at the website, click on **Technical Support Home**. After this page opens, find and select **Guided Tutorials** located under the **Using Your Products** column. An **Animated Tutorial** page will open. Under **Available Tutorials**, find **PID Trainer** and select **View the Powerpoint slide show** and begin viewing the tutorial. The Powerpoint Viewer can be downloaded if your computer does not have Powerpoint installed.

Troubleshooting Tips

Q. The loop will not go into Automatic Mode.

A. Check the following for possible causes:

- A PV alarm exists, or a PV alarm programming error exists.
- The loop is the major loop of a cascaded pair, and the minor loop is not in Cascade Mode.

Q. The Control Output stays at zero constantly when the loop is in Automatic Mode.

A. Check the following for possible causes:

- The Control Output upper limit in loop table location V+31 is zero.
- The loop is driven into saturation, because the error never goes to zero value and changes (algebraic) sign.

Q. The Control Output value is not zero, but it is incorrect.

A. Check the following for possible causes:

- The gain values are entered improperly. Remember, gains are entered in the loop table in BCD, while the SP and PV are in binary. If you are using ***Direct***SOFT, it displays the SP, PV, Bias and Control output in decimal (BCD), converting it to binary before updating the loop table.

Q. The Ramp/Soak Generator does not operate when I activate the Start bit.

A. Check the following for possible causes:

- The Ramp/Soak enable bit is off. Check the status of bit 11 of loop parameter table location V+00. It must be set =1.
- The hold bit or other bits in the Ramp/Soak control are on.
- The beginning SP value and the first ramp ending SP value are the same, so first ramp segment has no slope and consequently has no duration. The ramp/soak generator moves quickly to the soak segment, giving the illusion the first ramp is not working.
- The loop is in Cascade Mode, and is trying to get the SP remotely.
- The SP upper limit value in the loop table location V+27 is too low.
- Check your ladder program to verify it is not writing to the SP location (V+02 in the loop table). A quick way to do this is to temporarily place an end coil at the beginning of your program, then go to PLC Run Mode, and manually start the ramp/soak generator.

Q. The PV value in the table is constant, even though the analog module receives the PV signal.

A. Your ladder program must read the analog value from the module successfully and write it into the loop table V+03 location. Verify the analog module is generating the value, and the ladder is working.

Q. The Derivative gain doesn't seem to have any affect on the output.

A. The derivative limit is probably enabled (see section on derivative gain limiting).

Q. The loop Setpoint appears to be changing by itself.

A. Check the following for possible causes:

- The Ramp/Soak generator is enabled, and is generating setpoints.
- If this symptom occurs on loop Manual-to-Auto Mode changes, the loop is in Bumpless Transfer 1.
- Check your ladder program to verify it is not writing to the SP location (V+02 in the loop table). A quick way to do this is to temporarily place an end coil at the beginning of your program, then go to PLC Run Mode.

Q. The SP and PV values I enter with DirectSOFT work okay, but these values do not work properly when the ladder program writes the data.

A. The PID View in *Direct*SOFT lets you enter SP, PV, and Bias values in decimal, and displays them in decimal for your convenience. For example, when the data format is 12 bit unipolar, the values range from 0 to 4095. However, the loop table actually requires these in hex, so *Direct*SOFT converts them for you. The values in the table range from 0 to FFF, for 12-bit unipolar format.

Q. The loop seems unstable and impossible to tune, no matter what gains I use.

A. Check the following for possible causes:

- The loop sample time is set too long. Refer to the section near the front of this chapter on selecting the loop update time.
- The gains are too high. Start out by reducing the derivative gain to zero. Then reduce the integral gain, and the proportional gain if necessary.
- There is too much transfer lag in your process. This means the PV reacts sluggishly to control output changes. There may be too much "distance" between actuator and PV sensor, or the actuator may be weak in its ability to transfer energy into the process.
- There may be a process disturbance that is over-powering the loop. Make sure the PV is relatively steady when the SP is not changing.

Glossary of PID Loop Terminology

Automatic Mode An operational mode of a loop, in which it makes PID calculations and updates the loop's control output.

Bias Freeze A method of preserving the bias value (operating point) for a control output, by inhibiting the integrator when the output goes out-of-range. The benefit is a faster loop recovery.

Bias Term In the position form of the PID equation, it is the sum of the integrator and the initial control output value.

Bumpless Transfer A method of changing the operation mode of a loop while avoiding the usual sudden change in control output level. This consequence is avoided by artificially making the SP and PV equal, or the bias term and control output equal at the moment of mode change.

Cascaded Loops A cascaded loop receives its setpoint from the output of another loop. Cascaded loops have a major/minor relationship, and work together to ultimately control one PV.

Cascade Mode An operational mode of a loop, in which it receives its SP from another loop's output.

Continuous Control Control of a process done by delivering a smooth (analog) signal as the control output.

Control Output The numerical result of a PID equation which is sent by the loop with the intention of nulling out the current error.

Derivative Gain A constant that determines the magnitude of the PID derivative term in response to the current error.

Direct-Acting Loop A loop in which the PV increases in response to a control output increase. In other words, the process has a positive gain.

Error The difference in value between the SP and PV, Error = SP – PV

Error Deadband An optional feature which makes the loop insensitive to errors when they are small. You can specify the size of the deadband.

Error Squared An optional feature which multiplies the error by itself, but retains the original algebraic sign. It reduces the effect of small errors, while magnifying the effect of large errors.

Feedforward A method of optimizing the control response of a loop when a change in setpoint or disturbance offset is known and has a quantifiable effect on the bias term.

Integral Gain A constant that determines the magnitude of the PID integral term in response to the current error.

Major Loop In cascade control, it is the loop that generates a setpoint for the cascaded loop.

Manual Mode An operational mode of a loop, it which the PID calculations are stopped. The operator must manually control the loop by writing to the control output value directly.

Minor Loop In cascade control, the minor loop is the subordinate loop that receives its SP from the major loop.

On/Off Control A simple method of controlling a process, through on/off application of energy into the system. The mass of the process averages the on/off effect for a relatively smooth PV. A simple ladder program can convert the DL06's continuous loop output to on/off control.

PID Loop A mathematical method of closed-loop control involving the sum of three terms based on proportional, integral, and derivative error values. The three terms have independent gain constants, allowing one to optimize (tune) the loop for a particular physical system.

Position Algorithm The control output is calculated so it responds to the displacement (position) of the PV from the SP (error term)

Process A manufacturing procedure which adds value to raw materials. Process control particularly refers to inducing chemical changes to the material in process.

Process Variable (PV) A quantitative measurement of a physical property of the material in process, which affects final product quality and is important to monitor and control.

Proportional Gain A constant that determines the magnitude of the PID proportional term in response to the current error.

PV Absolute Alarm A programmable alarm that compares the PV value to alarm threshold values.

PV Deviation Alarm A programmable alarm that compares the difference between the SP and PV values to a deviation threshold value.

Ramp/Soak Profile A set of SP values called a profile, which is generated in real time upon each loop calculation. The profile consists of a series of ramp and soak segment pairs, greatly simplifying the task of programming the PLC to generate such SP sequences.

Rate Also called differentiator, the rate term responds to the changes in the error term.

Remote Setpoint The location where a loop reads its setpoint when it is configured as the minor loop in a cascaded loop topology.

Reset Also called integrator, the reset term adds each sampled error to the previous, maintaining a running total called the bias.

Reset Windup A condition created when the loop is unable to find equilibrium, and the persistent error causes the integrator (reset) sum to grow excessively (windup). Reset windup causes an extra recovery delay when the original loop fault is remedied.

Reverse-Acting Loop A loop in which the PV increases in response to a control output decrease. In other words, the process has a negative gain.

Sampling time The time between PID calculations. The CPU method of process control is called a sampling controller, because it samples the SP and PV only periodically.

Setpoint (SP) The desired value for the process variable. The setpoint (SP) is the input command to the loop controller during closed loop operation.

Soak Deviation The soak deviation is a measure of the difference between the SP and PV during a soak segment of the Ramp/Soak profile, when the Ramp/Soak generator is active.

Step Response The behavior of the process variable in response to a step change in the SP (in closed loop operation), or a step change in the control output (in open loop operation)

Transfer To change from one loop operational mode to another (between Manual, Auto, or Cascade). The word "transfer" probably refers to the transfer of control of the control output or the SP, depending on the particular mode change.

Velocity Algorithm The control output is calculated to represent the rate of change (velocity) for the PV to become equal to the SP.

Bibliography

Fundamentals of Process Control Theory, Second Edition Author: Paul W. Murrill Publisher: Instrument Society of America ISBN 1–55617–297–4	Application Concepts of Process Control Author: Paul W. Murrill Publisher: Instrument Society of America ISBN 1–55617–080–7
PID Controllers: Theory, Design, and Tuning, 2nd Edition Author: K. Astrom and T Hagglund Publisher: Instrument Society of America ISBN 1–55617–516–7	Fundamentals of Temperature, Pressure, and Flow Measurements, Third edition Author: Robert P. Benedict Publisher: John Wiley and Sons ISBN 0–471–89383–8
Process / Industrial Instruments & Controls Handbook, Fourth Edition Author (Editor-in-Chief): Douglas M. Considine Publisher: McGraw-Hill, Inc ISBN 0-07-012445-0	pH Measurement and Control, Second Edition Author: Gregory K. McMillan Publisher: Instrument Society of America ISBN 1–55617–483–7
Programmable Controllers Concepts and Applications, First Edition Authors: C.T. Jones and L.A. Bryan Publisher: International Programmable Controls ISBN 0-915425-00-9	Fundamentals of Programmable Logic Controllers, Sensors, and Communications Author: Jon Stenerson Publisher: Prentice Hall ISBN 0-13-726860-2
Process Control, Third Edition Instrument Engineer's Handbook Author (Editor-in-Chief): Bela G. Liptak Publisher: Chilton ISBN 0–8019–8242–1	Process Measurement and Analysis, Third Edition Instrument Engineer's Handbook Author (Editor-in-Chief): Bela G. Liptak Publisher: Chilton ISBN 0–8019–8197–2

MAINTENANCE AND TROUBLESHOOTING

In This Chapter...

Hardware System Maintenance

Standard Maintenance

No regular or preventative maintenance is required for this product (there are no internal batteries); however, a routine maintenance check (about every one or two months) of your PLC and control system is good practice, and should include the following items:

- Air Temperature – Monitor the air temperature in the control cabinet, so the operating temperature range of any component is not exceeded.
- Air Filter – If the control cabinet has an air filter, clean or replace it periodically as required.
- Fuses or breakers – Verify that all fuses and breakers are intact.
- Cleaning the Unit – Check that all air vents are clear. If the exterior case needs cleaning, disconnect the input power, and carefully wipe the case using a damp cloth. Do not let water enter the case through the air vents and do not use strong detergents because this may discolor the case.

Diagnostics

Diagnostics

Your DL06 Micro PLC performs many pre-defined diagnostic routines with every CPU scan. The diagnostics can detect various errors or failures in the PLC. The two primary error classes are *fatal and non-fatal.*

Fatal Errors

Fatal errors are errors which may cause the system to function improperly, perhaps introducing a safety problem. The CPU will automatically switch to Program Mode if it is in Run Mode. (Remember, in Program Mode all outputs are turned off.) If the fatal error is detected while the CPU is in Program Mode, the CPU will not allow you to transition to Run Mode until the error has been corrected.

Some examples of fatal errors are:

- Power supply failure
- Parity error or CPU malfunction
- Particular programming errors

Non-fatal Errors

Non-fatal errors are errors that need your attention, but should not cause improper operation. They do not cause or prevent any mode transitions of the CPU. The application program can use special relay contacts to detect non-fatal errors, and even take the system to an orderly shutdown or switch the CPU to Program Mode if desired. An example of a non-fatal error is:

- Particular programming errors - The programming devices will notify you of an error if one occurs while online.
- *Direct*SOFT provides the error number and an error message.
- The handheld programmer displays error numbers and short descriptions of the error.

Appendix B has a complete list of error messages in order by error number. Many error messages point to supplemental V-memory locations which contain related information. Special relays (SP contacts) also provide error indications (refer to Appendix D).

V-memory Error Code Locations

The following table names the specific memory locations that correspond to certain types of error messages.

Error Class	Error Category	Diagnostic V-memory
User-Defined	Error code used with FAULT instruction	V7751
System Error	Fatal Error code	V7755
	Major Error code	V7756
	Minor Error code	V7757
Grammatical	Address where syntax error occurs	V7763
	Error Code found during syntax check	V7764
CPU Scan	Number of scans since last Program to Run Mode transition	V7765
	Current scan time (ms)	V7775
	Minimum scan time (ms)	V7776
	Maximum scan time (ms)	V7777

Special Relays (SP) Corresponding to Error Codes

The special relay table also includes status indicators which can indicate errors. For a more detailed description of each of these special relays refer to Appendix D.

CPU Status Relays	
SP11	Forced Run mode
SP12	Terminal Run mode
SP13	Test Run mode
SP15	Test stop mode
SP16	Terminal Program mode
SP17	Forced stop
SP20	STOP instruction was executed
SP22	Interrupt enabled
System Monitoring Relays	
SP36	Override setup
SP37	Scan control error
SP40	Critical error
SP41	Non-critical error
SP42	Diagnostics error
SP44	Program memory error
SP45	I/O error
SP46	Communications error
SP50	Fault instruction was executed
SP51	Watchdog timeout
SP52	Syntax error
SP53	Cannot solve the logic
SP54	Communication error
SP56	Table instruction overrun

Accumulator Status Relays	
SP60	Acc. is less than value
SP61	Acc. is equal to value
SP62	Acc. is greater than value
SP63	Acc. result is zero
SP64	Half borrow occurred
SP65	Borrow occurred
SP66	Half carry occurred
SP67	Carry occurred
SP70	Result is negative (sign)
SP71	Pointer reference error
SP73	Overflow
SP75	Data is not in BCD
SP76	Load zero

DL06 Micro PLC Error Codes

These errors can be generated by the CPU or by the Handheld Programmer, depending on the actual error. Appendix B provides a more complete description of the error codes.

The errors can be detected at various times. However, most of them are detected at power-up, on entry to Run Mode, or when a Handheld Programmer key sequence results in an error or an illegal request.

Error Code	Description
E003	Software time-out
E004	Invalid instruction(RAM parity error in the CPU)
E104	Write failed
E151	Invalid command
E311	Communications error 1
E312	Communications error 2
E313	Communications error 3
E316	Communications error 6
E320	Time out
E321	Communications error
E360	HP Peripheral port time-out
E501	Bad entry
E502	Bad address
E503	Bad command
E504	Bad reference / value
E505	Invalid instruction
E506	Invalid operation
E520	Bad operation – CPU in Run
E521	Bad operation – CPU in Test Run
E523	Bad operation – CPU in Test Program
E524	Bad operation – CPU in Program
E525	Mode Switch not in Term position

Error Code	Description
E526	Unit is offline
E527	Unit is online
E528	CPU mode
E540	CPU locked
E541	Wrong password
E542	Password reset
E601	Memory full
E602	Instruction missing
E604	Reference missing
E620	Out of memory
E621	EEPROM Memory not blank
E622	No Handheld Programmer EEPROM
E624	V memory only
E625	Program only
E627	Bad write operation
E628	Memory type error (should be EEPROM)
E640	Mis-compare
E650	Handheld Programmer system error
E651	Handheld Programmer ROM error
E652	Handheld Programmer RAM error

Program Error Codes

The following table lists program syntax and runtime error codes. Error detection occurs during a Program-to-Run mode transition, or when you use AUX 21 – Check Program. The CPU will also turn on SP52 and store the error code in V7755. Appendix B provides a more complete description of the error codes.

Error Code	Description
E4**	No Program in CPU
E401	Missing END statement
E402	Missing LBL
E403	Missing RET
E404	Missing FOR
E405	Missing NEXT
E406	Missing IRT
E412	SBR / LBL >64
E421	Duplicate Stage reference
E422	Duplicate SBR/LBL reference
E423	Nested Loop
E431	Invalid ISG/SG address
E433	Invalid ISG / SG address
E434	Invalid RTC
E435	Invalid RT
E436	Invalid INT address
E437	Invalid IRTC

Error Code	Description
E438	Invalid IRT address
E440	Invalid Data Address
E441	ACON/NCON
E451	Bad MLS/MLR
E453	Missing T/C
E454	Bad TMRA
E455	Bad CNT
E456	Bad SR
E461	Stack Overflow
E462	Stack Underflow
E463	Logic Error
E464	Missing Circuit
E471	Duplicate Coil reference
E472	Duplicate TMR reference
E473	Duplicate CNT reference
E499	Print Instruction

CPU Indicators

The DL06 Micro PLCs have indicators on the front to help you determine potential problems with the system. In normal runtime operation only, the RUN and PWR indicators are on. The table below is a quick reference to potential problems.

Indicator Status	Potential Problems
PWR (Green LED off)	System voltage incorrect
	PLC power supply faulty
RUN (Green LED off)	CPU programming error
	CPU in program mode
RUN (Green LED flashing)	CPU in firmware upgrade mode
CPU (Red LED on)	Electrical noise interference
	Internal CPU defective
CPU (Blinking Red LED)	Low backup battery (refer to page 3-8)

PWR Indicator

In general there are three reasons for the CPU power status LED (PWR) to be OFF:

1. Power to the unit is incorrect or is not applied.
2. PLC power supply is faulty.
3. Other component(s) have the power supply shut down.

If the voltage to the power supply is not correct, the PLC may not operate properly or may not operate at all. Use the following guidelines to correct the problem.

WARNING: To minimize the risk of electrical shock, always disconnect the system power before inspecting the physical wiring.

1. First, disconnect the external power.
2. Verify that all external circuit breakers or fuses are still intact.
3. Check all incoming wiring for loose connections. If you're using a separate termination block, check those connections for accuracy and integrity.
4. If the connections are acceptable, reconnect the system power and verify the voltage at the DL06 power input is within specification. If the voltage is not correct, shut down the system and correct the problem.
5. If all wiring is connected correctly and the incoming power is within the specifications, the PLC internal supply may be faulty.

The best way to check for a faulty PLC is to substitute a known good one to see if this corrects the problem. The removable connectors on the DL06 make this relatively easy. If there has been a major power surge, it is possible the PLC internal power supply has been damaged. If you suspect this is the cause of the power supply damage, consider installing an AC line conditioner to attenuate damaging voltage spikes in the future.

RUN Indicator

If the CPU will not enter the Run mode (the RUN indicator is off), the problem is usually in the application program, unless the CPU has a fatal error. If a fatal error has occurred, the CPU LED should be on. (You can use a programming device to determine the cause of the error.)

Both of the programming devices, Handheld Programmer and *Direct*SOFT, will return an error message describing the problem. Depending on the error, there may also be an AUX function you can use to help diagnose the problem. The most common programming error is "Missing END Statement". All application programs require an END statement for proper termination. A complete list of error codes can be found in Appendix B.

The RUN indicator will flash (blink) whenever the CPU is in the firmware upgrade mode.

CPU Indicator

If the CPU indicator is on, a fatal error has occurred in the CPU. Generally, this is not a programming problem but an actual hardware failure. You can power cycle the system to clear the error. If the error clears, you should monitor the system and determine what caused the problem. You will find this problem is sometimes caused by high frequency electrical noise introduced into the CPU from an outside source. Check your system grounding and install electrical noise filters if the grounding is suspected. If power cycling the system does not reset the error, or if the problem returns, you should replace the CPU.

If the CPU indicator is blinking, the backup battery is low (refer to page 3-8).

Communications Problems

If you cannot establish communications with the CPU, check these items.

- The cable is disconnected.
- The cable has a broken wire or has been wired incorrectly.
- The cable is improperly terminated or grounded.
- The device connected is not operating at the correct baud rate (9600 baud).
- The device connected to the port is sending data incorrectly, or another application is running on the device.
- A grounding difference exists between the two devices.
- Electrical noise is causing intermittent errors.
- The PLC has a bad communication port and should be replaced.

For problems in communicating with *Direct*SOFT on a personal computer, refer to the *Direct*SOFT programming user manual. It includes a troubleshooting section that can help you diagnose PC problems in communications port setup, address or interrupt conflicts, etc.

I/O Point Troubleshooting

Possible Causes

If you suspect an I/O error, there are several things that could be causing the problem.

- High-Speed I/O configuration error
- A blown fuse in your machine or panel (the DL06 does not have internal I/O fuses)
- A loose terminal block
- The auxiliary 24 VDC supply has failed
- The Input or Output Circuit has failed

Some Quick Steps

When troubleshooting the DL06 Micro PLCs, please be aware of the following facts which may assist you in quickly correcting an I/O problem.

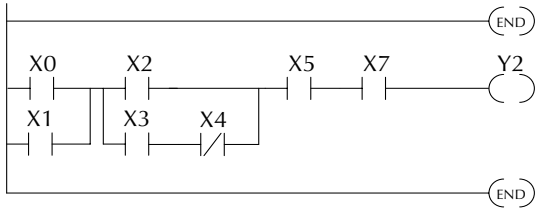
- HSIO configuration errors are commonly mistaken for I/O point failure during program development. If the I/O point in question is in X0–X2, or Y0–Y1, check all parameter locations listed in Chapter 3 that apply to the HSIO mode you have selected.
- The output circuits cannot detect shorted or open output points. If you suspect one or more faulty points, measure the voltage drop from the common to the suspect point. Remember when using a Digital Volt Meter, leakage current from an output device such as a triac or a transistor must be considered. A point which is off may appear to be on if no load is connected the point.
- The I/O point status indicators are logic-side indicators. This means the LED which indicates the on or off status reflects the status of the point with respect to the CPU. On an output point the status indicators could be operating normally while the actual output device (transistor, triac etc.) could be damaged. With an input point, if the indicator LED is on, the input circuitry is probably operating properly. Verify the LED goes off when the input signal is removed.
- Leakage current can be a problem when connecting field devices to an I/O point. False input signals can be generated when the leakage current of an output device is great enough to turn on the connected input device. To correct this install a resistor in parallel with the input or output of the circuit. The value of this resistor will depend on the amount of leakage current and the voltage applied but usually a 10K to 20K resistor will work. Verify the wattage rating of the resistor is correct for your application.
- Because of the removable terminal blocks on the DL06, the easiest method to determine if an I/O circuit has failed is to replace the unit if you have a spare. However, if you suspect a field device is defective, that device may cause the same failure in the replacement PLC as well. As a point of caution, you may want to check devices or power supplies connected to the failed I/O circuit before replacing the unit with a spare.

Output points can be set on or off in the DL06 series CPUs. If you want to do an I/O check-out independent of the application program, follow the procedure below:

Step	Action
1	Use a handheld programmer or *Direct*SOFT to communicate online to the PLC.
2	Change to Program Mode.
3	Go to address 0.
4	Insert an "END" statement at address 0. (This will cause program execution to occur only at address 0 and prevent the application program from turning the I/O points on or off).
5	Change to Run Mode.
6	Use the programming device to set (turn) on or off the points you wish to test.
7	When you finish testing I/O points delete the "END" statement at address 0.

WARNING: Depending on your application, forcing I/O points may cause unpredictable machine operation that can result in a risk of personal injury or equipment damage. Make sure you have taken all appropriate safety precautions prior to testing any I/O points.

Handheld Programmer Keystrokes Used to Test an Output Point



Insert an END statement at the beginning of the program. This disables the remainder of the program.

From a clear display, use the following keystrokes

STAT ENT

```
16P STATUS
BIT REF   X
```

Use the PREV or NEXT keys to select the Y data type

NEXT A 0 ENT

```
      Y 10          Y0
□□□□□□□□□□□□□□□□
```

Use arrow keys to select point, then use ON and OFF to change the status

← ← SHFT ON INS

Y2 is now on

```
      Y 10          Y0
□□□□□□□□□□□□□■□□
```

Noise Troubleshooting

Electrical Noise Problems

Noise is one of the most difficult problems to diagnose. Electrical noise can enter a system in many different ways and falls into one of two categories, conducted or radiated. It may be difficult to determine how the noise is entering the system but the corrective actions for either of the types of noise problems are similar.

- Conducted noise is when the electrical interference is introduced into the system by way of an attached wire, panel connection ,etc. It may enter through an I/O circuit, a power supply connection, the communication ground connection, or the chassis ground connection.
- Radiated noise is when the electrical interference is introduced into the system without a direct electrical connection, much in the same manner as radio waves.

Reducing Electrical Noise

While electrical noise cannot be eliminated it can be reduced to a level that will not affect the system.

- Most noise problems result from improper grounding of the system. A good earth ground can be the single most effective way to correct noise problems. If a ground is not available, install a ground rod as close to the system as possible. Ensure all ground wires are single point grounds and are not daisy chained from one device to another. Ground metal enclosures around the system. A loose wire can act as a large antenna, introducing noise into the system, so, tighten all connections in your system. Loose ground wires are more susceptible to noise than the other wires in your system. Review Chapter 2 Installation, Wiring, and Specifications if you have questions regarding how to ground your system.
- Electrical noise can enter the system through the power source for the PLC and I/O circuits. Installing an isolation transformer for all AC sources can correct this problem. DC sources should be well-grounded good quality supplies.
- Separate input wiring from output wiring. Never run low-voltage I/O wiring close to high voltage wiring.

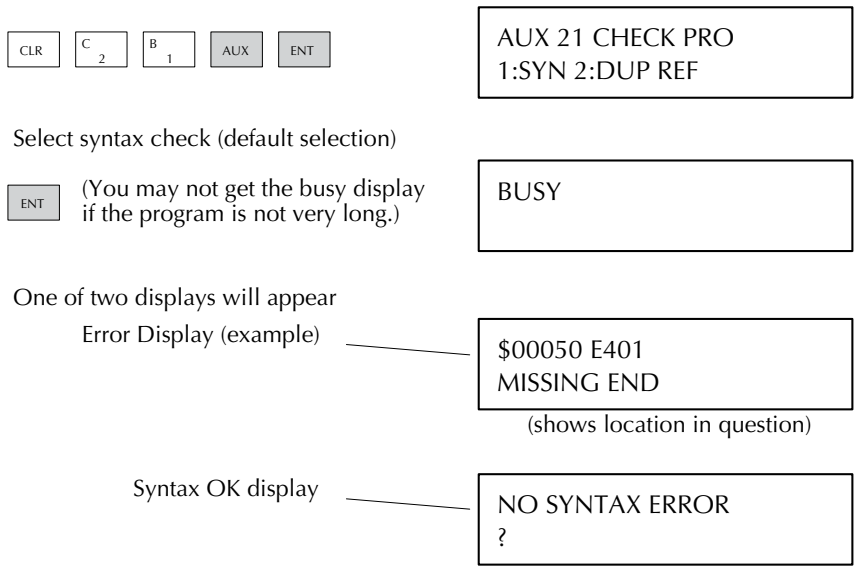
Machine Startup and Program Troubleshooting

The DL06 Micro PLCs provide several features that can help you debug your program before and during machine startup. This section discusses the following topics which can be very helpful.

- Program Syntax Check
- Duplicate Reference Check
- Special Instructions
- Run Time Edits
- Forcing I/O Points

Syntax Check

Even though the Handheld Programmer and ***Direct***SOFT provide error checking during program entry, you may want to check a program that has been modified. Both programming devices offer a way to check the program syntax. For example, you can use AUX 21, CHECK PROGRAM to check the program syntax from a Handheld Programmer, or you can use the PLC Diagnostics menu option within ***Direct***SOFT. This check will find a wide variety of programming errors. The following example shows how to use the syntax check with a Handheld Programmer.



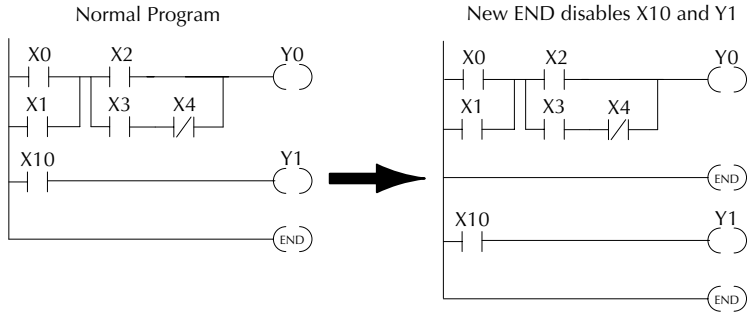
See the Error Codes Section for a complete listing of programming error codes. If you get an error, just press CLR and the Handheld will display the instruction where the error occurred. Correct the problem and continue running the Syntax check until the NO SYNTAX ERROR message appears.

Special Instructions

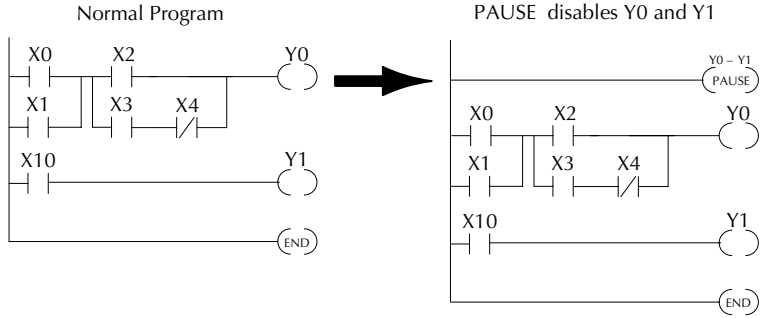
There are several instructions that can be used to help you debug your program during machine startup operations.

- END
- PAUSE
- STOP

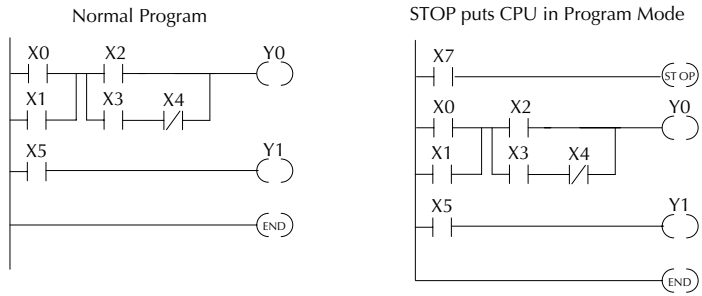
END Instruction: If you need a way to quickly disable part of the program, just insert an END statement prior to the portion that should be disabled. When the CPU encounters the END statement, it assumes that is the end of the program. The following diagram shows an example.



PAUSE Instruction: This instruction provides a quick way to allow the inputs (or other logic) to operate while disabling selected outputs. The output image register is still updated, but the output circuits are not. For example, you could make this conditional by adding an input contact or CR to control the instruction with a switch or a programming device. Or, you could just add the instruction without any conditions so the selected outputs would be disabled at all times.



STOP Instruction: Sometimes during machine startup you need a way to quickly turn off all the outputs and return to Program Mode. You can use the STOP instruction. When this instruction is executed the CPU automatically exits Run Mode and enters Program Mode. Remember, all outputs are turned off during Program Mode. The following diagram shows an example of a condition that returns the CPU to Program Mode.



In the example shown above, you could trigger X7 which would execute the STOP instruction. The CPU would enter Program Mode and all outputs would be turned off.

Duplicate Reference Check

You can also check for multiple uses of the same output coil. Both programming devices offer a way to check for this condition.. For example, you can AUX 21, CHECK PROGRAM to check for duplicate references from a Handheld Programmer, or you can use the PLC Diagnostics menu option within ***Direct***SOFT. The following example shows how to perform the duplicate reference check with a Handheld Programmer.

If you get an error, just press CLR and the Handheld will display the instruction where the error occurred. Correct the problem and continue running the Duplicate Reference check until no duplicate references are found.

Use AUX 21 to perform syntax check

CLR | C 2 | B 1 | AUX | ENT

```
AUX 21 CHECK PRO
1:SYN 2:DUP REF
```

Select duplicate reference check

→ | ENT (You may not get the busy display if the program is not very long.)

```
BUSY
```

One of two displays will appear

Error Display (example)
(shows location in question)

```
$00024 E471
DUP COIL REF
```

Syntax OK display

```
NO DUP REFS
?
```

***NOTE**: You can use the same coil in more than one location, especially in programs containing Stage instructions and / or OROUT instructions. The Duplicate Reference check will find occurrences, even though they are acceptable.*

Run Time Edits

The DL06 Micro PLC allows you to make changes to the application program during Run Mode. These edits are not "bumpless." Instead, CPU scan is momentarily interrupted (and the outputs are maintained in their current state) until the program change is complete. This means if the output is off, it will remain off until the program change is complete. If the output is on, it will remain on.

WARNING: Only authorized personnel fully familiar with all aspects of the application should make changes to the program. Changes during Run Mode become effective immediately. Make sure you thoroughly consider the impact of any changes to minimize the risk of personal injury or damage to equipment. There are some important operational changes during Run Time Edits.

1. If there is a syntax error in the new instruction, the CPU will not enter the Run Mode.

2. If you delete an output coil reference and the output was on at the time, the output will remain on until it is forced off with a programming device.

3. Input point changes are not acknowledged during Run Time Edits, so, if you're using a high-speed operation and a critical input comes on, the CPU may not see the change.

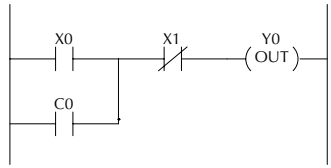
Not all instructions can be edited during a Run Time Edit session. The following list shows the instructions that can be edited.

Mnemonic	Description
TMR	Timer
TMRF	Fast timer
TMRA	Accumulating timer
TMRAF	Accumulating fast timer
CNT	Counter
UDC	Up / Down counter
SGCNT	Stage counter
STR, STRN	Store, Store not (Boolean)
AND, ANDN	And, And not (Boolean)
OR, ORN	Or, Or not (Boolean)
STRE, STRNE	Store equal, Store not equal
ANDE, ANDNE	And equal, And not equal
ORE, ORNE	Or equal, Or not equal
STR, STRN	Store greater than or equal Store less than (Comparative Boolean)
AND, ANDN	And greater than or equal And less than (Comparative Boolean)

Mnemonic	Description
OR, ORN	Or greater than or equal or less than (Comparative Boolean)
LD	Load data (constant)
LDD	Load data double (constant)
ADDD	Add data double (constant)
SUBD	Subtract data double (constant)
MUL	Multiply (constant)
DIV	Divide (constant)
CMPD	Compare accumulator (constant)
ANDD	And accumulator (constant)
ORD	Or accumulator (constant)
XORD	Exclusive or accumulator (constant)
LDF	Load discrete points to accumulator
OUTF	Output accumulator to discrete points
SHFR	Shift accumulator right
SHFL	Shift accumulator left
NCON	Numeric constant

Run Time Edit Example

We'll use the program logic shown to describe how this process works. In the example, we'll change X0 to C10. Note, the example assumes you have already placed the CPU in Run Mode.



Use the MODE key to select Run Time Edits

MODE NEXT NEXT ENT

```
*MODE CHANGE*
RUN TIME EDIT?
```

Press ENT to confirm the Run Time Edits

ENT (Note, the RUN LED on the D2–HPP Handheld starts flashing to indicate Run T ime Edits are enabled.)

```
*MODE CHANGE*
RUNTIME EDITS
```

Find the instruction you want to change (X0)

SHFT X SET A 0 SHFT FD REF FIND

```
$00000 STR X0
```

Press the arrow key to move to the X. Then enter the new contact (C10).

→ → SHFT C 2 B 1 A 0 ENT

```
RUNTIME EDIT?
STR C10
```

Press ENT to confirm the change.

ENT (Note, once you press ENT , the next address is displayed.

```
OR C0
```

Forcing I/O Points

There are many times, especially during machine startup and troubleshooting, that you need the capability to force an I/O point to be either on or off. Before you use a programming device to force any data type, it is important to understand how the DL06 CPUs process the forcing requests.

WARNING: Only authorized personnel fully familiar with all aspects of the application should make changes to the program. Make sure you thoroughly consider the impact of any changes to minimize the risk of personal injury or damage to equipment.

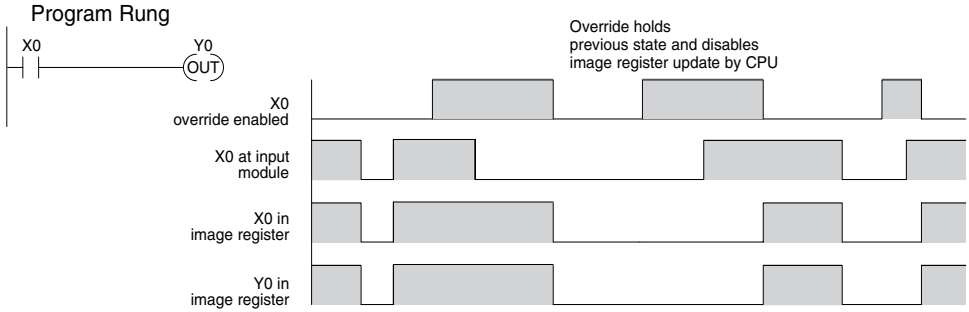
There are two types of forcing available with the DL06 CPUs. (Chapter 3 provides a detailed description of how the CPU processes each type of forcing request).

Regular Forcing: This type of forcing can temporarily change the status of a discrete bit. For example, you may want to force an input on, even though it is really off. This allows you to change the point status that was stored in the image register. This value will be valid until the image register location is written to during the next scan. This is primarily useful during testing situations when you need to force a bit on to trigger another event.

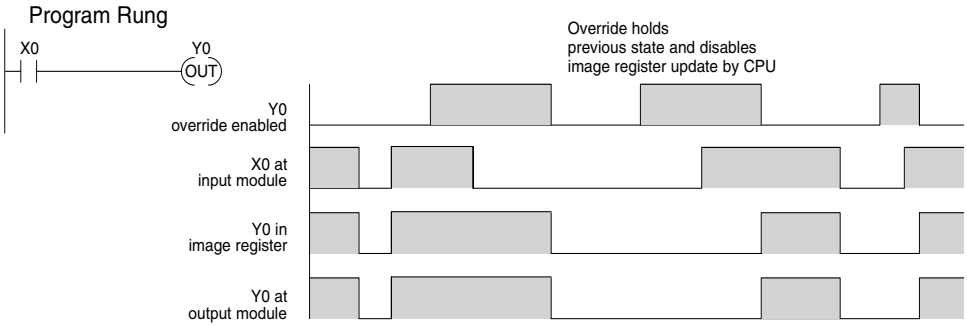
Bit Override: Bit override can be enabled on a point-by-point basis by using AUX 59 from the Handheld Programmer or by a menu option in ***Direct***SOFT. You can use Bit Override with X, Y, C, T, CT, and S data types. Bit override basically disables any changes to the discrete point by the CPU. For example, if you enable bit override for X1, and X1 is off at the time, the CPU will not change the state of X1. This means that even if X1 comes on, the CPU will not acknowledge the change. Therefore, if you used X1 in the program, it would always be evaluated as **off**, in this case. If X1 was on when the bit override was enabled, then X1 would always be evaluated as **on**.

There is an advantage available when you use the Bit Override feature. The Regular Forcing is not disabled because the Bit Override is enabled. For example, if you enabled the Bit Override for Y0 and it was off at the time, the CPU would not change the state of Y0. However, you can still use a programming device to change the status. If you use the programming device to force Y0 on, it will remain on and the CPU will not change the state of Y0. If you then force Y0 off, the CPU will maintain Y0 as off. The CPU will never update the point with the results from the application program or from the I/O update until the Bit Override is removed from the point.

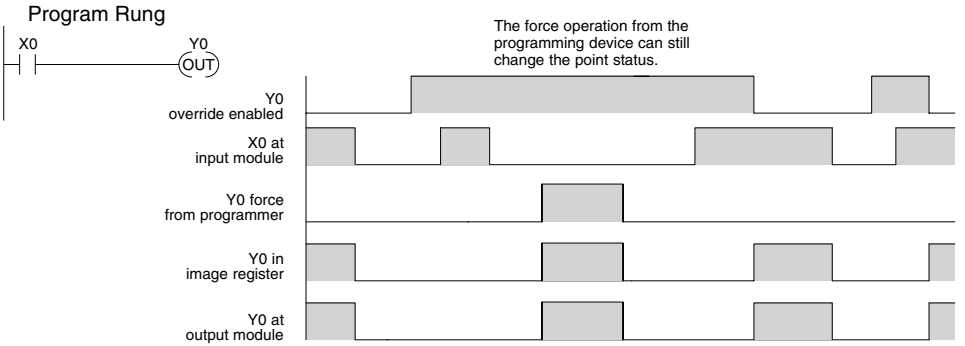
The following diagrams show how the bit override works for both input and output points. The example uses a simple rung, but the concepts are similar for any type of bit memory.



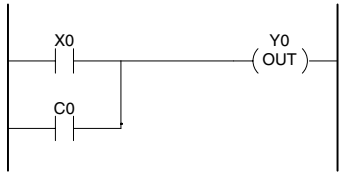
The following diagram shows how the bit override works for an output point. Notice the bit override maintains the output in the current state. If the output is on when the bit override is enabled, then the output stays on. If it is off, then the output stays off.



The following diagram shows how you can use a programming device in combination with the bit override to change the status of the point. Remember, bit override only disables CPU changes. You can still use a programming device to force the status of the point. Plus, since bit override maintains the current status, this enables true forcing. The example shown is for an output point, but you can also use the other bit data types.



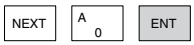
The following diagrams show a brief example of how you could use the DL06 Handheld Programmer to force an I/O point. Remember, if you are using the Bit Override feature, the CPU will retain the forced value until you disable the Bit Override or until you remove the force. The image register will not be updated with the status from the input module. Also, the solution from the application program will not be used to update the output image register. The example assumes you have already placed the CPU into Run Mode.



From a clear display, use the following keystrokes


```
16P STATUS
BIT REF   X
```

Use the PREV or NEXT keys to select the Y data type. Once the Y appears, press 0 to start at Y0.




```
   Y  10          Y    0
□□□□□□□□□□□□□□□□
```

Use arrow keys to select point, then use ON and OFF to change the status

Y2 is now on

```
   Y  10          Y    0
□□□□□□□□□□□□■□□
```

Regular Forcing with Direct Access

From a clear display, use the following keystrokes to force Y10 ON. Solid fill indicates point is on.

Solid fill indicates point is on.

```
BIT FORCE
Y10
```

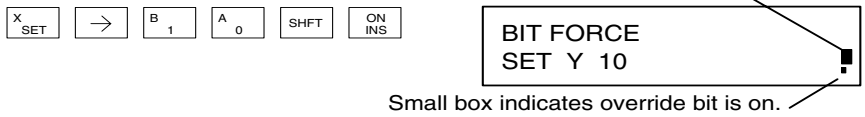
From a clear display, use the following keystrokes to force Y10 OFF. No fill indicates point is off.

No fill indicates point is off.

```
BIT FORCE
Y10
```

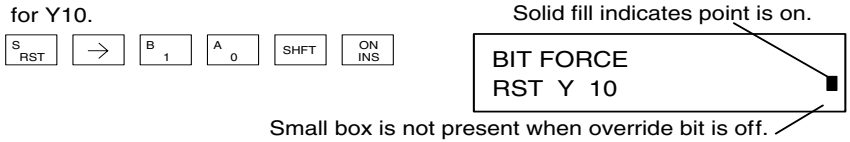
Bit Override Forcing

From a clear display, use the following keystrokes to turn on the override bit for Y10.



Note, at this point you can use the PREV and NEXT keys to move to adjacent memory locations and use the SHFT ON keys to set the override bit on.

From a clear display, use the following keystrokes to turn off the override bit for Y10. Solid fill indicates point is on.

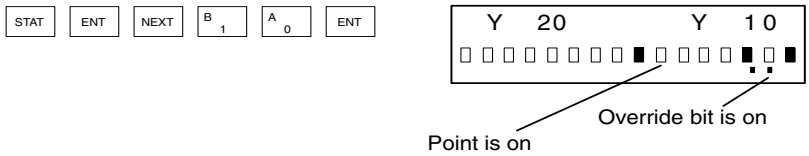


Like the example above, you can use the PREV and NEXT keys to move to adjacent memory locations and use the SHFT OFF keys to set the override bit off.

Bit Override Indicators

Override bit indicators are also shown on the handheld programmer status display. Below are the keystrokes to call the status display for Y10 – Y20.

From a clear display, use the following keystrokes to display the status of Y10 – Y20.



Reset the PLC to Factory Defaults

***NOTE**: Resetting to factory defaults will not clear any password stored in the PLC.*

Resetting a DirectLogic PLC to Factory Defaults is a two-step process. Be sure to have a verified backup of your program using "Save Project to Disk" from the File menu before performing this procedure. Please be aware that the program as well as any settings will be erased and not all settings are stored in the project. In particular you will need to write down any settings for Secondary Communications Ports and manually set the ports up after resetting the PLC to factory defaults.

Step One – While connected to the PLC with DirectSoft, go to the PLC menu and select; "Clear PLC Memory". Check the "ALL" box at the bottom of the list and press "OK".

Step Two – While connected with DirectSoft, go the PLC menu and then to the "Setup" submenu and select; "Initialize Scratch Pad" and press "Ok".

***NOTE**: All configurable communications ports will be reset to factory default state. If you are connected via Port 2 or another configurable port, you may be disconnected when this operation is complete.*

***NOTE**: Retentive ranges will be reset to the factory settings..*

***NOTE**: Manually addressed IO will be reset to factory default settings..*

The PLC has now been reset to factory defaults and you can proceed to program the PLC.

LCD Display Panel

In This Chapter...

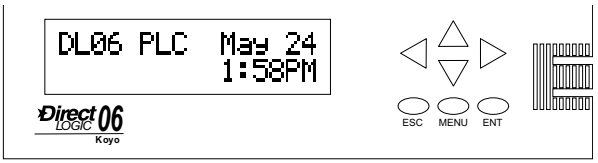
Introduction to the DL06 LCD Display Panel

The DL06 LCD Display Panel is a 16 character, two row display that mounts directly on the face of the DL06 PLC. The LCD is backlit for easy readability in most lighting situations. There are multiple ways of interacting with the LCD Display Panel:

- Built-in keypad
- LCD ladder instruction
- Using ladder instructions to write bit status changes to specified memory locations

The seven function keys on the face of the LCD Display Panel give the user access to clock and calendar setup, V-memory data values or I/O status, etc. Individuals with password authorization can:

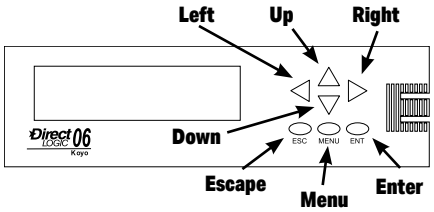
- Change clock or calendar settings or formats
- Monitor or change V-memory values (including DWord values)
- Force individual bits on or off (up to 16 per screen)
- Review error code history
- Set or change the password
- Turn the back light or buzzer on or off



The potential uses for the DL06's LCD display vary widely. An operator can change values for setting up batch processes or machine timing for manufacturing different products. Maintenance personnel can interface in the control cabinet to identify machine problems. LCD messages can be preprogrammed for process events or alarms. The LCD can satisfy these and many other needs.

Keypad

The LCD Display Panel keypad has seven keys you can use to navigate through the menu hierarchy. Each screen displayed has a specific set of active keys associated with it. All other keys (those not associated with the current screen) are inactive.

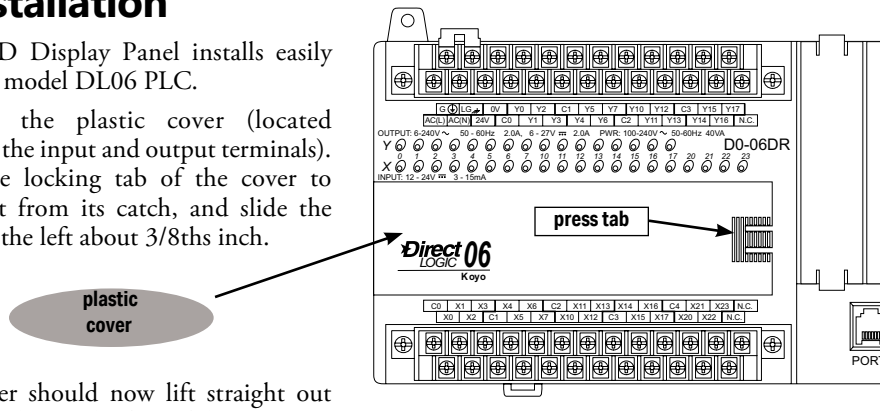


Function Keys		
Name	**Label**	**Function**
Up arrow	none	Move to selection above or increase value
Down arrow	none	Move to selection below or decrease value
Left arrow	none	Move to next digit to the left
Right arrow	none	Move to next digit to the right
Escape	ESC	Return to previous screen or next level up in the menu hierarchy
Menu	MENU	Scroll down through main menu or sub-menu selections
Enter	ENT	Enter the domain of the menu screen selected or save new value

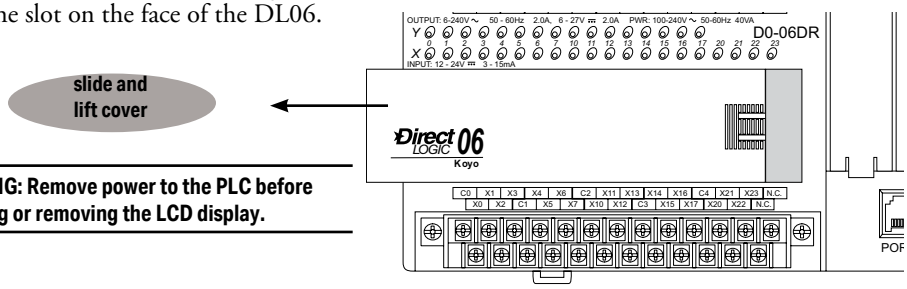
Snap-in installation

The LCD Display Panel installs easily into any model DL06 PLC.

Remove the plastic cover (located between the input and output terminals). Press the locking tab of the cover to release it from its catch, and slide the cover to the left about 3/8ths inch.

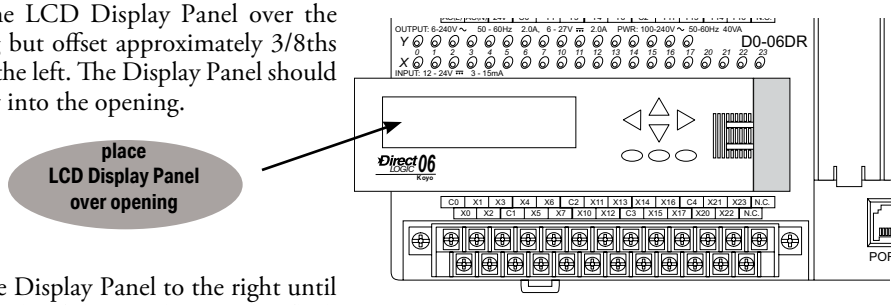


The cover should now lift straight out from the slot on the face of the DL06.

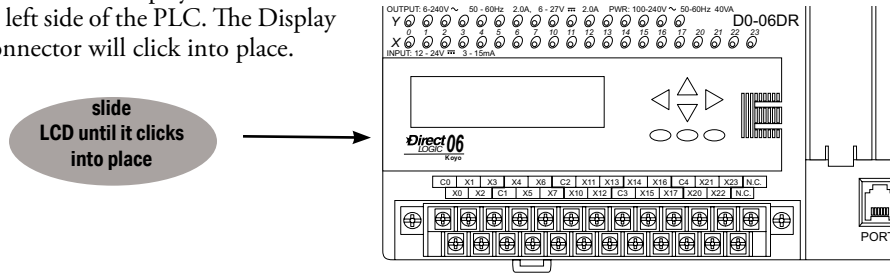


WARNING: Remove power to the PLC before installing or removing the LCD display.

Place the LCD Display Panel over the opening but offset approximately 3/8ths inch to the left. The Display Panel should fit easily into the opening.



Slide the Display Panel to the right until the left side of the Display Panel is flush with the left side of the PLC. The Display Panel connector will click into place.



Display Priority

The LCD Display Panel will show one of the following (unless power is removed from the PLC):

- Default screen (user defined or factory default)
- Menu selection
- Message from ladder program
- Error message

The built-in keypad allows you to navigate through these message displays.

On power-up the default message is normally displayed. The default message is set at the factory but can be customized by the user. Loading a custom default message is described later in this chapter.

D	L	0	6		P	L	C			M	a	y		0	8
								1	3	:	5	7	:	0	1

If a system error occurs, the error message supersedes the default message (or other current display screen), and the appropriate error code is displayed for diagnostic purposes.

D	i	a	g	n	o	s	t	i	c		E	r	r	o	r
E	4	*	*		N	O		P	R	O	G	R	A	M	

Menu Navigation

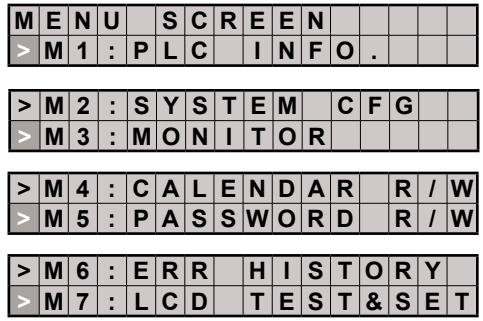
Beginning at the default screen, each time you press the MENU key the display will scroll to the next menu option. The up arrow and down arrow keys also scroll through the list of menus (in the direction indicated by the arrow), but ***you must initially press the MENU key (at the default screen) to activate the up and down arrow keys.***

There are seven built-in menus selections. Some of the menu items have sub-menus. The menus and sub-menus are described in this chapter. Each menu selection requires that you press the ENT key to view or change settings or values within the domain of that main menu selection.

Seven Menu Choices

Pressing and holding the MENU key will cause the display to scroll through the following menu options:

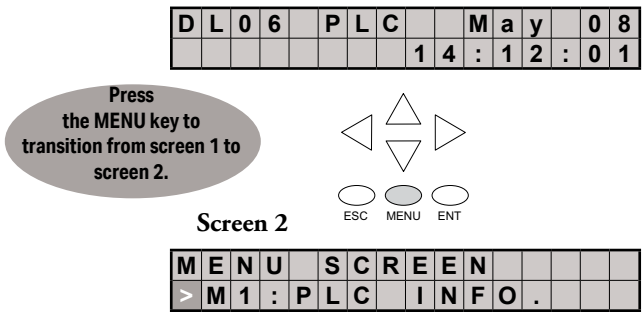
- **M1 : PLC information**
- **M2 : System configuration**
- **M3 : Monitor**
- **M4 : Calendar read/write**
- **M5 : Password read/write**
- **M6 : Error history read**
- **M7 : LCD test and set**



In this section we use illustrations of the LCD Display Panel keypad and display area to show how to navigate through the menu hierarchy. The example below shows the factory default screen as Screen 1 and the main menu entry screen as Screen 2.

The illustration of the keypad between the display screens indicates that pressing the MENU key causes a transition from Screen 1 to Screen 2. This type of representation is used throughout this section. When inside the menu hierarchy, the ESC key returns the display to the previous screen.

Screen 1 - factory default

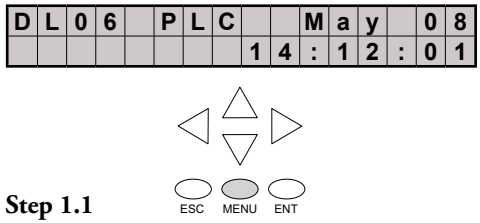


Confirm PLC Type, Firmware Revision Level, Memory Usage, Etc.

Menu 1, M1:PLC INFO.

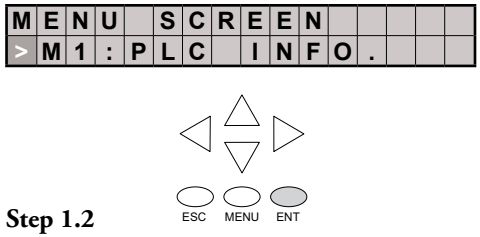
From the default screen, press the MENU key one time to arrive at the PLC INFO menu option.

Default screen

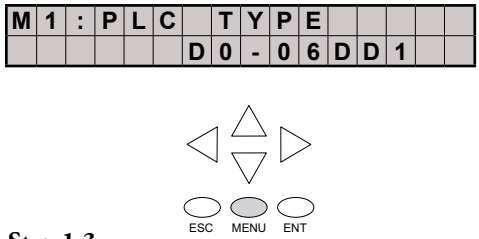


Step 1.1

Press ENT to enter this menu selection. The first screen inside the PLC INFO selection is M1:PLC TYPE. This selection displays the model number of the PLC.

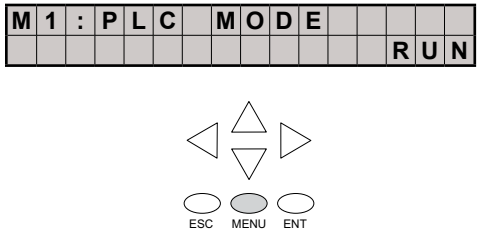


Step 1.2



Step 1.3

Press MENU again to sequence to PLC MODE. The PLC mode is either RUN, STOP (for Stop or Program Mode), TEST-STOP (for Test Stop Mode), or TEST-RUN (for Test Run Mode). You can put the DL06 in the Test Run Mode from the Test Stop Mode.

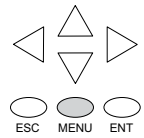


NOTE: *The menu screen examples shown in this section assume the password/lock feature is not turned on. If the password/lock feature is turned on, the user will be prompted by a message on the Display Panel to enter the password at the appropriate time. Users without password authorization will have access to a limited number of screens.*

Step 1.4

Press MENU again to sequence to FIRMWARE REV.

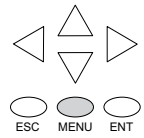
M1:FIRMWARE REV.
V1.000



Step 1.5

Press MENU again to sequence to LADDER MEMORY USED. The number of words used and the total number available in the PLC are displayed.

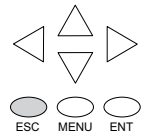
M1:LADDER MEMORY
USED 21 / 7680



Step 1.6

Press MENU again to sequence to LADDER PASSWORD, ACTIVATED OR NOT ACTIVATED. This is the last screen of the PLC INFO menu and is self-explanatory.

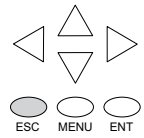
M1:LADDER PASSWD
NOT ACTIVATED



Return to Step 1.1

Press ESC to exit the M1 menu and return to the main menu.

MENU SCREEN
> M1:PLC INFO.



Default screen

Press ESC once more to return to the default screen.

DL06 PLC May 08
14:22:11

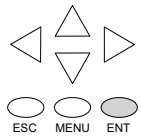
Examining Option Slot Contents

Menu 2, M2: SYSTEM CFG.

From the default screen, press MENU twice to arrive at the M2:SYSTEM CFG. (System Configuration) menu option.

Step 2.1

>	M	1	:	P	L	C		I	N	F	O	.			
>	M	2	:	S	Y	S	T	E	M		C	F	G	.	



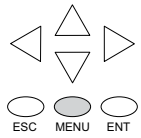
Press ENT to enter the SYSTEM CFG. menu selection.

Step 2.2

M	2	:	O	P	T	I	O	N		S	L	O	T		1
			D	0	-	D	E	V	N	E	T	S			

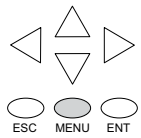
NOTE: *This is an example only and may not represent the contents of this or any option slot on your system.*

Pressing the MENU key four times will cycle through the four option slots. The model number of the option card in each slot is shown on line 2 or there is an indication that the slot is empty.



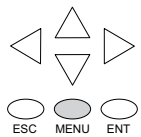
Step 2.3

M	2	:	O	P	T	I	O	N		S	L	O	T		2
		E	M	P	T	Y		I	/	O		S	L	O	T



Step 2.4

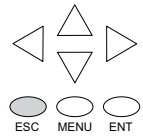
M	2	:	O	P	T	I	O	N		S	L	O	T		3
		F	0	-	0	4	A	D	-	1					



Press the ESC key twice to return to the default screen.

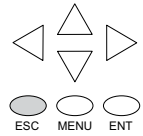
Step 2.5

M	2	:	O	P	T	I	O	N		S	L	O	T		4
		E	M	P	T	Y		I	/	O		S	L	O	T



Return to Step 2.1

>	M	1	:	P	L	C		I	N	F	O	.			
>	M	2	:	S	Y	S	T	E	M		C	F	G	.	



Return to default screen

D	L	0	6		P	L	C			M	a	y		0	8
								1	4	:	5	7	:	2	1

Monitoring and Changing Data Values

Menu 3, M3:MONITOR

From the default screen, press MENU three times to arrive at the M3:MONITOR menu option.

The M3:MONITOR sub-menu contains the data monitor and the bit monitor. The data monitor allows you to examine the contents of memory registers or pointers to determine their contents. The default format is BCD/HEX, but the format can be changed to decimal by setting bit 8 of V7742. Please refer to the DL06 Memory Map for ranges.

Data Monitor

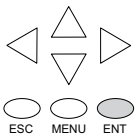
Data type = V for V-memory or P for pointer. Press MENU to change data type, or press ENT to designate the register whose data you want to view or change.

V-memory values

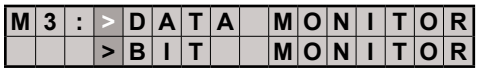
Use the right or left arrow key to move the cursor to the digit you want to change. Use the up or down arrow key to change the digit. The V-memory address is expressed as an octal number so you will not see 8's or 9's.

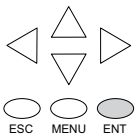
This screen allows you to view two adjacent V-memory locations in BCD format. The lower word is to the right. Pressing ENT makes it possible to change the value in the **lower word**. At this level of the menu hierarchy, you can also use the up and down arrow keys to scroll to other memory locations.

Step 3.1

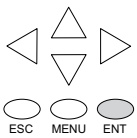


Step 3.2

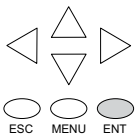




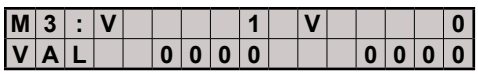
Step 3.3



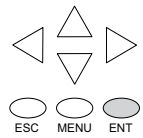
Step 3.4



Step 3.5



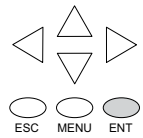
The data values on this screen will be four digits in length for BCD/HEX unless bit 8 of V7742 is set. Bit 8 of V7742 changes the data format to decimal (five digits).



Step 3.6

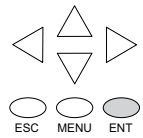
M	3	:	D	A	T	A				V				0
	C	H	G	=		0	0	0	0			0	0	0	0

Use the right or left arrow key to move the cursor to the digit you want to change. Use the up or down arrow key to move to another digit. The V-memory value is expressed as a BCD number so you will see values (in the range: 0 - F) available for each digit. The data format can be changed to decimal by setting bit 8 of V7742.



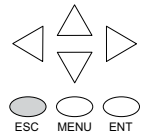
Step 3.7

M	3	:	D	A	T	A				V				0
	C	H	G	=		A	F	0	6			0	0	0	0



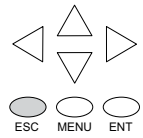
Step 3.8

M	3	:	V				1		V				0
V	A	L			0	0	0	0			A	F	0	6



Returns to Step 1.1

M	3	:	D	A	T	A		T	Y	P	E			V
A	D	D	R	E	S	S					0	0	0	0	0



Returns to default screen

Push the ESC key five (5) times to return to the default screen.

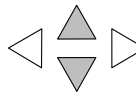
D	L	0	6		P	L	C			M	a	y		0	8
								1	5	:	0	2	:	1	3

Pointer values

Press ESC twice to return to the Step 3.3 screen with the cursor on the V, as shown. Use the up or down arrow key to change the V to P. Now, the pointer information is displayed.

Return to Step 3.3

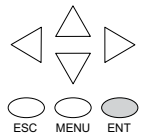
M	3	:	D	A	T	A		T	Y	P	E				V
A	D	D	R	E	S	S					0	0	0	0	0



Step 3.4a

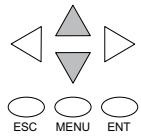
M	3	:	D	A	T	A		T	Y	P	E				P
A	D	D	R	E	S	S					0	0	0	0	0

Use the up or down arrow keys to change the value of the current digit. Use the left or right arrow keys to move from one digit to the next.



Step 3.5a

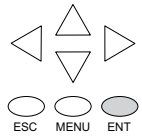
M	3	:	D	A	T	A		T	Y	P	E				P
A	D	D	R	E	S	S					0	0	0	0	0



Step 3.6a

M	3	:	D	A	T	A		T	Y	P	E				P
A	D	D	R	E	S	S					1	0	0	0	0

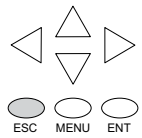
At Step 3.7a, the up and down arrow keys can be used to cycle through data words. Each time you press the up or down arrow key, the address increments or decrements by one 16-bit word (addresses are expressed in octal).



Step 3.7a

M	3	:	D	A	T	A				P	1	0	0	0	0
		(	V	0	0	0	0	0	)			2	0	0	0

Return to Step 3.3



To change from the data monitor to the bit monitor, press ESC three times to return to Step 3.2 (five times to return to the default screen).

M	3	:	>	D	A	T	A		M	O	N	I	T	O	R
			>	B	I	T			M	O	N	I	T	O	R

Bit Monitor

Bit status

From Step 3.3, press the up or down arrow key, then the ENT key. You will see one of eleven bit data types displayed. The data type that appears on the display is the last data type accessed. The address shown is also the last address accessed for that particular data type.

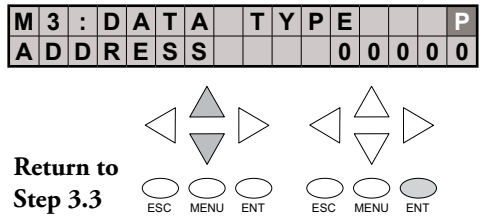
Press ENT to change the address.

Use the arrow keys to change the address as necessary.

Press ENT to view the selected bits.

Use the left and right arrow keys to select a bit whose status you want to change. Press ENT once to see the change status screen. Press ENT again to change the status from OFF to ON or ON to OFF.

Return to Step 3.3



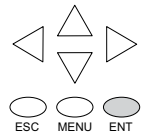
Changing Date and Time

Menu 4, M4 : CALENDAR R/W

From the default screen, press the MENU key four times to arrive at Step 4.1. Press ENT(Enter) to select M4: Calendar R/W

Step 4.1

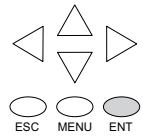
>	M	3	:	D	A	T	A		T	Y	P	E		
>	M	4	:	C	A	L	E	N	D	A	R		R	/	W



4.2 While viewing the current Date and Time Settings press ENT to change the date or time.

Step 4.2

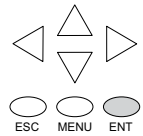
M	4	:	D	A	T	E		0	5	-	0	8	-	0	2
			T	I	M	E		0	1	:	2	1	:	2	8



4.3 Press ENT to select > Change Date/ Time.

Step 4.3

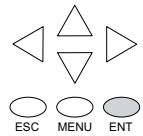
M	4	:	>	C	H	A	N	G	E		D	A	T	E	
			>	C	H	A	N	G	E		T	I	M	E	



At Step 4.4, use the up and down arrows to change the value for month, day, or year. Use the left and right arrow keys to move between the different digits in the date. After making the necessary changes using the arrow keys, press the ENT key to register the changes.

Step 4.4

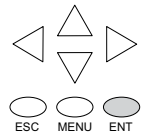
M	4	:	D	A	T	E		M	M	-	D	D	-	Y	Y
			C	H	G	=		0	5	-	0	8	-	0	2



Step 4.5

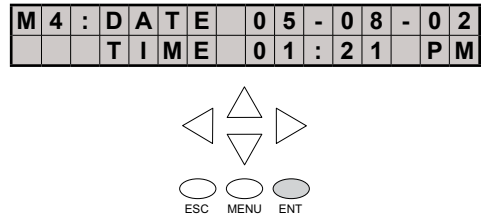
You will be asked if you want to set the date to the chosen value. Press ENT again if the date is correct. You will automatically return to Step 4.2, and the new date will be displayed.

M	4	:	D	A	T	E		M	M	-	D	D	-	Y	Y
			S	E	T	?		0	5	-	0	8	-	0	2

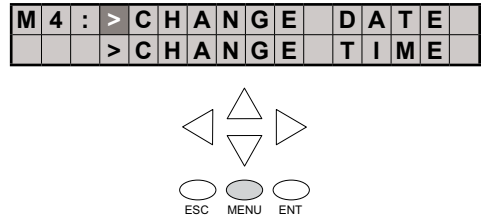


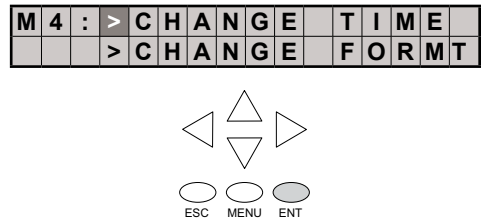
Returns to Step 4.2

In order to change the time or date/time format, press ENT again at Step 4.2.

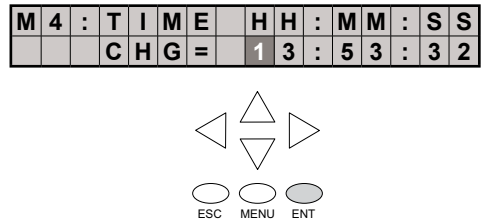


Use the up or down arrow keys or the MENU key to scroll through the sub-menu choices. At this point in our example, we will change the time setting.

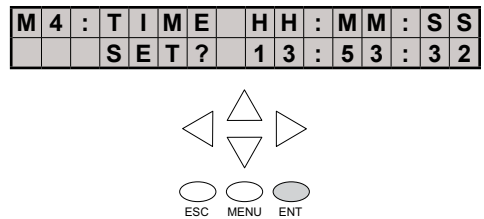




At Step 4.4, use the up and down arrows to change the value for hour, minute, or second. Use the left and right arrow keys to move between the different digits in the time. After making the necessary changes using the arrow keys, press the ENT key to register the changes.



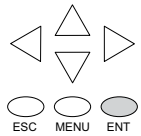
You will be asked if you want to set the date to the chosen value. Press ENT again if the date is correct. You will automatically return to Step 4.2, and the new date will be displayed.



If you want to change the format for the date or time, return to Step 4.2 and press ENT.

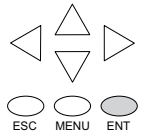
Returns to Step 4.2

M 4 : D A T E 0 5 - 0 8 - 0 2
T I M E 0 1 : 2 1 P M



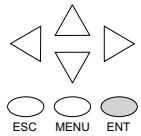
Press ENT, MENU, MENU to arrive at the menu selection for changing the date or time formats. Press ENT again to enter the format changing location.

M 4 : > C H A N G E F O R M T
> C H A N G E D A T E



Press ENT again to enter the date format changing location, or press MENU, ENT to change the time format.

M 4 : > D A T E F O R M A T
> T I M E F O R M A T



At Step 4.4, use the up and down arrow keys to scroll through the date formats. The choices are as follows:

- MM-DD-YY (US format)
- DD-MM-YY (European format)
- YY-MM-DD (Asian format)

Press the ENT key to save the format changes.

M 4 : D A T E F O R M A T
C H G = M M - D D - Y Y

M 4 : T I M E F O R M A T
C H G = H H : M M : S S

If you have chosen to make a time format change, your choices are:

- HH:MM US (12 hour 12:00 - 11:59AM/PM US format)
- HH:MM AS (12 hour 00:00 - 11:59AM/PM Asian format)
- HH:MM:SS (24 hour format)

Press the ENT key to save the format changes.

Press ESC until the default screen reappears.

Date and Time Variables and Formats		
_date:us	US format	MM/DD/YY
_date:e	European format	DD/MM/YY
_date:a	Asian format	YY/MM/DD
_time:12	12 hour format	HH:MMAM/PM
_time:24	24 hour format	HH:MM:SS

Setting Password and Locking

Menu 5, M5 : PASSWORD R/W

The LCD Display Panel has its own password protection separate from the *ladder password* protection of the PLC. An LCD Display password can be used to prevent unauthorized changes to clock and calendar setup and V-memory data values. Individuals with password authorization can change clock, calendar, V-memory values, force bits on or off, etc.

The LCD password inhibits unauthorized personnel from modifying the data in the DL06 with the LCD keypad. Even though the LCD password is locked, the user can still modify the data in the DL06 with ***Direct***SOFT or the D2-HPP. The LCD Display Panel does not support the multi-level password.

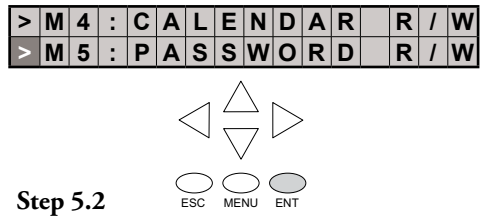
Only menu 5 on the LCD Display can modify the LCD password.

WARNING: The password protection available in *Direct*SOFT or the HPP does not prevent changes from the LCD Display Panel. To prevent changes from the LCD Display Panel, it is necessary to use the LCD password locking feature.

WARNING: The LCD password is apparently entered into the PLC, with no possibility to clear it. If you forget your LCD password, a new PLC must be purchased.

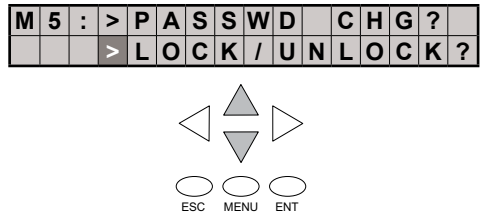
Use the MENU key to navigate to the M5 menu option. Press ENT to arrive at the display shown as Step 5.2.

Step 5.1

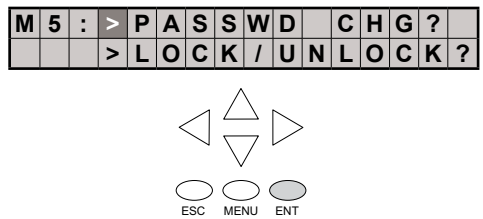


Assigning a password without locking the display allows access to all features and capabilities of the LCD.

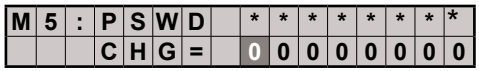
Step 5.2



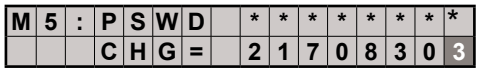
Use the up arrow or down arrow keys to toggle between PASSWD CHG? and LOCK/UNLOCK? Eight zeroes removes the password. If the password is eight zeroes, the display will not LOCK.

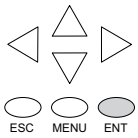


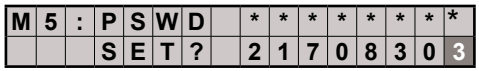
Use the up arrow or down arrow keys to scroll through number choices, and use the right arrow and left arrow keys to move from one digit position to another.

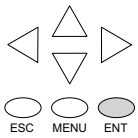


***NOTE**: It is important to record the password where it will not be forgotten and to issue the password only to qualified personnel. Full access to the LCD Display Panel gives access to change data values within the PLC.*

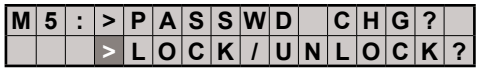


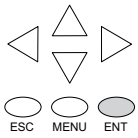






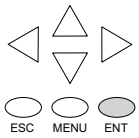
Return to Step 5.2





It is not possible to lock the display without assigning a password. It is possible to assign a password without locking the display, but doing so will not protect sensitive data.

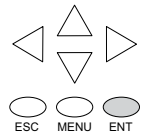
Press the ENT key at Step 5.2, and the display is now locked. If you do not wish to lock the display at this point, press ESC.



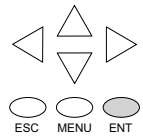
Return to Step 5.2

Before assigning a password, you can select LOCK/UNLOCK by pressing ENT at Step 5.2.

M	5	:	>	P	A	S	S	W	D		C	H	G	?	
			>	L	O	C	K	/	U	N	L	O	C	K	?



M	5	:	S	T	A	T	:	U	N	L	O	C	K	E	D
			E	N	T		T	O		L	O	C	K		



Here, the display prompts you to enter a password.

M	5	:	P	S	W	D		*	*	*	*	*	*	*	*
			L	O	C	K		0	0	0	0	0	0	0	0

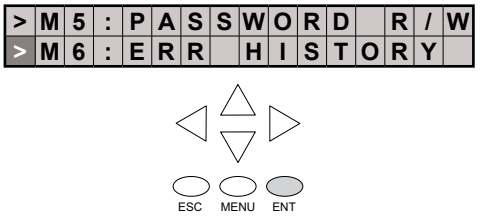
Reviewing Error History

Menu 6, M6 : ERR HISTORY

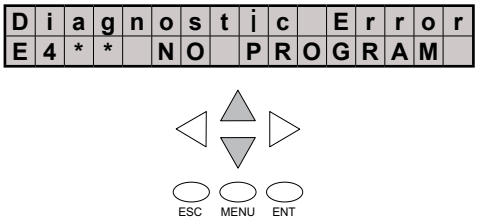
From the default screen, press the MENU key six times to arrive at Step 6.1.

Default screen

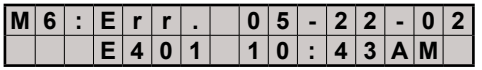
Step 6.1



The Error History screen will display **NO ERROR** if there is no record of errors. If errors have occurred, they can be identified by their Error Code. The Error Code table (see appendix B) will explain the source of the error message. The last 16 messages are displayed. Error messages are displaced when a new error message arrives



To review past error messages use the down arrow key to scroll through the historical record of error messages.



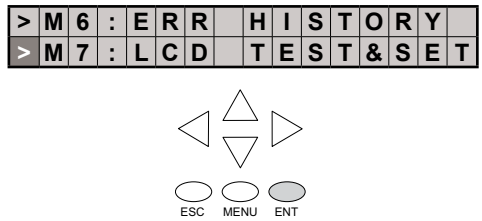
Toggle Light and Beeper, Test Keys

Menu 7, M7 : LCD TEST&SET

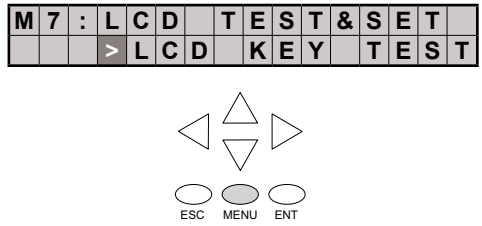
This menu selection gives you an opportunity to:

- Test each LCD key to assure that the PLC is receiving its input appropriately
- Turn the beep sound off or on
- Turn the LCD back light off or on

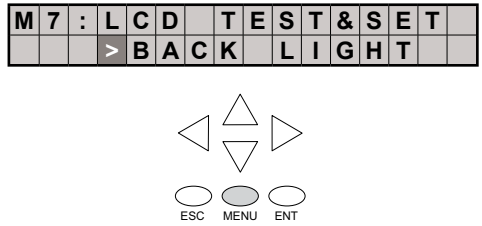
Make a menu selection by pressing the ENT key.



Press ENT to enter the LCD KEY TEST. All keys can be tested for proper function in this menu. To return to the menu, press the ESC key twice or hold the ESC key down until the menu layer reappears.



Press ENT to enter the Back light test menu.



The piezoelectric buzzer can be configured to provide pushbutton feedback.

PLC Memory Information for the LCD Display Panel

The valid memory ranges for storing text messages in the DL06 are:

V400 - V677 V1200 - V7577 V10000 - V17777

Data Format Suffixes for Embedded V-memory Data

Several data formats are available for displaying V-memory data on the LCD. The choices are shown in the table below. A colon is used to separate the embedded V-memory location from the data format suffix and modifier.

Data Format	Modifier		Example	Character Position/Content of the Output												
None (16-bit binary in HEX format)			V2000 = 0012	1	2	3	4									
			V2000	s	s	1	8									
	S	[:S]	V2000:S	1	8											
	C0	[:C0]	V2000:C0	0	0	1	8									
	0	[:0]	V2000:0	s	s	1	8									
:B (4 digit BCD)			V2000 = 0012	1	2	3	4									
		[:B]	V2000:B	0	0	1	2									
	S	[:BS]	V2000:BS	1	2											
	C0	[:BC0]	V2000:BC0	0	0	1	2									
	0	[:B0]	V2000:B0	s	s	1	2									
:D (32-bit binary)			V2000 = 0000													
			V2001 = 0001	1	2	3	4	5	6	7	8	9	10	11		
		[:D]	V2000:D	s	s	s	s	s	s	6	5	5	3	6		
	S	[:DS]	V2000:DS	6	5	5	3	6								
	C0	[:DC0]	V2000:DC0	0	0	0	0	0	0	6	5	5	3	6		
	0	[:D0]	V2000:D0	s	s	s	s	s	s	6	5	5	3	6		
:DB (8 digit BCD)			V2000 = 0000													
			V2001 = 0001	1	2	3	4	5	6	7	8					
		[:DB]	V2000:DB	0	0	0	1	0	0	0	0					
	S	[:DBS]	V2000:DBS	1	0	0	0	0								
	C0	[:DBC0]	V2000:DBC0	0	0	0	1	0	0	0	0					
	0	[:DB0]	V2000:DB0	s	s	s	1	0	0	0	0					
:R (Floating point number)			Value = 222.11111													
			V2000 = 1C72													
			V2001 = 435E	1	2	3	4	5	6	7	8	9	10	11	12	13
		[:R]	V2000:R	s	s	s	f	2	2	2	.	1	1	1	1	1
	S	[:RS]	V2000:RS	f	2	2	2	.	1	1	1	1	1			
	C0	[:RC0]	V2000:RC0	f	0	0	0	2	2	2	.	1	1	1	1	1
	0	[:R0]	V2000:R0	s	s	s	f	2	2	2	.	1	1	1	1	1
:E (Floating point number with exponent)			Value = 222.1													
			V2000 = 199A													
			V2001 = 435E	1	2	3	4	5	6	7	8	9	10	11	12	13
		[:E]	V2000:E	s	f	2	.	2	2	1	0	0	E	+	0	2
	S	[:ES]	V2000:ES	f	2	.	2	2	1	0	0	E	+	0	2	
	C0	[:EC0]	V2000:EC0	f	2	.	2	2	1	0	0	E	+	0	2	
	0	[:E0]	V2000:E0	f	2	.	2	2	1	0	0	E	+	0	2	

s = space f = plus/minus flag (plus = no symbol, minus = -)

The S, C0 and 0 modifiers alter the presentation of leading zeros and spaces. S removes leading spaces and left justifies the result. C0 replaces leading spaces with leading zeros. 0 is a modification of C0. 0 eliminates any leading zeros in the C0 format version and converts them to spaces.

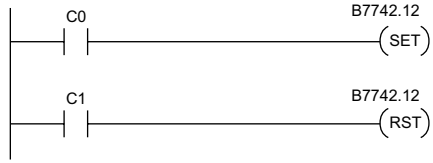
Reserved memory registers for the LCD Display Panel

Two V-memory registers are reserved for making changes to LCD functions via ladder logic. V7742 allows for bit flags to be set in the ladder program. The bit flags control such things as data formats, the back light, and the beeper. All V7742 bit flags are defined in the table on the next page.

The other reserved register is V7743. This register is used to write a customized default screen message to the LCD. A sample program for this purpose is illustrated later in this chapter.

V-memory address	Contents
V7742	**Various LCD flags** • Calendar date and time format • Default operation menu • Data format of data monitor • LCD password status flag • Key press acknowledgment buzzer on/off setting • Back light on/off setting
V7743	Default message location (writing 0 to this address returns the default message to the factory setting)

The following program segment uses the SET and RST coils to turn on and off bit 12 of V7742. When C0 is on, bit 12 is turned on. Bit 12 turns on the beeper in the LCD Display Panel. The C1 contact resets bit 12 to the off state.



V7742 bit definitions

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
V7742	*	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit 1, 0	**Date display format (default = 00)**		
	00, 11	=	Month/Day/Year (US format)
	01	=	Day/Month/Year (EU format)
	10	=	Year/Month/Day (Asian format)
Bit 3, 2	**Time display format (default = 00)**		
	00, 11	=	HH:MM:SS (24 hour format)
	01	=	HH:MM PM/AM (12 hour US format - 12:00 - 11:59)
	10	=	HH:MM PM/AM (12 hour Asian format - 00:00 - 11:59)
Bit 6 - 4	**Default menu setting (default = 000)**		
	000	=	Default menu sequence, begins menu sequence with Menu 1
	001	=	Begin menu sequence with Menu 1
	010	=	Begin menu sequence with Menu 2
	011	=	Begin menu sequence with Menu 3
	100	=	Begin menu sequence with Menu 4
	101	=	Begin menu sequence with Menu 5
	110	=	Begin menu sequence with Menu 6
	111	=	Begin menu sequence with Menu 7
Bit 8	**Data monitor format (default = 0)**		
	0	=	BCD/HEX format (0000 - FFFF)
	1	=	Decimal format (00000 - 65535)
Bit 9	**New message overwrite (default = 0)**		
	0	=	New LCD message clears both lines of previous message
	1	=	New LCD message leaves previous message, overwrites specified char. only
Bit 11	**LCD password status flag (Read only)**		
	0	=	Password unlock
	1	=	Password lock
Bit 12	**Status flag beep on/off control (default = 0)**		
	0	=	Beep OFF
	1	=	Beep ON (LCD beeps continuously during ON status of this flag)
Bit 13	**Keypad beep on/off control (default = 0)**		
	0	=	Beep OFF
	1	=	Beep ON (LCD beeps when keys are pressed)
Bit 14	**LCD back light setting flag (default = 1)**		
	0	=	Light OFF
	1	=	Light ON
Bit 15	**LCD installed status flag (Read only)**		
	0	=	LCD is not installed
	1	=	LCD is installed

Changing the Default Screen

At power-up the default screen is displayed. The default screen message is set at the factory but can be customized by the user. One method of customizing the default message uses the VPRINT instruction. The VPRINT instruction is described in Chapter 5.

Factory default message

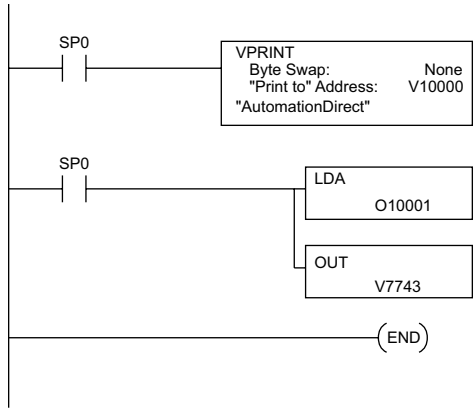
D	L	0	6		P	L	C			M	a	y		0	8
								1	4	:	2	0	:	4	9

Example program for setting the default screen message

The following program can be used to set up the default screen message. This program uses the VPRINT instruction to load ASCII text to a designated V-memory location and to embed a pointer to the current date.

The LDA and OUT instructions are used to point to the V-memory location (+1) where the text is located. The memory location V7743 is reserved for the pointer to the default message.

***NOTE**: The VPRINT instruction adds a one word (2 bytes) non-printing header to the text. For this reason, the LDA instruction points to the V-memory location V10001 rather than V10000.*



V10000	00h	16h
V10001	u	A
V10002	o	t
V10003	a	m
V10004	i	t
V10005	n	o
V10006	i	D
V10007	e	r
V10010	t	c
V10011		
V10012		
V10013		
V10014		
V10015		
V10016		
V10017		
V10020		

After running this program, press MENU, then ESC or cycle power. The new default message should look as indicated. See Menu 4 instructions for changing date and time information.

A	u	t	o	m	a	t	i	o	n	D	i	r	e	c	t

***NOTE**: It is possible to return to the factory default screen by writing 0 to V7743 and cycling power.*

DL06 LCD Display Panel Instruction (LCD)

From the DirectSOFT project folder, use the Instruction Browser to locate the LCD instruction. When you select the LCD instruction and click OK, the LCD dialog will appear.

The LCD Display Panel instruction is inserted into the ladder program via the set-up dialog box shown to the right. The dialog is used to specify a message to be displayed on line 1 or line 2 LCD Display Panel.

S	l	u	d	g	e		P	i	t		A	l	a	r	m
E	f	f	l	u	e	n	t		O	v	e	r	f	l	o

Source of message

The text of the message can originate from one of two places. It can be input directly from the instruction as a literal text string (see figure A), or it can originate as ASCII text stored in a V-memory location (figure B). In the latter case, it is necessary to specify its beginning V-memory location and length within the dialog box.

Display text strings can include embedded data. Any V-memory value or date and time settings can be embedded in the displayed text.

figure A

LCD
Line Number: K1
"Sludge Pit Alarm"

figure B

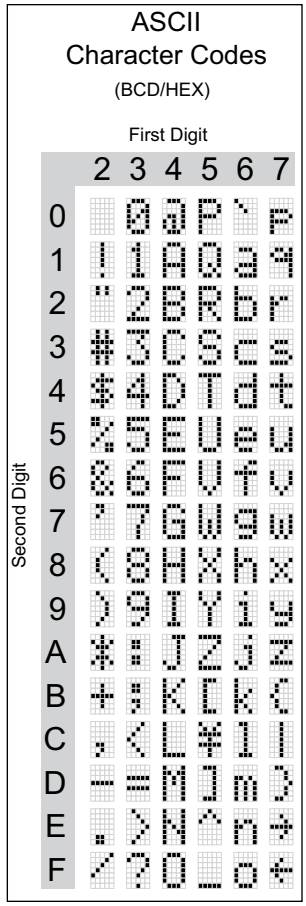
LCD
Line Number: Kn
Starting V Memory Address: A aaa
Number of Characters:

***NOTE**: The LCD Display Panel instruction is supported by DirectSOFT, Ver. 5 or later. It is not supported by the D2-HPP handheld programmer.*

ASCII Character Codes

ASCII characters can be written directly to V-memory locations and then displayed using the LCD instruction. The table to the right shows the two-digit BCD/HEX code for each character available for display.

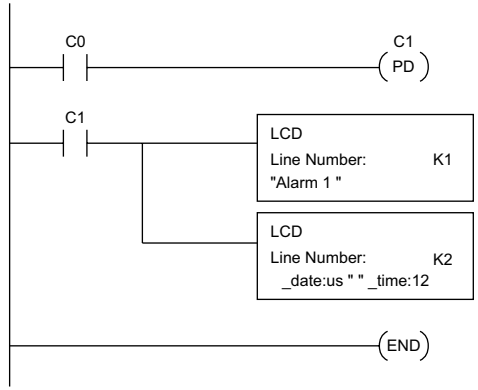
Example:
To display an upper case A, write 41 HEX to the memory location identified by the LCD instruction.



Example program: alarm with embedded date/time stamp

The following program will display the message "Alarm " and the time on line K1 of the display screen with the date on line K2.

The one-shot, or positive differential (PDd), is used so that the message displays but does not block other messages or menu options. Pressing MENU or ESC will cause the alarm message text to disappear.



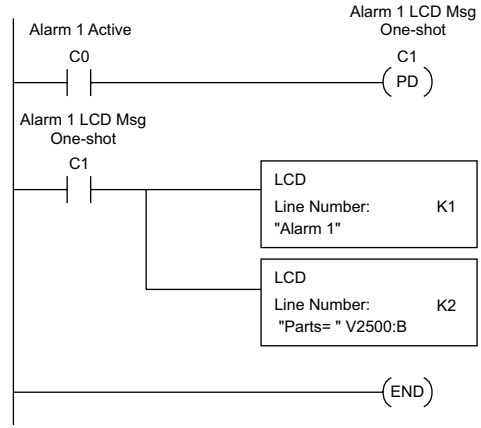
A	l	a	r	m		1								
0	5	/	0	8	/	0	4			5	:	2	3	P	M

Example program: alarm with embedded V-memory data

In this program example, the alarm notification text is displayed along with the contents of V2500. The suffix "B" is added to the memory location (V2500:B) to cause the data to be displayed as a BCD number.

In the first example, the alarm text is loaded directly via the LCD instruction. In the second example, the alarm text is loaded into V-memory and the LCD instruction is used to point to that text.

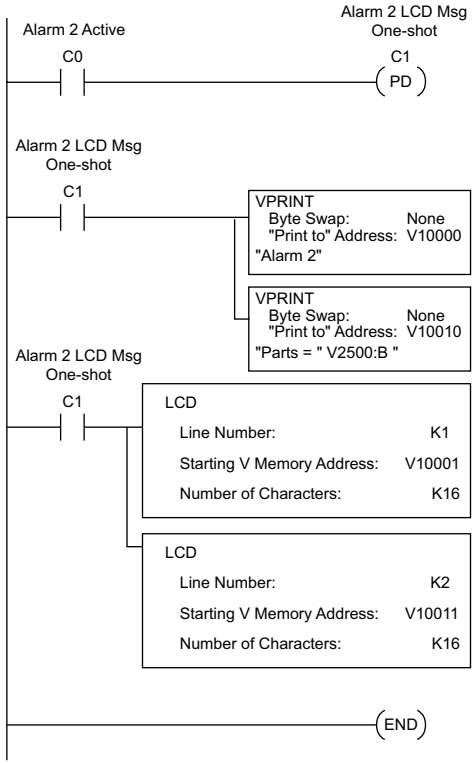
***NOTE**: When using the LCD instruction to display V2000:R, there is a limit of three characters of text because V2000:R uses 13 characters.*



A	l	a	r	m		1								
P	a	r	t	s		=		2	4	3	7			

Example program: alarm text from V-memory with embedded V-memory data

This program example uses the VPRINT instruction to write ASCII text (in the appropriate character sequence) to V10000 and V10010. The LCD instruction is used as a pointer to the V-memory location where the text for each line of the display resides.



A	l	a	r	m		2								
P	a	r	t	s		=		3	5	8	9			

AUXILIARY FUNCTIONS

In This Appendix...

Introduction

Purpose of Auxiliary Functions

Many CPU setup tasks involve the use of Auxiliary (AUX) Functions. The AUX Functions perform many different operations, including clearing ladder memory, displaying the scan time, and copying programs to EEPROM in the handheld programmer. They are divided into categories that affect different system resources. You can access the AUX Functions from *Direct*SOFT or from the D2–HPP Handheld Programmer. The manuals for those products provide step-by-step procedures for accessing the AUX Functions. Some of these AUX Functions are designed specifically for the Handheld Programmer setup, so they will not be needed (or available) with the *Direct*SOFT package. Even though this Appendix provides many examples of how the AUX functions operate, you should supplement this information with the documentation for your choice of programming device.

***NOTE**: The Handheld Programmer may have additional AUX functions that are not supported with the DL06 PLCs.*

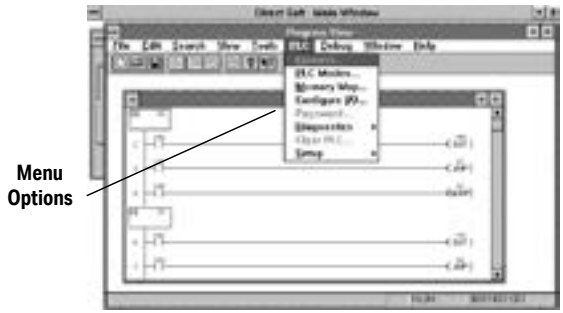
AUX Function and Description		DL06
AUX 2* — RLL Operations		
21	Check Program	*
22	Change Reference	*
23	Clear Ladder Range	*
24	Clear All Ladders	*
AUX 3* — V-Memory Operations		
31	Clear V Memory	*
AUX 4* — I/O Configuration		
41	Show I/O Configuration	*
AUX 5* — CPU Configuration		
51	Modify Program Name	*
53	Display Scan Time	*
54	Initialize Scratchpad	*
55	Set Watchdog Timer	*
56	Set Communication Port 2	*
57	Set Retentive Ranges	*
58	Test Operations	*
59	Override Setup	*
5B	HSIO Interface Configuration	*
5D	Scan Control Setup	*

AUX Function and Description		DL06
AUX 6* — Handheld Programmer Configuration		
61	Show Revision Numbers	*
62	Beeper On / Off	HP
65	Run Self Diagnostics	HP
AUX 7* — EEPROM Operations		
71	Copy CPU memory to HPP EEPROM	HP
72	Write HPP EEPROM to CPU	HP
73	Compare CPU to HPP EEPROM	HP
74	Blank Check (HPP EEPROM)	HP
75	Erase HPP EEPROM	HP
76	Show EEPROM Type (CPU and HPP)	HP
AUX 8* — Password Operations		
81	Modify Password	*
82	Unlock CPU	*
83	Lock CPU	*

***NOTE:** * - Supported; HP - Handheld Programmer function*

Accessing AUX Functions via DirectSOFT

DirectSOFT provides various menu options during both online and offline programming. Some of the AUX functions are only available during online programming, some only during offline programming, and some during both online and offline programming. The following diagram shows an example of the PLC operations menu available within DirectSOFT.



Accessing AUX Functions via the Handheld Programmer

You can also access the AUX functions by using a Handheld Programmer. Plus, remember some of the AUX functions are only available from the Handheld. Sometimes the AUX name or description cannot fit on one display. If you want to see the complete description, just press the arrow keys to scroll left and right. Also, depending on the current display, you may have to press CLR more than once.

CLR AUX

```
AUX FUNCTION SELECTION
AUX 2* RLL OPERATIONS
```

Use NXT or PREV to cycle through the menus

NEXT

```
AUX FUNCTION SELECTION
AUX 3* V OPERATIONS
```

Press ENT to select sub-menus

ENT

```
AUX 3* V OPERATIONS
AUX 31 CLR V MEMORY
```

You can also enter the exact AUX number to go straight to the sub-menu.

Enter the AUX number directly

CLR D 3 B 1 AUX

```
AUX 3* V OPERATIONS
AUX 31 CLR V MEMORY
```

AUX 2* — RLL Operations

RLL Operations auxiliary functions allow you to perform various operations on the ladder program.

AUX 21 Check Program

Both the Handheld and *Direct*SOFT automatically check for errors during program entry. However, there may be occasions when you want to check a program that has already been in the CPU. Two types of checks are available:

- Syntax
- Duplicate References

The Syntax check will find a wide variety of programming errors, such as missing END statements. If you perform this check and get an error, see Appendix B for a complete listing of programming error codes. Correct the problem and then continue running the Syntax check until the message **NO SYNTAX ERROR** appears.

Use the Duplicate Reference check to verify you have not used the same output coil reference more than once. Note, this AUX function will also find the same outputs even if they have been used with the OROUT instruction, which is perfectly acceptable.

This AUX function is available from the PLC Diagnostics sub-menu in *Direct*SOFT.

AUX 22 Change Reference

There will probably be times when you need to change an I/O address reference or control relay reference. AUX 22 allows you to quickly and easily change all occurrences, (within an address range), of a specific instruction. For example, you can replace every instance of X5 with X10.

AUX 23 Clear Ladder Range

There have been many times when we've taken existing programs and added or removed certain portions to solve new application problems. By using AUX 23, you can select and delete a portion of the program. *Direct*SOFT does not have a menu option for this AUX function, but you can just select the appropriate portion of the program and cut it with the editing tools.

AUX 24 Clear Ladders

AUX 24 clears the entire program from CPU memory. Before you enter a new program, you should always clear ladder memory. This AUX function is available on the PLC/Clear PLC sub-menu within *Direct*SOFT.

AUX 3* — V-memory Operations

AUX 31 Clear V-memory

AUX 31 clears all the information from the V-memory locations available for general use. This AUX function is available on the PLC/Clear PLC sub-menu within *Direct*SOFT.

AUX 4* — I/O Configuration

AUX 41 Show I/O Configuration

This AUX function allows you to display the current I/O configuration on the DL06. Both the Handheld Programmer and *Direct*SOFT will show the I/O configuration.

AUX 5* — CPU Configuration

The following auxiliary AUX functions allow you to setup, view, or change the CPU configuration.

AUX 51 Modify Program Name

DL06 PLCs can use a program name for the CPU program or a program stored on EEPROM in the Handheld Programmer. (Note, you cannot have multiple programs stored on the EEPROM.) The program name can be up to eight characters in length and can use any of the available characters (A–Z, 0–9). AUX 51 allows you to enter a program name. You can also perform this operation from within *Direct*SOFT by using the PLC/Setup sub-menu. Once you've entered a program name, you can only clear the name by using AUX 54 to reset the system memory. Make sure you understand the possible effects of AUX 54 before you use it!

AUX 53 Display Scan Time

AUX 53 displays the current, minimum, and maximum scan times. The minimum and maximum times are the ones that have occurred since the last Program Mode to Run Mode transition. You can also perform this operation from within DirectSOFT by using the PLC/Diagnostics sub-menu.

AUX 54 Initialize Scratchpad

The CPU maintains system parameters in a memory area often referred to as the ***scratchpad***. In some cases, you may make changes to the system setup that will be stored in system memory. For example, if you specify a range of Control Relays (CRs) as retentive, these changes are stored.

NOTE: *You may never have to use this feature unless you have made changes that affect system memory. Usually, you'll only need to initialize the system memory if you are changing programs and the old program required a special system setup. You can usually change from program to program without ever initializing system memory.*

AUX 54 resets the system memory to the default values. You can also perform this operation from within *Direct*SOFT by using the PLC/Setup sub-menu.

AUX 55 Set Watchdog Timer

DL06 PLCs have a watchdog timer that is used to monitor the scan time. The default value set from the factory is 200 ms. If the scan time exceeds the watchdog time limit, the CPU automatically leaves RUN mode and enters PGM mode. The Handheld displays the message, **E003 S/W TIMEOUT,** when the scan overrun occurs.

Use AUX 55 to increase or decrease the watchdog timer value. You can also perform this operation from within *Direct*SOFT by using the PLC/Setup sub-menu.

AUX 56 CPU Network Address

Since the DL06 CPU has an additional communication port, you can use the Handheld to set the network address for port 2 and the port communication parameters. The default settings are:

- Station address 1
- HEX mode
- Odd parity

You can use this port with either the Handheld Programmer, DirectSOFT, or, as a communication port for *Direct*NET and MODBUS. Refer to DirectNET and MODBUS manuals for additional information about communication settings required for network operation.

NOTE: *You will only need to use this procedure if you have port 2 connected to a network. Otherwise, the default settings will work fine.*

Use AUX 56 to set the network address and communication parameters. You can also perform this operation from within *Direct*SOFT by using the PLC/Setup sub-menu.

AUX 57 Set Retentive Ranges

DL06 CPUs provide certain ranges of retentive memory by default. Some of the retentive memory locations are backed up by a super-capacitor, and others are in non-volatile FLASH memory. The FLASH memory locations are V7400 to V7577 (**may be non-volatile if MOV instruction is used).** The default ranges are suitable for many applications, but you can change them if your application requires additional retentive ranges or no retentive ranges at all. The default settings are:

Memory Area	DL06	
	Default Range	Available Range
Control Relays	C1000 – C1777	C0 – C1777
V-memory	V400 – V37777	V0 – V37777
Timers	None by default	T0 – T377
Counters	CT0 – CT177	CT0 – CT177
Stages	None by default	S0 – S1777

Use AUX 57 to change the retentive ranges. You can also perform this operation from within *Direct*SOFT by using the PLC/Setup sub-menu.

WARNING: The DL06 CPUs have optional battery-backed RAM which is set as retentive. The super-capacitor will retain the values in the event of a power loss, but only up to 3 weeks. The retention time may be as short as 4 1/2 days in 60° C operating temperature.

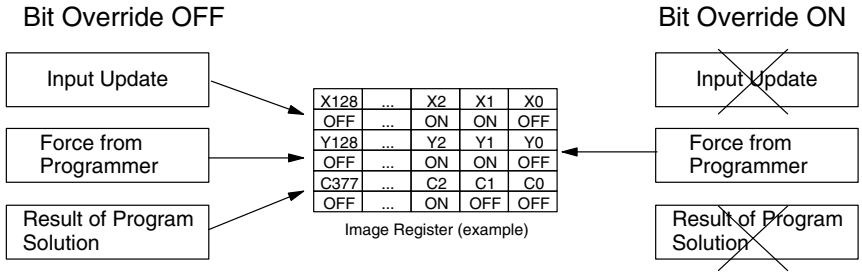
AUX 58 Test Operations

AUX 58 is used to override the output disable function of the Pause instruction. Use AUX 58 to program a single output or a range of outputs which will operate normally even when those points are within the scope of the pause instruction.

AUX 59 Bit Override

Bit override can be enabled on a point-by-point basis by using AUX 59 from the Handheld Programmer or by a menu option from within *Direct*SOFT. Bit override basically disables any changes to the discrete point by the CPU. For example, if you enable bit override for X1, and X1 is off at the time, then the CPU will not change the state of X1. This means that even if X1 comes on, the CPU will not acknowledge the change. So, if you used X1 in the program, it would always be evaluated as **OFF** in this case. Of course, if X1 was on when the bit override was enabled, then X1 would always be evaluated as **ON**.

There is an advantage available when you use the bit override feature. The regular forcing is not disabled because the bit override is enabled. For example, if you enabled the Bit Override for Y0 and it was off at the time, then the CPU would not change the state of Y0. However, you can still use a programming device to change the status. Now, if you use the programming device to force Y0 on, it will remain on and the CPU will not change the state of Y0. If you then force Y0 off, the CPU will maintain Y0 as off. The CPU will never update the point with the results from the application program or from the I/O update until the bit override is removed from the point. The following diagram shows a brief overview of the bit override feature. Notice the CPU does not update the Image Register when bit override is enabled.



AUX 5B Counter Interface Configuration

AUX 5B is used with the High-Speed I/O (HSIO) function to select the configuration. You can choose the type of counter, set the counter parameters, etc. See Chapter 3 for a complete description of how to select the various counter features.

AUX 5D Select PLC Scan Mode

The DL06 CPU has two program scan modes, fixed and variable. In fixed mode, the scan time is lengthened to the time you specify (in milliseconds). If the actual scan time is longer than the fixed scan time, then the error code **E504 BAD REF/VAL** is displayed. In variable scan mode, the CPU begins each scan as soon as the previous scan activities are complete.

AUX 6* — Handheld Programmer Configuration

The following auxiliary functions allow you to setup, view, or change the Handheld Programmer configuration.

AUX 61 Show Revision Numbers

As with most industrial control products, there are cases when additional features and enhancements are made. Sometimes these new features only work with certain releases of firmware. By using AUX 61 you can quickly view the CPU and Handheld Programmer firmware revision numbers. This information (for the CPU) is also available from within *Direct*SOFT from the PLC/Diagnostics sub-menu.

AUX 62 Beeper On/Off

The Handheld has a beeper that provides confirmation of keystrokes. You can use Auxiliary (AUX) Function 62 to turn off the beeper.

AUX 65 Run Self Diagnostics

If you think the Handheld Programmer is not operating correctly, you can use AUX 65 to run a self diagnostics program. You can check the following items.

- Keypad
- Display
- LEDs and Backlight
- Handheld Programmer EEPROM check

AUX 7* — EEPROM Operations

The following auxiliary functions allow you to move the ladder program from one area to another and perform other program maintenance tasks.

Transferable Memory Areas

Many of these AUX functions allow you to copy different areas of memory to and from the CPU and handheld programmer. The following table shows the areas that may be mentioned.

Option and Memory Type	DL06 Default Range
1:PGM — Program	$00000 – $02047
2:V — V-memory	$00000 – $07777
3:SYS — System	Non-selectable copies system parameters
4:etc (All)— Program, System and non-volatile V — V-memory only	Non-selectable

AUX 71 CPU to HPP EEPROM

AUX 71 copies information from the CPU memory to an EEPROM installed in the Handheld Programmer. You can copy different portions of EEPROM (HP) memory to the CPU memory as shown in the previous table.

AUX 72 HPP EEPROM to CPU

AUX 72 copies information from the EEPROM installed in the Handheld Programmer to CPU memory in the DL06. You can copy different portions of EEPROM (HP) memory to the CPU memory as shown in the previous table.

AUX 73 Compare HPP EEPROM to CPU

AUX 73 compares the program in the Handheld programmer (EEPROM) with the CPU program. You can compare different types of information as shown previously.

AUX 74 HPP EEPROM Blank Check

AUX 74 allows you to check the EEPROM in the handheld programmer to make sure it is blank. It's a good idea to use this function anytime you start to copy an entire program to an EEPROM in the handheld programmer.

AUX 75 Erase HPP EEPROM

AUX 75 allows you to clear all data in the EEPROM stored in the handheld programmer. You should use this AUX function before you copy a program from the CPU.

AUX 76 Show EEPROM Type

You can use AUX 76 to quickly determine what size EEPROM is installed in the Handheld Programmer.

AUX 8* — Password Operations

There are several AUX functions available that you can use to modify or enable the CPU password. You can use these features during on-line communications with the CPU. You can also use them with an EEPROM installed in the Handheld Programmer during off-line operation. This will allow you to develop a program in the Handheld Programmer and include password protection.

- AUX 81 — Modify Password
- AUX 82 — Unlock CPU
- AUX 83 — Lock CPU

AUX 81 Modify Password

You can use AUX 81 to provide an extra measure of protection by entering a password that prevents unauthorized machine operations. The password must be an eight-character numeric (0–9) code. Once you've entered a password, you can remove it by entering all zeros (00000000). (This is the default from the factory.)

Once you've entered a password, you can lock the CPU against access. There are two ways to lock the CPU with the Handheld Programmer.

- The CPU is always locked after a power cycle (if a password is present).
- You can use AUX 82 and AUX 83 to lock and unlock the CPU.

You can also enter or modify a password from within *Direct*SOFT by using the PLC/Password sub-menu. This feature works slightly differently in *Direct*SOFT. Once you've entered a password, the CPU is automatically locked when you exit the software package. It will also be locked if the CPU is power cycled.

WARNING: Make sure you remember the password before you lock the CPU. Once the CPU is locked you cannot view, change, or erase the password. If you do not remember the password, you have to return the CPU to the factory for password removal. It is the policy of Automationdirect to clear the PLC memory along with the password.

NOTE: *The DL06 CPUs support multi-level password protection of the ladder program. This allows password protection while not locking the communication port to an operator interface. The multi-level password can be invoked by creating a password with an upper case "A" followed by seven numeric characters (e.g. A1234567).*

AUX 82 Unlock CPU

AUX 82 can be used to unlock a CPU that has been password protected. *Direct*SOFT will automatically ask you to enter the password if you attempt to communicate with a CPU that contains a password.

AUX 83 Lock CPU

AUX 83 can be used to lock a CPU that contains a password. Once the CPU is locked, you will have to enter a password to gain access. Remember, this is not necessary with *Direct*SOFT since the CPU is automatically locked whenever you exit the software package.

DL06 Error Codes

In This Appendix...

DL06 Error Codes

DL06 Error Code	Description
E001 CPU FATAL ERROR	You may possibly clear the error by power cycling the CPU. If the error returns, replace the DL06.
E003 SOFTWARE TIME-OUT	If the program scan time exceeds the time allotted to the watchdog timer, this error will occur. SP51 will be on and the error code will be stored in V7755. To correct this problem use AUX 55 to extend the time allotted to the watchdog timer.
E041 CPU BATTERY LOW	The DL06 battery is low and should be replaced. SP43 will be on and the error code will be stored in V7757. The CPU indicator will blink if the battery is less than 2.5 VDC (refer to the table on page 3-6).
E104 WRITE FAILED	A write to the DL06 was not successful. Power cycle the DL06. If the error returns, replace the DL06.
E151 BAD COMMAND	A parity error has occurred in the application program. SP44 will be on and the error code will be stored in V7755. This problem may possibly be due to electrical noise. Clear the memory and download the program again. Correct any grounding problems. If the error returns replace the Micro DL06.
E155 RAM FAILURE	A checksum error has occurred in the system RAM. SP44 will be on and the error code will be stored in V7755. This problem may possibly be due to a low battery, electrical noise or a CPU RAM failure. Clear the memory and download the program again. Correct any grounding problems. If the error returns replace the DL06.
E2** I/O MODULE FAILURE	An I/O module has failed. Run AUX42 to determine the actual error.
E202 MISSING I/O MODULE	An I/O module has failed to communicate with the DL06 or is missing from the slot. SP45 will be on and the error code will be stored in V7756. Run AUX42 to determine the slot and base location of the module reporting the error.
E210 POWER FAULT	A short duration power drop-out occurred on the main power line supplying power to the DL06.
E252 NEW I/O CFG	This error occurs when the auto configuration check is turned on in the DL06 and the actual I/O configuration has changed, either by moving modules in a base, or changing types of modules in a base. You can return the modules to the original position/types or run AUX45 to accept the new configuration. SP45 will be on and the error code will be stored in V7755.
E262 I/O OUT OF RANGE	An out of range I/O address has been encountered in the application program. Correct the invalid address in the program. SP45 will be on and the error code will be stored in V7755.
E263 CONFIGURED I/O ADDRESS OUT OF RANGE	Out of range addresses have been assigned while manually configuring the I/O. Correct the address assignments using AUX46.
E311 HP COMM ERROR 1	A request from the handheld programmer could not be processed by the DL06. Clear the error and retry the request. If the error continues replace the DL06 SP46 will be on and the error code will be stored in V7756.
E312 HP COMM ERROR 2	A data error was encountered during communications with the DL06. Clear between the two devices, replace the handheld programmer, then if necessary replace the DL06. The error code will be stored in V7756.
E313 HP COMM ERROR 3	An address error was encountered during communications with the DL06. Clear the error and retry the request. If the error continues, check the cabling between the two devices, replace the handheld programmer; then, if necessary, replace the DL06. The error code will be stored in V7756.
E316 HP COMM ERROR 6	A mode error was encountered during communications with the DL06. Clear the error and retry the request. If the error continues, replace the handheld programmer; then, if necessary, replace the DL06. The error code will be stored in V7756.
E320 HP COMM TIME-OUT	The DL06 did not respond to the handheld programmer communication request. Check to ensure cabling is correct and not defective. Power cycle the system. If the error continues, replace the DL06 first and then the handheld programmer, 00 if necessary.

DL06 Error Code	Description
E321 COMM ERROR	A data error was encountered during communication with the DL06. Check to ensure cabling is correct and not defective. Power cycle the system and, if the error continues, replace the DL06 first and then the handheld programmer, if necessary.
E4** NO PROGRAM	A syntax error exists in the application program. The most common is a missing END statement. Run AUX21 to determine which one of the E4** series of errors is being flagged. SP52 will be on and the error code will be stored in V7755.
E401 MISSING END STATEMENT	All application programs must terminate with an END statement. Enter the END statement in appropriate location in your program. SP52 will be on and the error code will be stored in V7755.
E402 MISSING LBL	A MOVMC or LDLBL instruction was used without the appropriate label. Refer to Chapter 5 for details on these instructions. SP52 will be on and the error code will be stored in V7755.
E403 MISSING RET	A subroutine in the program does not end with the RET instruction. SP52 will be on and the error code will be stored in V7755.
E404 MISSING FOR	A NEXT instruction does not have the corresponding FOR instruction. SP52 will be on and the error code will be stored in V7755.
E405 MISSING NEXT	A FOR instruction does not have the corresponding NEXT instruction. SP52 will be on and the error code will be stored in V7755.
E406 MISSING IRT	An interrupt routine in the program does not end with the IRT instruction. SP52 will be on and the error code will be stored in V7755.
E412 SBR/LBL>256	There is greater than 256 SBR or DLBL instructions in the program. This error is also returned if there is greater than 4 INT instructions used in the program. SP52 will be on and the error code will be stored in V7755.
E421 DUPLICATE STAGE REFERENCE	Two or more SG or ISG labels exist in the application program with the same number. A unique number must be allowed for each Stage and Initial Stage. SP52 will be on and the error code will be stored in V7755.
E422 DUPLICATE LBL REFERENCE	Two or more LBL instructions exist in the application program with the same number. A unique number must be allowed for each label. SP52 will be on and the error code will be stored in V7755.
E423 NESTED LOOPS	Nested loops (programming one FOR/NEXT loop inside of another) are not allowed. SP52 will be on and the error code will be stored in V7755.
E431 INVALID ISG/SG ADDRESS	An ISG or SG instruction must not be placed after the end statement (such as inside a subroutine). SP52 will be on and the error code will be stored in V7755.
E432 INVALID JUMP (GOTO) ADDRESS	A LBL that corresponds to a GOTO instruction must not be programmed after the end statement such as in a subroutine. SP52 will be on and the error code will be stored in V7755.
E433 INVALID SBR ADDRESS	An SBR must be programmed after the end statement, not in the main body of the program or in an interrupt routine. SP52 will be on and the error code will be stored in V7755.
E434 INVALID RTC ADDRESS	An RTC must be programmed after the end statement, not in the main body of the program or in an interrupt routine. SP52 will be on and the error code will be stored in V7755.
E435 INVALID RT ADDRESS	An RT must be programmed after the end statement, not in the main body of the program or in an interrupt routine. SP52 will be on and the error code will be stored in V7755.
E436 INVALID INT ADDRESS	An INT must be programmed after the end statement, not in the main body of the program. SP52 will be on and the error code will be stored in V7755.

DL06 Error Code	Description
E437 INVALID IRTC ADDRESS	An IRTC must be programmed after the end statement, not in the main body of the program. SP52 will be on and the error code will be stored in V7755.
E438 INVALID IRT ADDRESS	An IRT must be programmed after the end statement, not in the main body of the program. SP52 will be on and the error code will be stored in V7755.
E440 INVALID DATA ADDRESS	Either the DLBL instruction has been programmed in the main program area (not after the END statement), or the DLBL instruction is on a rung containing input contact(s).
E441 ACON/NCON	An ACON or NCON must be programmed after the end statement, not in the main body of the program. SP52 will be on and the error code will be stored in V7755.
E451 BAD MLS/MLR	MLS instructions must be numbered in ascending order from top to bottom.
E452 X AS COIL	An X data type is being used as a coil output.
E453 MISSING T/C	A timer or counter contact is being used where the associated timer or counter does not exist.
E454 BAD TMRA	One of the contacts is missing from a TMRA instruction.
E455 BAD CNT/UDC	One of the contacts is missing from a CNT or UDC instruction.
E456 BAD SR	One of the contacts is missing from the SR instruction.
E461 STACK OVERFLOW	More than nine levels of logic have been stored on the stack. Check the use of OR STR and AND STR instructions.
E462 STACK UNDERFLOW	An unmatched number of logic levels have been stored on the stack. Ensure the number of AND STR and OR STR instructions match the number of STR instructions.
E463 LOGIC ERROR	An STR/STRN instruction was not used to begin a rung of ladder logic.
E464 MISSING CKT	A rung of ladder logic is not terminated properly.
E471 DUPLICATE COIL REFERENCE	Two or more OUT instructions reference the same I/O point.
E472 DUPLICATE TMR REFERENCE	Two or more TMR instructions reference the same number.
E473 DUPLICATE CNT REFERENCE	Two or more CNT instructions reference the same number.
E480 INVALID CV ADDRESS	The CV instruction is used in a subroutine or program interrupt routine. The CV instruction may only be used in the main program area (before the END statement).
E481 CONFLICTING INSTRUCTION	An instruction exists between convergence stages.
E482 MAX. CV INSTRUCTIONS EXCEEDED	Number of CV instructions exceeds 17.
E483 INVALID CV JUMP ADDRESS	CVJMP has been used in a subroutine or a program interrupt routine.
E484 MISSING CV INSTRUCTION	CVJMP is not preceded by the CV instruction. A CVJMP must immediately follow the CV instruction.

DL06 Error Code	Description
E485 MISSING REQUIRED INSTRUCTION	A CV JMP instruction is not placed between the CV and the [SG, ISG, ST BLK, END BLK, END] instruction.
E486 INVALID CALL BLK ADDRESS	CALL BLK is used in a subroutine or a program interrupt routine. The CALL BLK instruction may only be used in the main program area (before the END statement).
E487 MISSING ST BLK INSTRUCTION	The CALL BLK instruction is not followed by a ST BLK instruction.
E488 INVALID ST BLK ADDRESS	The ST BLK instruction is used in a subroutine or a program interrupt. Another ST BLK instruction is used between the CALL BLK and the END BLK instructions.
E489 DUPLICATE CR REFERENCE	The control relay used for the BLK instruction is being used as an output elsewhere.
E490 MISSING SG INSTRUCTION	The BLK instruction is not immediately followed by the SG instruction.
E491 INVALID ISG INSTRUCTION ADDRESS	There is an ISG instruction between the ST BLK and END BLK instructions.
E492 INVALID END BLK ADDRESS	The END BLK instruction is used in a subroutine or a program interrupt routine. The END BLK instruction is not followed by a ST BLK instruction.
E493 MISSING END REQUIRED INSTRUCTION	A [CV, SG, ISG, ST BLK, END] instruction must immediately follow the END BLK instruction.
E494 MISSING END BLK INSTRUCTION	The ST BLK instruction is not followed by a END BLK instruction.
E499 PRINT INSTRUCTION	Invalid PRINT instruction usage. Quotations and/or spaces were not entered or entered incorrectly.
E501 BAD ENTRY	An invalid keystroke or series of keystrokes was entered into the handheld programmer.
E502 BAD ADDRESS	An invalid or out of range address was entered into the handheld programmer.
E503 BAD COMMAND	An invalid command was entered into the handheld programmer.
E504 BAD REF/VAL	An invalid value or reference number was entered with an instruction.
E505 INVALID INSTRUCTION	An invalid instruction was entered into the handheld programmer.
E506 INVALID OPERATION	An invalid operation was attempted by the handheld programmer.
E520 BAD OP–RUN	An operation which is invalid in the RUN mode was attempted by the handheld programmer.
E521 BAD OP–TRUN	An operation which is invalid in the TEST RUN mode was attempted by the handheld programmer.
E523 BAD OP–TPGM	An operation which is invalid in the TEST PROGRAM mode was attempted by the handheld programmer.
E524 BAD OP–PGM	An operation which is invalid in the PROGRAM mode was attempted by the handheld programmer.
E525 MODE SWITCH	An operation was attempted by the handheld programmer while the DL06 mode switch was in a position other than the TERM position.

DL06 Error Code	Description
E526 OFF LINE	The handheld programmer is in the OFFLINE mode. To change to the ONLINE mode use the MODE key.
E527 ON LINE	The handheld programmer is in the ON LINE mode. To change to the OFF LINE mode use the MODE key.
E528 CPU MODE	The operation attempted is not allowed during a Run Time Edit.
E540 CPU LOCKED	The DL06 has been password locked. To unlock the DL06 use AUX82 with the password.
E541 WRONG PASSWORD	The password used to unlock the DL06 with AUX82 was incorrect.
E542 PASSWORD RESET	The DL06 powered up with an invalid password and reset the password to 00000000. A password may be re-entered using AUX81.
E601 MEMORY FULL	Attempted to enter an instruction which required more memory than is available in the DL06.
E602 INSTRUCTION MISSING	A search function was performed and the instruction was not found.
E603 DATA MISSING	A search function was performed and the data was not found.
E604 REFERENCE MISSING	A search function was performed and the reference was not found.
E610 BAD I/O TYPE	The application program has referenced an I/O module as the incorrect type of module.
E620 OUT OF MEMORY	An attempt to transfer more data between the DL06 and handheld programmer than the receiving device can hold.
E621 EEPROM NOT BLANK	An attempt to write to a non-blank EEPROM in the handheld programmer was made. Erase the EEPROM and then retry the write.
E622 NO HPP EEPROM	A data transfer was attempted with no EEPROM (or possibly a faulty EEPROM) installed in the handheld programmer.
E623 SYSTEM EEPROM	A function was requested with an EEPROM in the handheld programmer which contains system information only.
E624 V-MEMORY ONLY	A function was requested with an EEPROM in the handheld programmer which contains V-memory data only.
E625 PROGRAM ONLY	A function was requested with an EEPROM in the handheld programmer which contains program data only.
E627 BAD WRITE	An attempt to write to a faulty EEPROM in the handheld programmer was made. Replace the EEPROM if necessary.
E628 EEPROM TYPE ERROR	The wrong size EEPROM is being used.
E640 COMPARE ERROR	A compare between the EEPROM handheld programmer and the DL06 was found to be in error.
E642 CHECKSUM ERROR	An error was detected while data was being transferred to the handheld programmer's EEPROM. Check cabling and retry the operation.
E650 HPP SYSTEM ERROR	A system error has occurred in the handheld programmer. Power cycle the handheld programmer. If the error returns, replace the handheld programmer.
E651 HPP ROM ERROR	A ROM error has occurred in the handheld programmer. Power cycle the handheld programmer. If the error returns, replace the handheld programmer.
E652 HPP RAM ERROR	A RAM error has occurred in the handheld programmer. Power cycle the handheld programmer. If the error returns, replace the handheld programmer.

Instruction Execution Times

In This Appendix...

Introduction

This appendix contains several tables that provide the instruction execution times for DL06 Micro PLCs. Many of the execution times depend on the type of data used with the instruction. Registers may be classified into the following types:

- Data (word) Registers
- Bit Registers

V-Memory Data Registers

Some V-memory locations are considered data registers, such as timer or counter current values. Standard user V-memory is classified as a V-memory data register. Note that you can load a bit pattern into these types of registers, even though their primary use is for data registers. The following locations are data registers:

Data Registers	DL06
Timer Current Values	V0 - V377
Counter Current Values	V1000 - V1177
User Data Words	V400 - V677 V1200 - V7377 V10000 - V17777

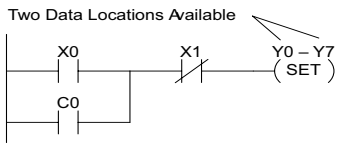
V-Memory Bit Registers

You may recall that some of the discrete points such as X, Y, C, etc., are automatically mapped into V-memory. The following bit registers contain this data:

Bit Registers	DL06
Input Points (X)	V40400 - V40437
Output Points (Y)	V40500 - V40537
Control Relays (C)	V40600 - V40677
Stages (S)	V41000 - V41077
Timer status Bits	V41100 - V41177
Counter status Bits	V41140 - V41147
Special Relays (SP)	V41200 - V41237

How to Read the Tables

Some instructions can have more than one parameter. For example, the SET instruction shown in the ladder program to the right can set a single point or a range of points.



In these cases, execution times depend on the amount and type of parameters. The execution time tables list execution times for both situations, as shown below:

SET	1st #: X, Y, C, 2nd #: X, Y, C, S (N pt)	9.2 µs 9.6 µs + 0.9 V x N
RST	1st #: X, Y, C, 2nd #: X, Y, C, S (N pt)	9.2 µs 9.6 µs + 0.9 V x N

Execution depends on numbers of locations and types of data used

Instruction Execution Times

Boolean Instructions

Boolean Instructions		DL06	
Instruction	Legal Data Types	Execute	Not Execute
STR	X, Y, C, T, CT, S, SP, GX, GY	0.67 µs	0.00 µs
STRN	X, Y, C, T, CT, S, SP, GX, GY	0.67 µs	0.0 µs
OR	X, Y, C, T, CT, S, SP, GX, GY	0.51 µs	0.51 µs
ORN	X, Y, C, T, CT, S, SP, GX, GY	0.55 µs	0.55 µs
AND	X, Y, C, T, CT, S, SP, GX, GY	0.42 µs	0.42 µs
ANDN	X, Y, C, T, CT, S, SP, GX, GY	0.51 µs	0.51 µs
ANDSTR	None	0.37 µs	0.37 µs
ORSTR	None	0.37 µs	0.37 µs
OUT	X, Y, C, GX, GY	1.82 µs	1.82 µs
OROUT	X, Y, C, GX, GY	2.09 µs	2.09 µs
NOT	None	1.04 µs	1.04 µs
SET	1st #: X, Y, C, S, 2nd #: X, Y, C, S (N pt)	9.2 µs 9.6 µs+0.9 µs x N	1.0 µs 1.1 µs
RST	1st #: X, Y, C, S, GX, GY 2nd #: X, Y, C, S (N pt), GX, GY	9.2 µs 9.6 µs+0.9 µs x N	1.0 µs 1.1 µs
	1st #: T, CT, GX, GY 2nd #: T, CT (N pt), GX, GY	25.7 µs 16.8 µs + 2.7 µs x N	1.1 µs 1.4 µs
PAUSE	1wd: Y 2wd: Y (N points)	5.6 µs 9.2 µs + 0.3 µs x N	5.4 µs 4.8 µs

Comparative Boolean Instructions

Comparative Boolean Instructions			DL06	
Instruction	Legal Data Types		Execute	Not Execute
	1st	*2nd*		
	V Data Reg.	V:Data Reg.	7.6 µs	7.6 µs
		V:Bit Reg	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	V: Bit Reg.	V:Data Reg.	7.6 µs	7.6 µs
		V:Bit Reg	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	P:Indir. (Data)	V:Data Reg	29.9 µs	29.9 µs
STRE		V:Bit Reg	29.9 µs	29.9 µs
		K:Constant	27.7 µs	27.7 µs
		P:Indir. (Data)	51.0 µs	51.0 µs
		P:Indir. (Bit)	51.0 µs	51.0 µs
	P:Indir. (Bit)	V:Data Reg	29.9 µs	29.9 µs
		V:Bit Reg	29.9 µs	29.9 µs
		K:Constant	27.7 µs	27.7 µs
		P:Indir. (Data)	51.0 µs	51.0 µs
		P:Indir. (Bit)	51.0 µs	51.0 µs
	1st	*2nd*		
	V: Data Reg.	V:Data Reg.	7.6 µs	7.6 µs
		V:Bit Reg.	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	V: Bit Reg.	V:Data Reg.	7.6 µs	7.6 µs
		V:Bit Reg.	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	P:Indir. (Data)	V:Data Reg.	30.3 µs	30.3 µs
STRNE		V:Bit Reg.	30.3 µs	30.3 µs
		K:Constant	27.4 µs	27.4 µs
		P:Indir. (Data)	51.0 µs	51.0 µs
		P:Indir. (Bit)	51.0 µs	51.0 µs
	P:Indir. (Bit)	V:Data Reg.	30.3 µs	30.3 µs
		V:Bit Reg.	30.3 µs	30.3 µs
		K:Constant	27.4 µs	27.4 µs
		P:Indir. (Data)	51.0 µs	51.0 µs
		P:Indir. (Bit)	51.0 µs	51.0 µs

Comparative Boolean Instructions (cont'd)

Comparative Boolean Instructions			DL06	
Instruction	Legal Data Types		Execute	Not Execute
ORE	*1st*	*2nd*		
	V Data Reg	V:Data Reg	7.6 µs	7.6 µs
		V:Bit Reg	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	V: Bit Reg.	V:Data Reg.	7.6 µs	7.6 µs
		V:Bit Reg	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	P:Indir. (Data)	V:Data Reg	30.3 µs	30.3 µs
		V:Bit Reg	30.3 µs	30.3 µs
		K:Constant	27.4 µs	27.4 µs
		P:Indir. (Data)	50.4 µs	50.4 µs
		P:Indir. (Bit)	50.4 µs	50.4 µs
	P:Indir. (Bit)	V:Data Reg	30.3 µs	30.3 µs
		V:Bit Reg	30.3 µs	30.3 µs
		K:Constant	27.4 µs	27.4 µs
		P:Indir. (Data)	50.4 µs	50.4 µs
		P:Indir. (Bit)	50.4 µs	50.4 µs
ORNE	*1st*	*2nd*		
	Data Reg.	V:Data Reg.	7.6 µs	7.6 µs
		V:Bit Reg.	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	V: Bit Reg.	V:Data Reg	7.6 µs	7.6 µs
		V:Bit Reg.	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	P:Indir. (Data)	V:Data Reg.	29.9 µs	29.9 µs
		V:Bit Reg.	29.9 µs	29.9 µs
		K:Constant	27.4 µs	27.4 µs
		P:Indir. (Data)	51.0 µs	51.0 µs
		P:Indir. (Bit)	51.0 µs	51.0 µs
	P:Indir. (Bit)	V:Data Reg.	29.9 µs	29.9 µs
		V:Bit Reg.	29.9 µs	29.9 µs
		K:Constant	27.4 µs	27.4 µs
		P:Indir. (Data)	51.0 µs	51.0 µs
		P:Indir. (Bit)	51.0 µs	51.0 µs

Comparative Boolean Instructions (cont'd)

Comparative Boolean Instructions			DL06	
Instruction	Legal Data Types		Execute	Not Execute
ANDE	*1st* V Data Reg.	*2nd* V:Data Reg V:Bit Reg K:Constant P:Indir. (Data) P:Indir. (Bit)	7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs	7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs
	V: Bit Reg.	V:Data Reg V:Bit Reg K:Constant P:Indir. (Data) P:Indir. (Bit)	7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs	7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs
	P:Indir. (Data)	V:Data Reg V:Bit Reg K:Constant P:Indir. (Data) P:Indir. (Bit)	29.9 µs 29.9 µs 27.4 µs 51.0 µs 51.0 µs	29.9 µs 29.9 µs 27.4 µs 51.0 µs 51.0 µs
	P:Indir. (Bit)	V:Data Reg V:Bit Reg K:Constant P:Indir. (Data) P:Indir. (Bit)	29.9 µs 29.9 µs 27.4 µs 51.0 µs 51.0 µs	29.9 µs 29.9 µs 27.4 µs 51.0 µs 51.0 µs
ANDNE	*1st* V: Data Reg.	*2nd* V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs	7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs
	V: Bit Reg.	V:Data Reg. V:Bit Reg K:Constant P:Indir. (Data) P:Indir. (Bit)	7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs	7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs
	P:Indir. (Data)	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	29.9 µs 29.9 µs 27.4 µs 51.0 µs 51.0 µs	29.9 µs 29.9 µs 27.4 µs 51.0 µs 51.0 µs
	P:Indir. (Bit)	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	29.9 µs 29.9 µs 27.4 µs 51.0 µs 51.0 µs	29.9 µs 29.9 µs 27.4 µs 51.0 µs 51.0 µs

Comparative Boolean Instructions (cont'd)

Comparative Boolean Instructions			DL06	
Instruction	Legal Data Types		Execute	Not Execute
STR	*1st* T, CT	*2nd* V:Data Reg. V:Bit Reg K:Constant P:Indir. (Data) P:Indir. (Bit)	 7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs	 7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs
	V Data Reg	V:Data Reg. V:Bit Reg K:Constant P:Indir. (Data) P:Indir. (Bit)	7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs	7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs
	V: Bit Reg.	V:Data Reg. V:Bit Reg K:Constant P:Indir. (Data) P:Indir. (Bit)	7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs	7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs
	P:Indir. (Data)	V:Data Reg. V:Bit Reg K:Constant P:Indir. (Data) P:Indir. (Bit)	29.9 µs 29.9 µs 27.4 µs 51.0 µs 51.0 µs	29.9 µs 29.9 µs 27.4 µs 51.0 µs 51.0 µs
	P:Indir. (Bit)	V:Data Reg V:Bit Reg K:Constant P:Indir. (Data) P:Indir. (Bit)	29.9 µs 29.9 µs 27.4 µs 51.0 µs 51.0 µs	29.9 µs 29.9 µs 27.4 µs 51.0 µs 51.0 µs
STRN	*1st* T, CT	*2nd* V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	 7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs	 7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs
	V: Data Reg.	V:Data Reg. V:Bit Reg K:Constant P:Indir. (Data) P:Indir. (Bit	7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs	7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs

Comparative Boolean Instructions (cont'd)

Comparative Boolean Instructions			DL06	
Instruction	Legal Data Types		Execute	Not Execute
STRN (cont.)	*1st* V: Bit Reg	*2nd* V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	 7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs	 7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs
	P:Indir. (Data)	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	29.9 µs 29.9 µs 27.4 µs 51.0 µs 51.0 µs	29.9 µs 29.9 µs 27.4 µs 51.0 µs 51.0 µs
	P:Indir. (Bit)	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	29.9 µs 29.9 µs 27.4 µs 51.0 µs 51.0 µs	29.9 µs 29.9 µs 27.4 µs 51.0 µs 51.0 µs
OR	*1st* T, CT	*2nd* V Data Reg V:Bit Reg K:Constant P:Indir. (Data) P:Indir. (Bit)	 7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs	 7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs
	V Data Reg.	V:Data Reg. V:Bit Reg K:Constant P:Indir. (Data) P:Indir. (Bit)	7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs	7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs
	V: Bit Reg.	V:Data Reg. V:Bit Reg K:Constant P:Indir. (Data) P:Indir. (Bit)	7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs	7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs
	P:Indir. (Data)	V:Data Reg V:Bit Reg K:Constant P:Indir. (Data) P:Indir. (Bit)	29.9 µs 29.9 µs 27.4 µs 51.0 µs 51.0 µs	29.9 µs 29.9 µs 27.4 µs 51.0 µs 51.0 µs
	P:Indir. (Bit)	V:Data Reg V:Bit Reg K:Constant P:Indir. (Data P:Indir. (Bit)	29.9 µs 29.9 µs 27.4 µs 51.0 µs 51.0 µs	29.9 µs 29.9 µs 27.4 µs 51.0 µs 51.0 µs

Comparative Boolean Instructions (cont'd)

Comparative Boolean Instructions			DL06	
Instruction	Legal Data Types		Execute	Not Execute
ORN	*1st* T, CT	*2nd* V:Data Reg. V:Bit Reg K:Constant P:Indir. (Data P:Indir. (Bit)	 7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs	 7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs
	V: Data Reg	V:Data Reg V:Bit Reg K:Constant P:Indir. (Data) P:Indir. (Bit)	7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs	7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs
	V: Bit Reg.	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs	7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs
	P:Indir. (Data)	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	29.9 µs 29.9 µs 27.4 µs 51.0 µs 51.0 µs	29.9 µs 29.9 µs 27.4 µs 51.0 µs 51.0 µs
	P:Indir. (Bit)	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	29.9 µs 29.9 µs 27.4 µs 51.0 µs 51.0 µs	29.9 µs 29.9 µs 27.4 µs 51.0 µs 51.0 µs

Comparative Boolean Instructions (cont'd)

Comparative Boolean Instructions			DL06	
Instruction	Legal Data Types		Execute	Not Execute
AND	*1st* T, CT	*2nd* V Data Reg. V:Bit Reg K:Constant P:Indir. (Data) P:Indir. (Bit)	 7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs	 7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs
	V: Data Reg.	V:Data Reg V:Bit Reg K:Constant P:Indir. (Data) P:Indir. (Bit)	7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs	7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs
	V: Bit Reg.	V:Data Reg. V:Bit Reg K:Constant P:Indir. (Data) P:Indir. (Bit)	7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs	7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs
	P:Indir. (Data)	V:Data Reg V:Bit Reg K:Constant P:Indir. (Data) P:Indir. (Bit)	29.9 µs 29.9 µs 27.4 µs 51.0 µs 51.0 µs	29.9 µs 29.9 µs 27.4 µs 51.0 µs 51.0 µs
	P:Indir. (Bit)	V:Data Reg V:Bit Reg K:Constant P:Indir. (Data) P:Indir. (Bit)	29.9 µs 29.9 µs 27.4 µs 51.0 µs 51.0 µs	29.9 µs 29.9 µs 27.4 µs 51.0 µs 51.0 µs

Comparative Boolean Instructions (cont'd)

Comparative Boolean Instructions			DL06	
Instruction	Legal Data Types		Execute	Not Execute
ANDN	*1st* T, CT	*2nd* V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	 7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs	 7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs
	V: Data Reg.	V:Data Reg V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs	7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs
	V: Bit Reg.	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs	7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs
	P:Indir. (Data)	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	29.9 µs 29.9 µs 27.4 µs 51.0 µs 51.0 µs	29.9 µs 29.9 µs 27.4 µs 51.0 µs 51.0 µs
	P:Indir. (Bit)	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	29.9 µs 29.9 µs 27.4 µs 51.0 µs 51.0 µs	29.9 µs 29.9 µs 27.4 µs 51.0 µs 51.0 µs

Immediate Instructions

Immediate Instructions		DL06	
Instruction	Legal Data Types	Execute	Not Execute
LDI	V	20.6 µs	1.1 µs
LDIF	1st #: Y 2nd #: K Constant	26.6 µs+0.9µs x N	1.4 µs
STRI	X	19.3 µs	19.3 µs
STRNI	X	19.4 µs	19.4 µs
ORI	X	19.1 µs	18.7 µs
ORNI	X	19.2 µs	18.9 µs
ANDI	X	18.7 µs	18.7 µs
ANDNI	X	18.8 µs	18.8 µs
OUTI	Y	25.5 µs	25.5 µs
OROUTI	Y	25.7 µs	25.7 µs
OUTIF	1st #: Y 2nd #: Y (N pt)	66.1 µs+0.9µs x N	1.4 µs
SETI	1st #: Y 2nd #: K Constant	23.1 µs, 22.8 µs+1.4µsxN	0.9 µs, 0.9 µs
RSTI	1st #: Y 2nd #: Y (N pt)	23.2 µs, 22.8 µs+1.4µsxN	0.9 µs, 0.9 µs

Bit of Word Boolean Instructions

Bit of Word Boolean Instructions		DL06	
Instruction	Legal Data Types	Execute	Not Execute
STRB	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	3.1 µs 3.1 µs 30.0 µs 30.0 µs	3.1 µs 3.1 µs 30.0 µs 30.0 µs
STRNB	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	3.0 µs 3.0 µs 29.8 µs 29.8 µs	3.0 µs 3.0 µs 29.8 µs 29.8 µs
ORB	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	2.9 µs 2.9 µs 29.9 µs 29.9 µs	2.9 µs 2.9 µs 29.9 µs 29.9 µs
ORNB	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	2.8 µs 2.8 µs 29.6 µs 29.6 µs	2.8 µs 2.8 µs 29.6 µs 29.6 µs
ANDB	V:Data Reg V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	2.8 µs 2.8 µs 29.6 µs 29.6 µs	2.8 µs 2.8 µs 29.6 µs 29.6 µs
ANDNB	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	2.7 µs 2.7 µs 29.6 µs 29.6 µs	2.7 µs 2.7 µs 29.6 µs 29.6 µs
OUTB	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	3.1 µs 3.1 µs 30.3 µs 30.3 µs	3.4 µs 3.4 µs 30.7 µs 30.7 µs
SETB	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	13.4 µs 13.4 µs 41.1 µs 41.1 µs	3.4 µs 3.4 µs 29.1 µs 29.1 µs
RSTB	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	13.5 µs 13.5 µs 41.3 µs 41.3 µs	1.4 µs 1.4 µs 29.1 µs 29.1 µs

Timer, Counter and Shift Register

Timer, Counter and Shift Register			DL06	
Instruction	Legal Data Types		Execute	Not Execute
	1st	*2nd*		
TMR	T	V Data Reg.	26.8 µs	7.3 µs
		V:Bit Reg	26.8 µs	7.3 µs
		K:Constant	20.0 µs	4.8 µs
		P:Indir. (Data)	45.6 µs	30.2 µs
		P:Indir. (Bit)	45.6 µs	30.2 µs
TMRF	T	V:Data Reg.	51.4 µs	7.3 µs
		V:Bit Reg	51.4 µs	7.3 µs
		K:Constant	48.4 µs	4.6 µs
		P:Indir. (Data)	75.9 µs	30.2 µs
		P:Indir. (Bit)	75.9 µs	30.2 µs
TMRA	T	V:Data Reg.	48.9 µs	7.3 µs
		V:Bit Reg	48.9 µs	7.3 µs
		K:Constant	45.0 µs	4.6 µs
		P:Indir. (Data)	75.9 µs	30.2 µs
		P:Indir. (Bit)	75.9 µs	30.2 µs
TMRAF	*1st*	*2nd*		
	T	V Data Reg.	54.2 µs	7.3 µs
		V:Bit Reg	54.2 µs	7.3 µs
		K:Constant	50.3 µs	4.6 µs
		P:Indir. (Data)	81.2 µs	30.2 µs
		P:Indir. (Bit)	81.2 µs	30.2 µs
CNT	CT	V:Data Reg.	25.8 µs	7.3 µs
		V:Bit Reg	25.8 µs	7.3 µs
		K:Constant	22.2 µs	4.6 µs
		P:Indir. (Data)	53.5 µs	30.2 µs
		P:Indir. (Bit)	53.5 µs	30.2 µs
SGCNT	CT	V:Data Reg.	27.3 µs	7.3 µs
		V:Bit Reg	27.3 µs	7.3 µs
		K:Constant	23.5 µs	4.6 µs
		P:Indir. (Data)	54.9 µs	30.2 µs
		P:Indir. (Bit)	54.9 µs	30.2 µs
UDC	CT	V:Data Reg	39.8 µs	7.3 µs
		V:Bit Reg	39.8 µs	7.3 µs
		K:Constant	35.4 µs	4.6 µs
		P:Indir. (Data)	67.8 µs	30.2 µs
		P:Indir. (Bit)	67.8 µs	30.2 µs
SR	C (N points to shift)		17.8 µs + 0.9 µs x N	9.8 µs

Accumulator Data Instructions

Accumulator / Stack Load and Output Data Instructions			DL06	
Instruction	Legal Data Types		Execute	Not Execute
LD	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)		11.8 µs 11.8µs 9.0 µs 33.9 µs 33.9 µs	1.0 µs 1.0 µs 1.0 µs 0.9 µs 0.9 µs
LDD	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)		12.2 µs 12.2 µs 9.0 µs 37.8 µs 37.8 µs	1.0 µs 1.0 µs 1.0 µs 0.9 µs 0.9 µs
LDF	*1st* X, Y, C, S T, CT, SP	*2nd* K:Constant	20.5 µs+0.9 µsxN	0.9 µs
LDA	O: (Octal constant for address)		10.4 µs	1.0 µs
LDR	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)		29.5 µs 29.5 µs 25.5 µs 54.9 µs 54.9 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs 1.0 µs
LDSX	K: Constant		14.6 µs	1.0 µs
LDX	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)		10.8 µs 10.8 µs 45.2 µs 45.2 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs
OUT	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)		9.3 µs 9.3 µs 35.2 µs 35.2 µs	1.0 µs 1.0 µs 0.9 µs 0.9 µs
OUTD	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)		10.2 µs 10.2 µs 35.8 µs 35.8 µs	1.0 µs 1.0 µs 0.9 µs 0.9 µs
OUTF	*1st* X, Y, C	*2nd* K:Constant	54 µs+1.0 µsxN	0.9 µs
OUTL	V:Data Reg. V:Bit Reg.		13.5 µs 13.5 µs	1.0 µs 1.0 µs
OUTM	V:Data Reg. V:Bit Reg.		13.7 µs 13.7 µs	1.0 µs 1.0 µs
OUTX	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)		17.2 µs 17.2 µs 43.4 µs 43.4 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs
POP	NONE		8.4 µs	1.0 µs

Logical Instructions

Logical (Accumulator) Instructions		DL06	
Instruction	Legal Data Types	Execute	Not Execute
AND	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	7.9 µs 7.9 µs 33.4 µs 33.4 µs	1.0 µs 1.0 µs 0.9 µs 0.9 µs
ANDD	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	8.9 µs 8.9 µs 5.7 µs 34.4 µs 34.4 µs	1.0 µs 1.0 µs 1.0 µs 0.9 µs 0.9 µs
ANDF	***1st:*** X, Y, C, S T, CT, SP, GX, GY ***2nd:*** K:Constant	21.6 µs + 0.9 µs x N	1.0 µs
ANDS	None	10.0 µs	1.0 µs
OR	V:Data Reg V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	8.1 µs 8.1 µs 33.8 µs 33.8 µs	1.0 µs 1.0 µs 0.9 µs 0.9 µs
ORD	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	9.0 µs 9.0 µs 5.8 µs 34.5 µs 34.5 µs	1.0 µs 1.0 µs 1.0 µs 0.9 µs 0.9 µs
ORF	***1st:*** X, Y, C, S T, CT, SP, GX, GY ***2nd:*** K:Constant	20.9 µs + 0.9 µs x N	1.0 µs
ORS	None	10.2 µs	1.0 µs
XOR	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	8.0 µs 8.0 µs 33.6 µs 33.6 µs	1.0 µs 1.0 µs 0.9 µs 0.9 µs
XORD	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	9.0 µs 9.0 µs 5.4 µs 34.4 µs 34.4 µs	1.0 µs 1.0 µs 1.0 µs 0.9 µs 0.9 µs
XORF	***1st:*** X, Y, C, S T, CT, SP, GX, GY ***2nd:*** K:Constant	20.9 µs + 0.9 µs x N	1.0 µs
XORS	None	10.1 µs	1.0 µs
CMP	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	9.4 µs 9.4 µs 34.9 µs 34.9 µs	1.0 µs 1.0 µs 0.9 µs 0.9 µs
CMPD	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	9.9 µs 9.9 µs 6.7 µs 35.4 µs 35.4 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs 1.0 µs

Logical Instructions (cont'd)

Logical (Accumulator) Instructions		DL06	
Instruction	Legal Data Types	Execute	Not Execute
CMPF	*1st:* X, Y, C, S T, CT, SP, GX, GY *2nd:* K:Constant	20.9 µs + 1.0 µs x N	1.0 µs
CMPR	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	42.8 µs 42.8 µs 38.4 µs 69.0 µs 69.0 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs 1.0 µs
CMPS	None	11.2 µs	1.0 µs

Math Instructions

Math Instructions (Accumulator)		DL06	
Instruction	Legal Data Types	Execute	Not Execute
ADD	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	78.4 µs 78.4 µs 101.2 µs 101.2 µs	0.9 µs 0.9 µs 0.9 µs 0.9 µs
ADDD	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Daa) P:Indir. (Bit)	83.3 µs 83.3 µs 67.7 µs 101.2 µs 101.2 µs	0.9 µs 0.9 µs 0.9 µs 0.9 µs 0.9 µs
SUB	V:Data Reg. V:Bit Reg P:Indir. (Data) P:Indir. (Bit)	77.4 µs 77.4 µs 95.1 µs 95.1 µs	0.9 µs 0.9 µs 0.9 µs 0.9 µs
SUBD	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	82.5 µs 82.5 µs 66.0 µs 99.7 µs 99.7 µs	0.9 µs 0.9 µs 0.9 µs 0.9 µs 0.9 µs
MUL	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	266.1 µs 266.1 µs 286.9 µs 290.0 µs 290.0 µs	0.9 µs 0.9 µs 0.9 µs 0.9 µs 0.9 µs
MULD	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	839.1 µs 839.1 µs 863.1 µs 863.1 µs	0.9 µs 0.9 µs 0.9 µs 0.9 µs
DIV	V:Data Reg. V:Bit Reg K:Constant P:Indir. (Data) P:Indir. (Bit)	363.9 µs 363.9 µs 384.4 µs 419.8 µs 419.8 µs	0.9 µs 0.9 µs 0.9 µs 0.9 µs 0.9 µs
DIVD	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	398.3 µs 398.3 µs 390.9 µs 390.9 µs	0.9 µs 0.9 µs 0.9 µs 0.9 µs

Math Instructions (cont'd)

Math Instructions (Accumulator)		DL06	
Instruction	Legal Data Types	Execute	Not Execute
INC	V:Data Reg V:Bit Reg P:Indir. (Data) P:Indir. (Bit)	48.5 µs 48.5 µs 74.7 µs 74.7 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs
DEC	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	47.5 µs 47.5 µs 71.5 µs 71.5 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs
INCB	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	13.2 µs 13.2 µs 38.6 µs 38.6 µs	1.0 µs 1.0 µs 0.9 µs 0.9 µs
DECB	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	13.2 µs 13.2 µs 38.0 µs 38.0 µs	1.0 µs 1.0 µs 0.9 µs 0.9 µs
ADDB	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	24.9 µs 24.9 µs 23.5 µs 51.1 µs 51.1 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs 1.0 µs
ADDBD	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	24.4 µs 24.4 µs 20.7 µs 50.7 µs 50.7 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs 1.0 µs
SUBB	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	24.7 µs 24.7 µs 23.3 µs 50.6 µs 50.6 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs 1.0 µs
SUBBD	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	24.2 µs 24.2 µs 20.2 µs 50.2 µs 50.2 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs 1.0 µs
MULB	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	10.8 µs 10.8 µs 8.2 µs 37.1 µs 37.1 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs 1.0 µs
DIVB	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	28.7 µs 28.7 µs 26.1 µs 54.9 µs 54.9 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs 1.0 µs
ADDR	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	48.1 µs 48.1 µs 41.7 µs 74.3 µs 74.3 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs 1.0 µs

Math Instructions (cont'd)

Math Instructions (Accumulator)		DL06	
Instruction	Legal Data Types	Execute	Not Execute
SUBR	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	50.1 µs 50.1 µs 58.7 µs 76.3 µs 76.3 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs 1.0 µs
MULR	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	54.2 µs 54.2 µs 42.7 µs 80.4 µs 80.4 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs 1.0 µs
DIVR	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	50.1 µs 50.1 µs 58.7 µs 76.3 µs 76.3 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs 1.0 µs
ADDF	*1st:* X, Y, C, S T, CT, SP, GX, GY *2nd:* K:Constant	109.3 µs + 0.9 µs x N	1.0 µs
SUBF	*1st:* X, Y, C, S T, CT, SP, GX, GY *2nd:* K:Constant	107.3 µs + 0.9 µs x N	1.0 µs
MULF	*1st:* X, Y, C, S T, CT, SP, GX, GY *2nd:* K:Constant	352.5 µs + 0.9 µs x N	1.0 µs
DIVF	*1st:* X, Y, C, S T, CT, SP, GX, GY *2nd:* K:Constant	477.3 µs + 0.8 µs x N	1.0 µs
ADDS	None	99.5 µs	1.0 µs
SUBS	None	97.5 µs	1.0 µs
MULS	None	342.5 µs	1.0 µs
DIVS	None	467.3 µs	1.0 µs
ADDBS	None	24.3 µs	1.0 µs
SUBBS	None	23.7 µs	1.0 µs
MULBS	None	11.7 µs	1.0 µs
DIVBS	None	29.7 µs	1.0 µs
SQRTR	None	87.9 µs	1.0 µs
SINR	None	226.8 µs	1.0 µs
COSR	None	213.1 µs	1.0 µs
TANR	None	285.5 µs	1.0 µs
ASINR	None	489.8 µs	1.0 µs
ACOSR	None	508.3 µs	1.0 µs
ATANR	None	317.1 µs	1.0 µs

Differential Instructions

Differential Instructions		DL06	
Instruction	Legal Data Types	Execute	Not Execute
PD	X, Y, C	14.4 µs	14.4 µs
STRPD	X, Y, C, S, T, CT	5.4 µs	5.4 µs
STRND	X, Y, C, S, T, CT	7.3 µs	7.3 µs
ORPD	X, Y, C, S, T, CT	6.8 µs	5.2 µs
ORND	X, Y, C, S, T, CT	7.1 µs	4.9 µs
ANDPD	X, Y, C, S, T, CT	6.8 µs	5.2 µs
ANDND	X, Y, C, S, T, CT	7.1 µs	4.9 µs

Bit Instructions

Bit Instructions (Accumulator)		DL06	
Instruction	Legal Data Types	Execute	Not Execute
SUM	None	6.7 µs	1.0 µs
SHFR	V:Data Reg. (N bits) V:Bit Reg. (N bits) K:Constant (N bits)	12.1 µs + 0.1 x N 8.4 µs + 0.1 x N	0.9 µs
SHFL	V:Data Reg. (N bits) V:Bit Reg. (N bits) K:Constant (N bits)	12.1 µs + 0.1 x N 8.4 µs + 0.1 x N	0.9 µs
ROTR	V:Data Reg. (N bits) V:Bit Reg. (N bits) K:Constant (N bits)	16.4 µs 16.4 µs 12.9 µs	1.0 µs 1.0 µs 1.0 µs
ROTL	V:Data Reg. (N bits) V:Bit Reg. (N bits) K:Constant (N bits)	16.4 µs 16.4 µs 12.7 µs	1.0 µs 1.0 µs 1.0 µs
ENCO	None	33.9 µs	0.9 µs
DECO	None	5.7 µs	1.0 µs

Number Conversion Instructions

Number Conversion Instructions (Accumulator)		DL06	
Instruction	Legal Data Types	Execute	Not Execute
BIN	None	100.2 µs	0.9 µs
BCD	None	95.2 µs	0.9 µs
INV	None	2.5 µs	1.0 µs
BCDPL	None	75.6 µs	1.0 µs
ATH	V	25.4 µs	1.0 µs
HTA	V	25.4 µs	1.0 µs
GRAY	None	110.8 µs	1.0 µs
SFLDGT	None	23.1 µs	1.0 µs
BTOR	None	18.6 µs	1.0 µs
RTOB	None	8.6 µs	1.0 µs
RADR	None	51.4 µs	1.0 µs
DEGR	None	81.5 µs	1.0 µs

Table Instructions

Table Instructions		DL06	
Instruction	Legal Data Types	Execute	Not Execute
MOV	Move V:data reg. to V:data reg Move V:bit reg. to V:data reg Move V:data reg. to V:bit reg Move V:bit reg. to V:bit reg. N=#of words	60.2 µs+9.5 x N	0.9 µs
MOVMC	Move V:data Reg to EEPROM Move V:Bit Reg to EEPROM Move from Date Label to V: Data Reg Move from Data Label to V: Bit Reg N= #of words	35 µs + 10.4 µs x N	0.9 µs
LDLBL	K	6.4 µs	1.3 µs
FILL	V: Data Reg V:Bit Reg K:Constant P:Indir. (Data) P:Indir. (bit)	29.4 µs + 8.0 µs x N 26.2 µs + 8.0 µs x N 55.1 µs + 8.0 µs x N	1.0 µs 1.0 µs 1.0 µs
FIND	V: Data Reg (N bits) V:Bit Reg. (N bits) K:Constant(N bits)	66.8 µs 66.8 µs 64.0 µs	1.0 µs 1.0 µs 1.0 µs

Table Instructions (cont'd)

Table Instructions		DL06	
Instruction	Legal Data Types	Execute	Not Execute
FDGT	V: Data Reg (N bits) V:Bit Reg. (N bits) K:Constant(N bits)	66.1 µs 66.1 µs 55.2 µs	1.0 µs 1.0 µs 1.0 µs
FINDB	V: Data Reg (N bits) V:Bit Reg. (N bits) P:Indir. (Data) P:Indir. (Bit)	210.8 µs 210.8 µs 237.0 µs 237.0 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs
TTD	V: Data Reg V:Bit Reg	66.9 µs 66.9 µs	1.0 µs 1.0 µs
RFB	V: Data Reg V:Bit Reg	66.8 µs 66.8 µs	1.0 µs 1.0 µs
STT	V: Data Reg V:Bit Reg K:Constant	67.8 µs 67.8 µs 65.0 µs	1.0 µs 1.0 µs 1.0 µs
RFT	V: Data Reg V:Bit Reg	51.1 µs 51.1 µs	1.0 µs 1.0 µs
ATT	V: Data Reg V:Bit Reg K:Constant	53.5 µs 53.5 µs 50.8 µs	1.0 µs 1.0 µs 1.0 µs
TSHFL	V: Data Reg V:Bit Reg	134.0 µs 134.0 µs	1.0 µs 1.0 µs
TSHFR	V: Data Reg V:Bit Reg	133.9 µs 133.9 µs	1.0 µs 1.0 µs
ANDMOV	V: Data Reg V:Bit Reg	80.2 µs 80.2 µs	1.0 µs 1.0 µs
ORMOV	V: Data Reg V:Bit Reg	80.4 µs 80.4 µs	1.0 µs 1.0 µs
XORMOV	V: Data Reg V:Bit Reg	80.4 µs 80.4 µs	1.0 µs 1.0 µs
SWAP	V: Data Reg V:Bit Reg	84.1 µs 84.1 µs	1.0 µs 1.0 µs
SETBIT	V: Data Reg (N bits) V:Bit Reg. (N bits)	59.5 µs 59.5 µs	1.0 µs 1.0 µs
RSTBIT	V: Data Reg (N bits) V:Bit Reg. (N bits)	59.5 µs 59.5 µs	1.0 µs 1.0 µs

CPU Control Instructions

CPU Control Instructions		DL06	
Instruction	Legal Data Types	Execute	Not Execute
NOP	None	1.1 µs	1.1 µs
END	None	24.0 µs	24.0 µs
STOP	None	10.0 µs	1.1 µs
RSTWT	None	5.9 µs	2.2 µs

Program Control Instructions

Program Control Instructions		DL06	
Instruction	Legal Data Types	Execute	Not Execute
GOTO	K	5.1 µs	4.8 µs
LBL	K	5.7 µs	0.0 µs
FOR	V, K	125.9 µs	14.5 µs
NEXT	None	64.4 µs	64.4 µs
GTS	K	27.5 µs	14.8 µs
SBR	K	1.5 µs	1.5 µs
RTC	None	25.7 µs	12.1 µs
RT	None	21.2 µs	21.2 µs
MLS	K	(1–7) 35.2 µs	35.2 µs
MLR	K	(0–7) 30.9 µs	30.9 µs

Interrupt Instructions

Interrupt Instructions		DL06	
Instruction	Legal Data Types	Execute	Not Execute
ENI	None	24.2 µs	2.7 µs
DISI	None	9.4 µs	2.3 µs
INT	O(0,1)	7.5 µs	–
IRTC	None	0.9 µs	1.3 µs
IRT	None	6.6 µs	–

Network Instructions

Network Instructions		DL06	
Instruction	Legal Data Types	Execute	Not Execute
RX	X, Y, C, T, CT, SP, S, $ V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	852.0 µs 852.0 µs 852.0 µs 868.2 µs 868.2 µs	4.4 µs 4.4 µs 4.4 µs 4.2 µs 4.2 µs
WX	X, Y, C, T, CT, SP, S, $ V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	1614.0 µs 1614.0 µs 1614.0 µs 1630.0 µs 1630.0 µs	4.4 µs 4.4 µs 4.4 µs 4.4 µs 4.4 µs

Intelligent I/O Instructions

Network Instructions		DL06	
Instruction	Legal Data Types	Execute	Not Execute
RD	V:Data Reg. V:Bit Reg.	385.7 µs 385.7 µs	1.2 µs 1.2 µs
WT	V:Data Reg. V:Bit Reg.	385.6 µs 385.6 µs	1.2 µs 1.2 µs

Message Instructions

Message Instructions		DL06	
Instruction	Legal Data Types	Execute	Not Execute
FAULT	V:Data Reg. V:Bit Reg. K:Constant	65.0 µs 65.0 µs 204.7 µs	4.4 µs 4.4 µs 4.4 µs
DLBL	K	–	–
NCON	K	–	–
ACON	A	–	–
PRINT	ASCII	631.0 µs	3.6 µs

RLLplus Instructions

RLL^{PLUS} Instructions		DL06	
Instruction	Legal Data Types	Execute	Not Execute
ISG	S	44.0 µs	41.1 µs
SG	S	44.0 µs	41.1 µs
JMP	S	76.0 µs	9.3 µs
NJMP	S	77.4 µs	9.3 µs
CV	S	42.1 µs	27.5 µs
CVJMP	S	89.5 µs	17.6 µs
BCALL	C	22.1 µs	22.6 µs
BLK	C	17.1 µs	14.6 µs
BEND	None	8.7 µs	0.0 µs

Drum Instructions

Drum Instructions		DL06	
Instruction	Legal Data Types	Execute	Not Execute
DRUM	CT	840.0 µs	339.6 µs
EDRUM	CT	753.2 µs	357.0 µs
MDRMD	CT	411.3 µs	216.4 µs
MDRMW	CT	378.6 µs	147.0 µs

Clock/Calendar Instructions

Clock/Calendar Instructions		DL06	
Instruction		Execute	Not Execute
DATE	V:Data Reg. V:Bit Reg.	24.0 µs	1.2 µs
TIME	V:Data Reg. V:Bit Reg.	50.8 µs	1.2 µs

MODBUS Instructions

Clock/Calendar Instructions		DL06	
Instruction		Execute	Not Execute
MRX	Input, Input Register Coil, Holding Register	120.2 µs	1.3 µs
MWX	Input, Input Register Coil, Holding Register	21.3 µs	1.3 µs

ASCII Instructions

ASCII Instructions		DL06	
Instruction	Legal Data Types	Execute	Not Execute
AIN	V	13.9 µs	12.0 µs
AFIND	V	111.5 µs	1.3 µs
AEX	V	111.7 µs	1.3 µs
CMPV	V	12.2 µs	1.3 µs
SWAPB	V	109.8 µs	1.3 µs
VPRINT	Text Data	161.6 µs	1.3 µs
PRINTV	V	163.3 µs	1.3 µs
ACRB	V	3.9 µs	1.1 µs

Special Relays

In This Appendix...

DL06 PLC Special Relays

Special Relays are just contacts that are set by the CPU operating system to indicate a particular system event has occurred. These contacts are available for use in your ladder program. Knowing just the right special relay contact to use for a particular situation can save a lot of programming time. Since the CPU operating system sets and clears special relay contacts, the ladder program only has to use them as inputs in ladder logic.

Startup and Real-Time Relays

SP0	First scan	On for the first scan after a power cycle or program to run transition only. The relay is reset to off on the second scan. It is useful where a function needs to be performed only on program startup.
SP1	Always ON	Provides a contact to ensure an instruction is executed every scan.
SP2	Always OFF	Provides a contact that is always off.
SP3	1 minute clock	On for 30 seconds and off for 30 seconds.
SP4	1 second clock	On for 0.5 second and off for 0.5 second.
SP5	100 ms clock	On for 50 ms. and off for 50 ms.
SP6	50 ms clock	On for 25 ms. and off for 25 ms.
SP7	Alternate scan	On every other scan.

CPU Status Relays

SP11	Forced run mode	On when the mode switch is in the run position and the CPU is running.
SP12	Terminal run mode	On when the mode switch is in the TERM position and the CPU is in the run mode.
SP13	Test run mode	On when the CPU is in the test run mode.
SP15	Test stop mode	On when the CPU is in the test stop mode.
SP16	Terminal PGM mode	On when the mode switch is in the TERM position and the CPU is in program mode.
SP17	Forced stop	On when the mode switch is in the STOP position.
SP20	Forced stop mode	On when the STOP instruction is executed.
SP22	Interrupt enabled	On when interrupts have been enabled using the ENI instruction.

System Monitoring

SP36	Override setup relay	On when the override function is used.
SP37	Scan controller	On when the actual scan time runs over the prescribed scan time.
SP40	Critical error	On when a critical error such as I/O communication loss has occurred.
SP41	Warning	On when a non critical error has occurred.
SP42	Diagnostics error	On when a diagnostics error or a system error occurs.
SP43	Low battery error	On when the CPU battery voltage is low (only if bit 12 of V7633 is set).
SP44	Program memory error	On when a memory error such as a memory parity error has occurred.
SP45	I/O error	On when an I/O error such as a blown fuse occurs.
SP46	Communications error	On when a communication error occurs on any of the CPU ports.
SP50	Fault instruction	On when a Fault Instruction is executed.
SP51	Watch Dog timeout	On if the CPU Watch Dog timer times out.
SP52	Grammatical error	On if a grammatical error has occurred either while the CPU is running or if the syntax check is run. V7755 will hold the exact error code.
SP53	Solve logic error	On if CPU cannot solve the logic.
SP54	Communication error	On when RX, WX, instructions are executed with the wrong parameters.
SP56	Table instruction overrun	On if a table instruction with a pointer is executed and the pointer value is outside the table boundary.

Accumulator Status

SP60	Value less than	On when the accumulator value is less than the instruction value.
SP61	Value equal to	On when the accumulator value is equal to the instruction value.
SP62	Greater than	On when the accumulator value is greater than the instruction value.
SP63	Zero	On when the result of the instruction is zero (in the accumulator).
SP64	Half borrow	On when the 16 bit subtraction instruction results in a borrow.
SP65	Borrow	On when the 32 bit subtraction instruction results in a borrow.
SP66	Half carry	On when the 16 bit addition instruction results in a carry.
SP67	Carry	On when the 32 bit addition instruction results in a carry.
SP70	Sign	On anytime the value in the accumulator is negative.
SP71	Pointer reference error	On when the V-memory specified by a pointer (P) is not valid.
SP72	Floating point number	On anytime the value in the accumulator is a valid floating point number.
SP73	Overflow	On if overflow occurs in the accumulator when a signed addition or subtraction results in an incorrect sign bit.
SP74	Underflow	On anytime a floating point math operation results in an underflow error.
SP75	Data error	On if a BCD number is expected and a non–BCD number is encountered.
SP76	Load zero	On when any instruction loads a value of zero into the accumulator.

HSIO Input Status

SP100	X0 status	On when X0 is on
SP101	X1 status	On when X1 is on
SP102	X2 status	On when X2 is on
SP103	X3 status	On when X3 is on

HSIO Pulse Output Relay

SP104	Profile Complete	On when the pulse output profile is completed. (Mode 30)

Communication Monitoring Relay

SP116	CPU port busy Port 2	On when port 2 is the master and sending data.
SP117	Communications error Port 2	On when port 2 is the master and has a communication error.

Option Slot Communication Monitoring Relay

SP120	Slot 1 busy	H0-ECOM/D0-DCM Port2
SP121	Slot 1 error	H0-ECOM/D0-DCM Port2
SP122	Slot 2 busy	H0-ECOM/D0-DCM Port2
SP123	Slot 2 error	H0-ECOM/D0-DCM Port2
SP124	Slot 3 busy	H0-ECOM/D0-DCM Port2
SP125	Slot 3 error	H0-ECOM/D0-DCM Port2
SP126	Slot 4 busy	H0-ECOM/D0-DCM Port2
SP127	Slot 4 error	H0-ECOM/D0-DCM Port2

Option Slot Special Relay

SP140-237	Slot 1	SP relay for option card
SP240-337	Slot 2	SP relay for option card
SP340-437	Slot 3	SP relay for option card
SP430-537	Slot 4	SP relay for option card

Counter 1 Mode 10 Equal Relays

SP540	Current = target value	On when the counter current value equals the value in V3631/3630
SP541	Current = target value	On when the counter current value equals the value in V3633/3632
SP542	Current = target value	On when the counter current value equals the value in V3635/3634
SP543	Current = target value	On when the counter current value equals the value in V3637/3636
SP544	Current = target value	On when the counter current value equals the value in V3641/3640
SP545	Current = target value	On when the counter current value equals the value in V3643/3642
SP546	Current = target value	On when the counter current value equals the value in V3645/3644
SP547	Current = target value	On when the counter current value equals the value in V3647/3646
SP550	Current = target value	On when the counter current value equals the value in V3651/3650
SP551	Current = target value	On when the counter current value equals the value in V3653/3652
SP552	Current = target value	On when the counter current value equals the value in V3655/3654
SP553	Current = target value	On when the counter current value equals the value in V3657/3656
SP554	Current = target value	On when the counter current value equals the value in V3661/3660
SP555	Current = target value	On when the counter current value equals the value in V3663/3662
SP556	Current = target value	On when the counter current value equals the value in V3665/3664
SP557	Current = target value	On when the counter current value equals the value in V3667/3666
SP560	Current = target value	On when the counter current value equals the value in V3671/3670
SP561	Current = target value	On when the counter current value equals the value in V3673/3672
SP562	Current = target value	On when the counter current value equals the value in V3675/3674
SP563	Current = target value	On when the counter current value equals the value in V3677/3676
SP564	Current = target value	On when the counter current value equals the value in V3771/3770
SP565	Current = target value	On when the counter current value equals the value in V3703/3702
SP566	Current = target value	On when the counter current value equals the value in V3705/3704
SP567	Current = target value	On when the counter current value equals the value in V3707/3706

Counter 2 Mode 10 Equal Relays

SP570	Current = target value	On when the counter current value equals the value in V3711/3710
SP571	Current = target value	On when the counter current value equals the value in V3713/3712
SP572	Current = target value	On when the counter current value equals the value in V3715/3714
SP573	Current = target value	On when the counter current value equals the value in V3717/3716
SP574	Current = target value	On when the counter current value equals the value in V3721/3720
SP575	Current = target value	On when the counter current value equals the value in V3723/3722
SP576	Current = target value	On when the counter current value equals the value in V3725/3724
SP577	Current = target value	On when the counter current value equals the value in V3727/3726
SP600	Current = target value	On when the counter current value equals the value in V3731/3730
SP601	Current = target value	On when the counter current value equals the value in V3733/3732
SP602	Current = target value	On when the counter current value equals the value in V3735/3734
SP603	Current = target value	On when the counter current value equals the value in V3737/3736
SP604	Current = target value	On when the counter current value equals the value in V3741/3740
SP605	Current = target value	On when the counter current value equals the value in V3743/3742
SP606	Current = target value	On when the counter current value equals the value in V3745/3744
SP607	Current = target value	On when the counter current value equals the value in V3747/3746
SP610	Current = target value	On when the counter current value equals the value in V3751/3750
SP611	Current = target value	On when the counter current value equals the value in V3753/3752
SP612	Current = target value	On when the counter current value equals the value in V3755/3754
SP613	Current = target value	On when the counter current value equals the value in V3757/3756
SP614	Current = target value	On when the counter current value equals the value in V3761/3760
SP615	Current = target value	On when the counter current value equals the value in V3763/3762
SP616	Current = target value	On when the counter current value equals the value in V3765/3764
SP617	Current = target value	On when the counter current value equals the value in V3767/3766

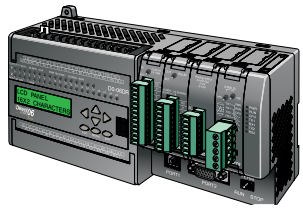
High-Speed Input and Pulse Output Features

In This Appendix...

Introduction

Built-in Motion Control Solution

Many machine control applications require various types of simple high-speed monitoring and control. These applications usually involve some type of motion control, or high-speed interrupts for time-critical events. The DL06 Micro PLC solves this traditionally expensive problem with built-in CPU enhancements. Let's take a closer look at the available high-speed I/O features.



The available **high-speed input features** are:

- High Speed Counter (7 kHz max.) with up to 24 counter presets and built-in interrupt subroutine, counts up only, with reset
- Quadrature encoder inputs to measure counts and clockwise or counter-clockwise direction (7 kHz max.), counts up or down, with reset
- High-speed interrupt inputs for immediate response to critical or time-sensitive tasks
- Pulse catch feature to monitor one input point, having a pulse width as small as 100µs (0.1ms)
- Programmable discrete filtering (both on and off delay up to 99ms) to ensure input signal integrity (this is the default mode for inputs X0–X3)

The available **pulse output features** are:

- Single-axis programmable pulse output (10 kHz max.) with three profile types, including trapezoidal moves, registration, and velocity control

Availability of HSIO Features

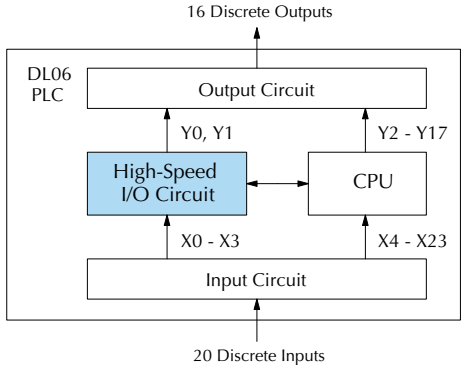
IMPORTANT: Please note the following restrictions on availability of features:

- High-speed input options are available only on DL06s with DC inputs.
- Pulse output options are available only on DL06s with DC outputs.
- Only one HSIO feature may be in use at one time. You cannot use a high–speed input feature and the pulse output at the same time.

Specifications				
DL06 Part Number	**Discrete Input Type**	**Discrete Output Type**	**High-Speed Input**	**Pulse Output**
D0–06AA	AC	AC	No	No
D0–06AR	AC	Relay	No	No
D0–06DA	DC	AC	Yes	No
D0–06DD1	DC	DC	Yes	Yes
D0–06DD2	DC	DC	Yes	Yes
D0–06DR	DC	Relay	Yes	No
D0–06DD1–D	DC	DC	Yes	Yes
D0–06DD2–D	DC	DC	Yes	Yes
D0–06DR–D	DC	Relay	Yes	No

Dedicated High- Speed I/O Circuit

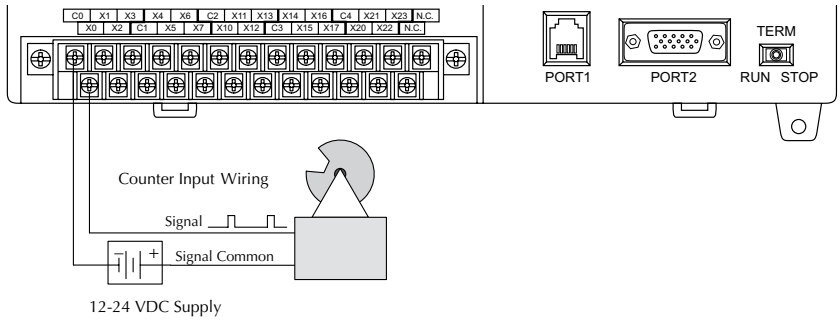
The internal CPU's main task is to execute the ladder program and read/write all I/O points during each scan. In order to service high-speed I/O events, the DL06 includes a special circuit which is dedicated to a portion of the I/O points. Refer to the DL06 block diagram in the figure below.



The high-speed I/O circuit (HSIO) is dedicated to the first four inputs (X0 – X3) and the first two outputs (Y0 – Y1). We might think of this as a *CPU helper*. In the default operation (called "Mode 60") the HSIO circuit just passes through the I/O signals to or from the CPU, so that all twenty inputs behave equally and all sixteen outputs behave equally. When the CPU is configured in any other HSIO Mode, the HSIO circuit imposes a specialized function on the portion of inputs and outputs shown. The HSIO circuit *operates independently of the CPU program scan*. This provides accurate measurement and capturing of high-speed I/O activity while the CPU is busy with ladder program execution.

Wiring Diagrams for Each HSIO Mode

After choosing the appropriate HSIO mode for your application, you'll need to refer to the section in this appendix for that specific mode. Each section includes wiring diagrams to help you connect the High-Speed I/O points correctly to field devices. An example of a High Speed Counter mode diagram is shown below.



Choosing the HSIO Operating Mode

Understanding the Six Modes

The High-Speed I/O circuit operates in one of 6 basic modes as listed in the table below. The number in the left column is the mode number (later, we'll use these numbers to configure the PLC). Choose one of the following modes according to the primary function you want from the dedicated High-Speed I/O circuit. You can simply use all twenty inputs and sixteen outputs as regular I/O points with Mode 60.

High Speed I/O Basic Modes		
Mode		Mode Features
10	High-Speed Counter	Two 7 kHz counters with 24 presets and reset input, counts up only, cause interrupt on preset
20	Up/Down Counter	7 kHz up/down counter with 24 presets and reset, causes interrupt on preset
		Channel A / Channel B 7 kHz quadrature input, counts up and down
30	Pulse Output	Stepper control – pulse and direction signals, programmable motion profile (10 kHz max.)
40	High-Speed Interrupt	Generates an interrupt based on input transition or time
50	Pulse Catch	Captures narrow pulses on a selected input
60	Filtered Input	Rejects narrow pulses on selected inputs

In choosing one of the six high-speed I/O modes, the I/O points listed in the table below operate only as the function listed. If an input point is not specifically used to support a particular mode, it usually operates as a filtered input by default. Similarly, output points operate normally unless Pulse Output mode is selected.

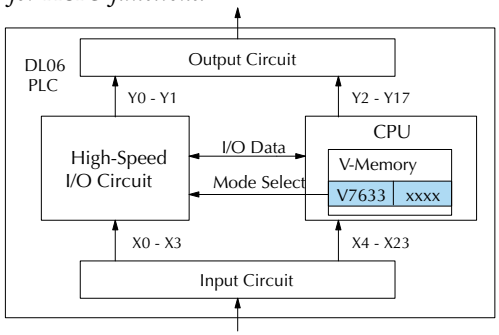
Physical I/O Point Usage							
Mode		DC Input Points				DC Output Points	
		X0	X1	X2	X3	Y0	Y1
10	High-Speed Counter	Counter #1	Counter #2, Interrupt, Pulse Input or Filtered Input	Reset #1, Interrupt, Pulse Input or Filtered Input	Reset #2, Interrupt, Pulse Input or Filtered Input	Regular Output	Regular Output
20	Up/Down counter (Standard counting)	Up Counting	Down Counting	Reset, Pulse Input or Filtered Input	Pulse Input or Filtered Input	Regular Output	Regular Output
	Up/Down counter (Quadrature counting)	Phase A Input	Phase B Input				
30	Pulse Output	Pulse Input or Filtered Input	Pulse Input or Filtered Input	Pulse Input or Filtered Input	Pulse Input or Filtered Input	Pulse or CW Pulse	Direction or CCW Pulse
40	High-Speed Interrupt	Interrupt	Interrupt, Pulse Input or Filtered Input	Interrupt, Pulse Input or Filtered Input	Interrupt, Pulse Input or Filtered Input	Regular Output	Regular Output
50	Pulse Catch	Pulse Input	Pulse Input, Interrupt or Filtered Input	Pulse Input, Interrupt or Filtered Input	Pulse Input, Interrupt or Filtered Input	Regular Output	Regular Output
60	Filtered Input	Filtered Input	Filtered Input	Filtered Input	Filtered Input	Regular Output	Regular Output

Default Mode

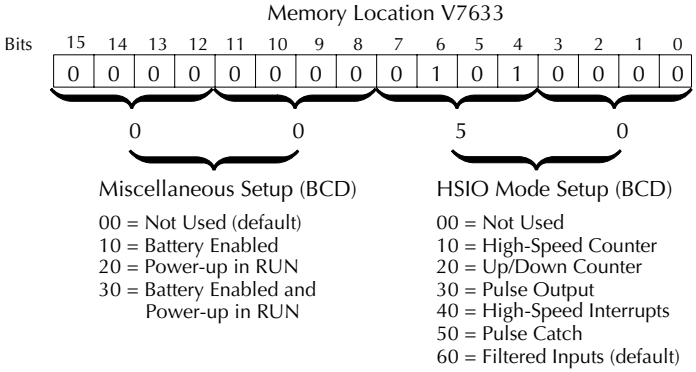
Mode 60 (Filtered Inputs) is the default mode. The DL06 is initialized to this mode at the factory, and any time you initialize the scratchpad memory. In the default condition, X0–X3 are filtered inputs (10 ms delay) and Y0–Y1 are standard outputs.

Configuring the HSIO Mode

If you have chosen a mode suited to the high-speed I/O needs of your application, we're ready to proceed to configure the PLC to operate accordingly. In the block diagram below, notice the V-memory detail in the expanded CPU block. V-memory location V7633 determines the functional mode of the high-speed I/O circuit. *This is the most important V-memory configuration value for HSIO functions!*



The contents of V7633 is a 16-bit word, to be entered in binary–coded decimal. The figure below defines what each 4-bit BCD digit of the word represents.



Bits 0 – 7 define the mode number 00, 10.. 60 previously referenced in this appendix. The example data "0050" shown selects Mode 50 – Pulse Catch (BCD = 50).

Configuring Inputs X0 – X3

In addition to configuring V7633 for the HSIO mode, you'll need to program the next four locations in certain modes according to the desired function of input points X0 – X3. Other memory locations may require configuring, depending on the HSIO mode (see the corresponding section for particular HSIO modes).

	V-Memory	
Mode	V7633	xxxx
X0	V7634	xxxx
X1	V7635	xxxx
X2	V7636	xxxx
X3	V7637	xxxx

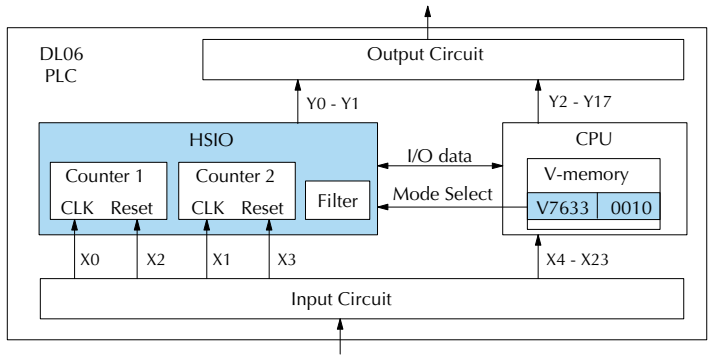
Mode 10: High-Speed Counter

Purpose

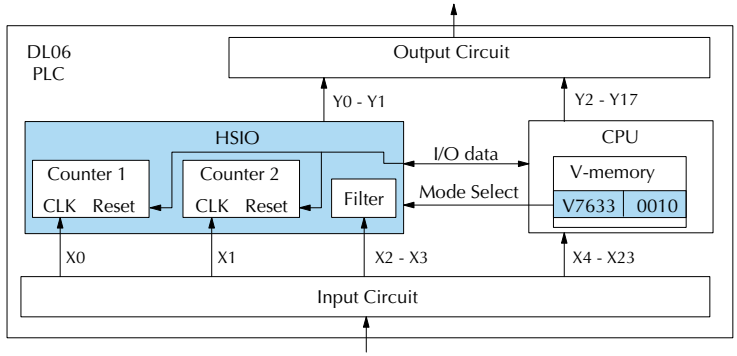
The HSIO circuit contains two high-speed counters. A single pulse train from an external source (X0) clocks the counter on each signal leading edge. The counter counts only upwards, from 0 to 99999999. The counter compares the current count with up to 24 preset values, which you define. The purpose of the presets is to quickly cause an action upon arrival at specific counts, making it ideal for such applications as cut-to-length. It uses counter registers CT174 to CT177 in the CPU.

Functional Block Diagram

Refer to the block diagram below. When the lower byte of HSIO Mode register V7633 contains a BCD "10", the high-speed up counter in the HSIO circuit is enabled. X0 and X1 automatically become the "clock" inputs for the high-speed counters, incrementing them upon each off-to-on transition. The external reset input on X2 and X3 are the default configuration for Mode 10.

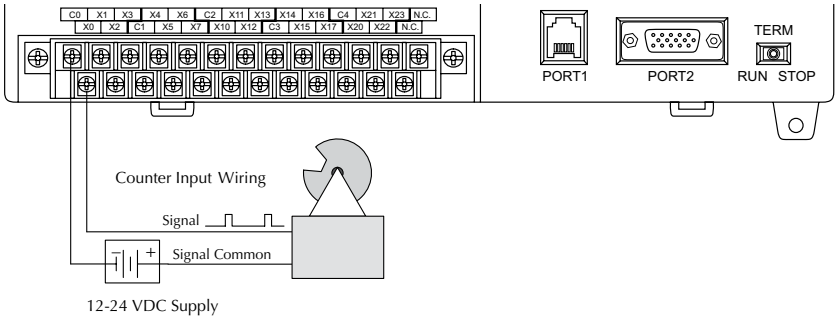


Instead of using X2 and X3 as dedicated reset inputs, you can configure X2 and X3 as normal filtered inputs. In this way, the counter reset must be generated in ladder logic.



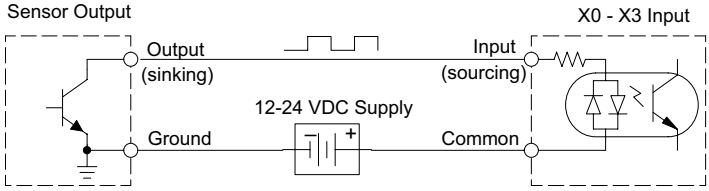
Wiring Diagram

A general wiring diagram for counters/encoders to the DL06 in HSIO Mode 10 is shown below. Many types of pulse-generating devices may be used, such as proximity switches, single-channel encoders, magnetic or optical sensors, etc. Devices with sinking outputs (NPN open collector) are probably the best choice for interfacing. If the counter sources to the inputs, it must output 12 to 24 VDC. Note that devices with 5V sourcing outputs will not work with DL06 inputs.

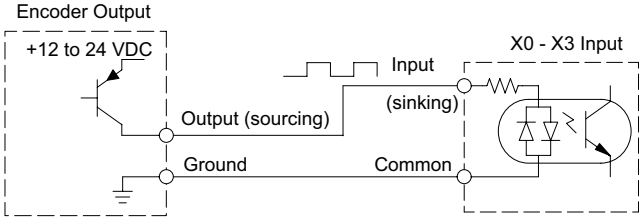


Interfacing to Counter Inputs

The DL06's DC inputs are flexible in that they detect current flow in either direction, so they can be wired to a sensor with either sourcing or sinking outputs. In the following circuit, a sensor has an open-collector NPN transistor output. It sinks current from the PLC input point, which sources current. The power supply can be the FA–24PS or another supply (+12VDC or +24VDC), as long as the input specifications are met.

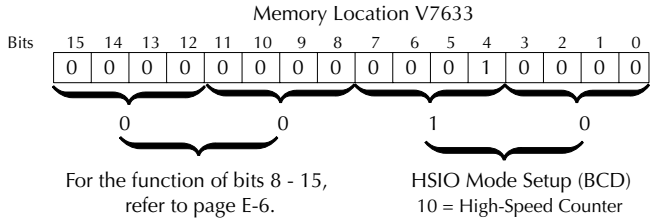


In the circuit diagram below, an encoder has open-emitter PNP transistor outputs. It sources current to the PLC input point, which sinks the current back to ground. Since the encoder sources current, no additional power supply is required. However, note that the encoder output must be 12 to 24 volts (5V encoder outputs will not work).



Setup for Mode 10

V7633 is the HSIO Mode Select register. Refer to the diagram below. Use BCD 10 in the lower byte of V7633 to select the High-Speed Counter Mode.



Choose the most convenient method of programming V7633 from the following:

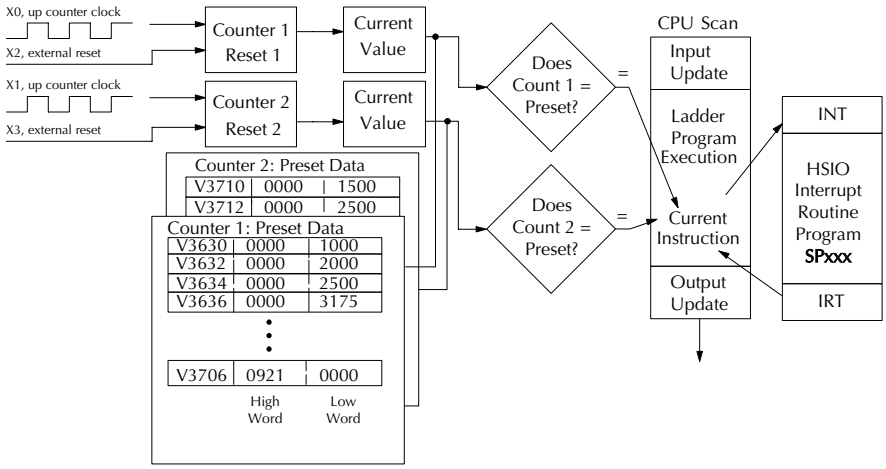
- Include load and out instructions in your ladder program
- *Direct*SOFT's memory editor or Data View
- Use the Handheld Programmer D2–HPP

We recommend using the first method above so that the HSIO setup becomes an integral part of your application program. An example program later in this section shows how to do this

Presets and Special Relays

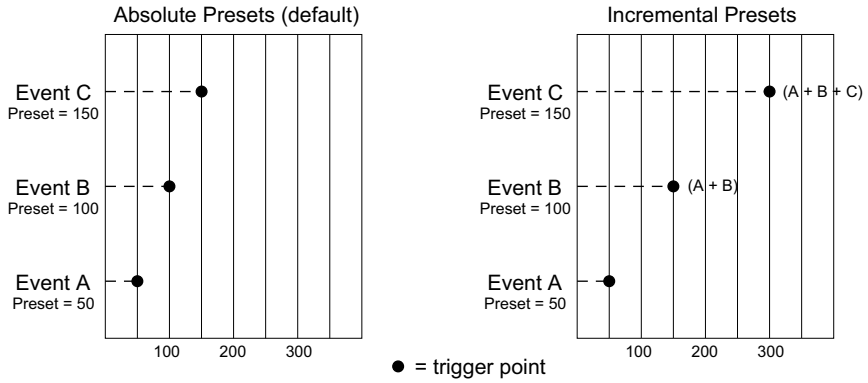
Presets are used to cause a particular action to occur when the count reaches the preset value. Refer to the figure below. Each counter features 24 presets, which you can program. Presets are double word numbers so they occupy two V-memory registers. The user selects the preset values, and the counter continuously compares the current count with the preset. When the two are equal, a special relay contact is energized and program execution jumps to the interrupt routine.

We recommend using the special relay(s) in the interrupt service routine to cause any immediate action you desire. After the interrupt service routine is complete, the CPU returns to the ladder program, resuming program execution from the point of interruption. The compare function is ready for the next preset event.



Absolute and Incremental Presets

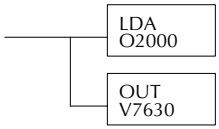
Two preset modes are available, absolute and incremental. Presets are entered into a contiguous block of V-memory registers. In the absolute mode, each preset is treated as the total count. In the incremental mode, the presets are cumulative. Incremental presets represent the number of counts between events.

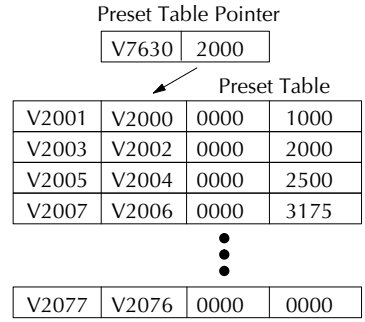


In the example above, presets are established at 50, 100, and 150. The difference between absolute and incremental modes is shown. Absolute presets trigger events at the preset values, 50, 100, and 150. Incremental presets trigger events at the cumulative totals: 50, 150, and 300.

Preset Data Starting Location

V7630 is the pointer to the V-memory location which contains the beginning of the Preset Data Tables. The default starting location for the Preset Data Tables is V3630 (default after initializing scratchpad). However, you may change this by programming a different value in V7630. Use the LDA and OUT instructions as shown:



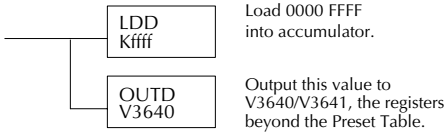


Using Fewer than 24 Presets

When all 24 available presets are used, the CPU knows automatically when it reaches the end of the preset table. When using fewer than 24 presets, however, it is necessary to signal the CPU that it has reached the last preset. The way to signal the end of the block of presets is to insert one of the following **table-end** codes into the next available register pair:

Table-end Code	Applicable Mode	Meaning
0000 FFFF	Absolute and Incremental	Signals end of presets
0000 00FF	Incremental	Signals end of presets and restarts presets. Does not reset accumulated pulse counts of CT174 or CT176.
0000 FF00	Incremental	Signals end of presets, restarts presets and resets accumulated pulse counts of CT174 or CT176.

As shown in the table above, each of the **table-end** signals has a different meaning. Use the LDD Kffff instruction to insert the table-end code into the next register pair beyond the preset table. In the example, four presets are used. The 0000 FFFF in V3641-V3640 indicates the previous preset was the last preset.



Default Preset Table Example

V3631	V3630	0000	1000
V3633	V3632	0000	2000
V3635	V3634	0000	2500
V3637	V3636	0000	3175
V3641	V3640	0000	FFFF

In incremental mode, you can choose not to reset the counter or the cumulative total, or you can choose to reset only the counter, or you can choose to reset the counter and the cumulative total when the table-end code is read. In the example, FFFF has been placed in V3640 since the last preset was in V3636, and we are using fewer than 24 presets.

NOTE: *In absolute mode each successive preset must be greater than the previous preset value. If a preset value is less than a lower-numbered preset value, the CPU cannot compare to that value, since the counter can only count upwards.*

Equal Relay Numbers

The following table lists all 24 preset register default locations for each high-speed counter. Each occupies two 16-bit V-memory registers. The corresponding special relay contact number is in the next column. We might also call these *equal* relay contacts, because they are true (closed) when the present high-speed counter value is equal to the preset value. Each contact remains closed until the counter value equals the next preset value.

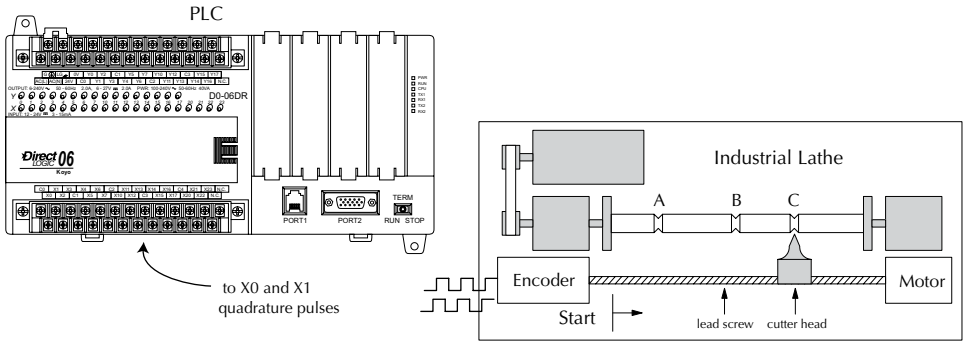
Preset Register Table					
Counter 1 Preset	Preset V-memory Register	Special Relay Number	Counter 2 Preset	Preset V-memory Register	Special Relay Number
1	V3631 / V3630	SP540	1	V3711 / V3710	SP570
2	V3633 / V3632	SP541	2	V3713 / V3712	SP571
3	V3635 / V3634	SP542	3	V3715 / V3714	SP572
4	V3637 / V3636	SP543	4	V3717 / V3716	SP573
5	V3641 / V3640	SP544	5	V3721 / V3720	SP574
6	V3643 / V3642	SP545	6	V3723 / V3722	SP575
7	V3645 / V3644	SP546	7	V3725 / V3724	SP576
8	V3647 / V3646	SP547	8	V3727 / V3726	SP577
9	V3651 / V3650	SP550	9	V3731 / V3730	SP600
10	V3653 / V3652	SP551	10	V3733 / V3732	SP601
11	V3655 / V3654	SP552	11	V3735 / V3734	SP602
12	V3657 / V3656	SP553	12	V3737 / V3736	SP603
13	V3661 / V3660	SP554	13	V3741 / V3740	SP604
14	V3663 / V3662	SP555	14	V3743 / V3742	SP605
15	V3665 / V3664	SP556	15	V3745 / V3744	SP606
16	V3667 / V3666	SP557	16	V3747 / V3746	SP607
17	V3671 / V3670	SP560	17	V3751 / V3750	SP610
18	V3673 / V3672	SP561	18	V3753 / V3752	SP611
19	V3675 / V3674	SP562	19	V3755 / V3754	SP612
20	V3677 / V3676	SP563	20	V3757 / V3756	SP613
21	V3701 / V3700	SP564	21	V3761 / V3760	SP614
22	V3703 / V3702	SP565	22	V3763 / V3762	SP615
23	V3705 / V3704	SP566	23	V3765 / V3764	SP616
24	V3707 / V3706	SP567	24	V3767 / V3766	SP617

The consecutive addresses shown above for each relay are those assigned by the CPU as default addresses. The Pointer for the start of these addresses is stored by the CPU at V7630. If you have a conflict of addresses because of pre-existing code written to these addresses, you can change the default block of addresses merely by having your ladder logic place a different pointer value in V7630. To change the table location. use the LDA and OUT instructions as shown on the previous page.

Calculating Your Preset Values

The preset values occupy two data words each. They can range in value from -8388608 to 8388607, just like the high-speed counter value. All 24 values are absolute values, meaning that each one is an offset from the counter zero value.

The preset values must be individually derived for each application. In the industrial lathe diagram below, the PLC monitors the position of the lead screw by counting pulses. At points A, B, and C along the linear travel, the cutter head pushes into the work material and cuts a groove.



The timing diagram below shows the duration of each equal relay contact closure. Each contact remains on until the next one closes. All go off when the counter resets.

NOTE: *Each successive preset must be two numbers greater than the previous preset value. In the industrial lathe example, B>A+2 and C>B+2.*

X Input Configuration

The configurable discrete input options for High-Speed Counter Mode are listed in the table below. Input X0 is dedicated for the first counter clock input. Input X1 can be the clock for the second counter or a filtered input. The section on Mode 60 operation at the end of this appendix describes programming the filter time constants. Inputs X2 and X3 can be configured as the counter resets, with or without the interrupt option. The interrupt option allows the reset input (X2 and X3) to cause an interrupt like presets do, but there is no SP relay contact closure (instead, X2 and X3 will be on during the interrupt routine, for 1 scan). Or finally, X2 and X3 may be left simply as a filtered input.

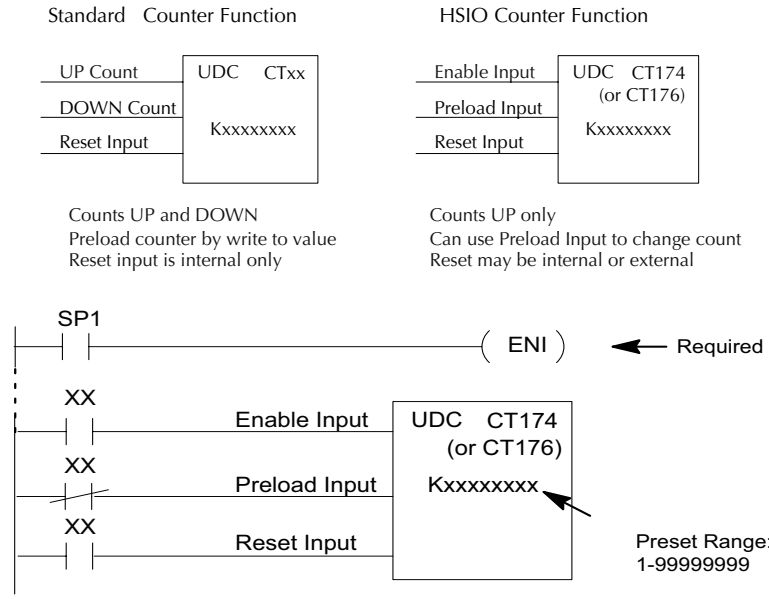
Input Options			
Input	**Configuration Register**	**Function**	**Hex Code Required**
X0	V7634	Counter #1 Clock	0001 (absolute) (default) 0101 (incremental)
X1	V7635	Counter #2 Clock	0001 (absolute) (default) 0101 (incremental)
		Interrupt	0004
		Pulse Input	0005
		Filtered Input	xx06, xx = filter time 0 - 99 ms (BCD)
X2	V7636	Counter #1 Reset (no interrupt)	0007* (default) 0207*
		Counter #1 Reset (with interrupt)	0107* 0307*
		Interrupt	0004
		Pulse Input	0005
		Filtered Input	xx06, xx= filter time 0 - 99 ms (BCD)
X3	V7637	Counter #2 Reset (no interrupt)	0007* (default) 0207*
		Counter #2 Reset (with interrupt)	0107* 0307*
		Interrupt	0004
		Pulse Input	0005
		Filtered Input	xx06, xx= filter time 0 - 99 ms (BCD)

*With the counter reset, you have the option of a normal reset or a faster reset. However, the fast reset does not recognize changed preset values during program execution. When '0007' or '0107' are set in V7636 or V7637 and preset values are changed during program execution, the DL06 recognizes the changed preset values at the time of the reset. When '0207' or '0307' are set in V7636 or V7637, the CPU does not check for changed preset values, so the DL06 has a faster reset time.

Writing Your Control Program

The mnemonic for the counter instruction is UDC (up-down counter).The DL06 can have up to 128 counters, labeled CT0 through CT177. The high speed counter in the HSIO circuit is accessed in ladder logic by using UDC CT174 and CT176. It uses counter registers CT174 through CT177 exclusively when the HSIO mode 10 is active (otherwise, CT174 through CT177 are available for standard counter use). The HSIO counter needs two registers because it is a double-word counter. It has three inputs as shown. The first input (Enable) allows counting when active. The middle input is used to preload the counter value. The bottom signal is the reset. The Preload Input must be off while the counter is counting.

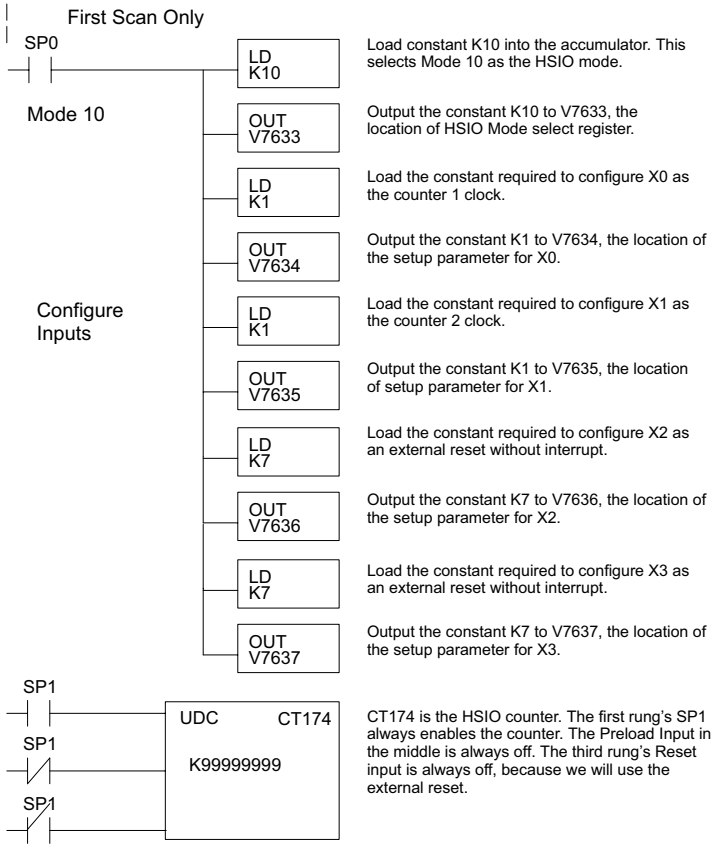
The next figure shows how the HSIO counter will appear in a ladder program. Note that the Enable Interrupt (ENI) command must execute before the counter value reaches the first preset value. We do this at powerup by using the first scan relay. When using the counter but not the presets and interrupt, we can omit the ENI.



When the enable input is energized, the up/down counter CT174 will respond to pulses on X0 and increment. The up/down counter CT176 will respond to pulses on X1 and increment. The reset input contact behaves in a logical OR fashion with the physical reset input. X2 (when selected) resets counter 1. X3 (when selected) resets counter 2. So, the high speed counter can receive a reset from either the contact(s) on the reset rung in the ladder, OR the external reset X2 or X3, if you have configured X2 or X3 as an external reset.

Program Example 1: Counter Without Presets

The following example is the simplest way to use the high-speed counters, which does not use the presets and special relays in the interrupt routine. The program configures the HSIO circuit for Mode 10 operation, so X0 is automatically the counter clock input for the first counter, and X1 is the counter clock input for the second counter. It uses the Compare-double (CMPD) instruction to cause action at certain count values. Note that this allows you to have more than 24 presets. Then, it configures X2 and X3 to be the external reset of the counter.

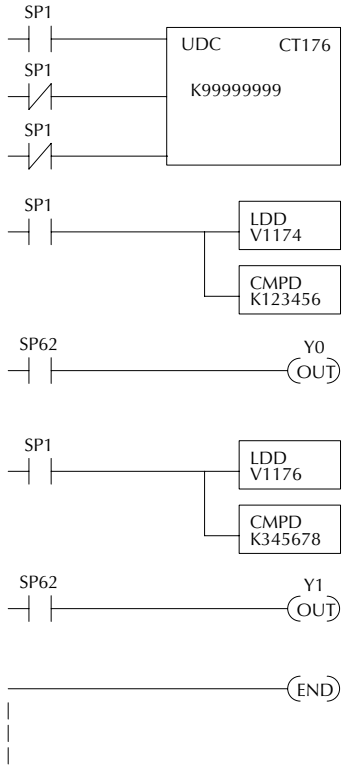


continued on next page

Program Example: (cont'd)

The compare double instructions below use the current count of the HSIO counter to turn on Y0 and Y1. This technique can make more than 24 comparisons, but it is scan-time dependent. However, use the 24 built-in presets with the interrupt routine if your application needs a very fast response time, as shown in the next example.

continued from previous page



CT176 always enables the counter. The Preload Input in the middle is always of f. The third rung's Reset input is always off, because we will use the external reset.

Load the current count of the HSIO counter in V1174 and V1175 into the accumulator

Use the Compare-double instruction to compare the double word in the accumulator to the constant K123456.

The execution of the above CMPD instruction turns on special relay contact SP62 if the current count is greater than the comparison number (K123456).

Load the current count of the HSIO counter in V1176 and V1177 into the accumulator

Use the Compare-double instruction to compare the double word in the accumulator to the constant K345678

The execution of the above CMPD instruction turns on special relay contact SP62 if the current count is greater than the comparison number (K345678).

END coil marks the end of the main program.

Program Example 2: Counter With Presets

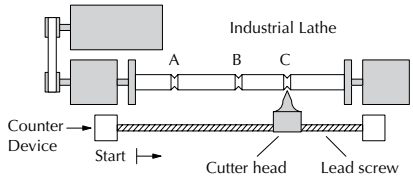
The following example shows how to program the HSIO circuit to trigger on three preset values. You may recall the industrial lathe example from the beginning of this appendix. This example program shows how to control the lathe cutter head to make three grooves in the work-piece at precise positions. When the lead screw turns, the counter device generates pulses which the DL06 can count. The three preset variables A, B, and C represent the positions (number of pulses) corresponding to each of the three grooves. In this example, only one high-speed counter is used. The second counter could be used in the same manner.

Preset Data

A	V3630	0000	1500
B	V3632	0000	3780
C	V3634	0000	4850
	V3636	0000	FFFF

I/O Assignments

X3 - Cutter head extended
X4 - Cutter head retracted
Y0 - Lead screw motor
Y1 - Cutter head solenoid

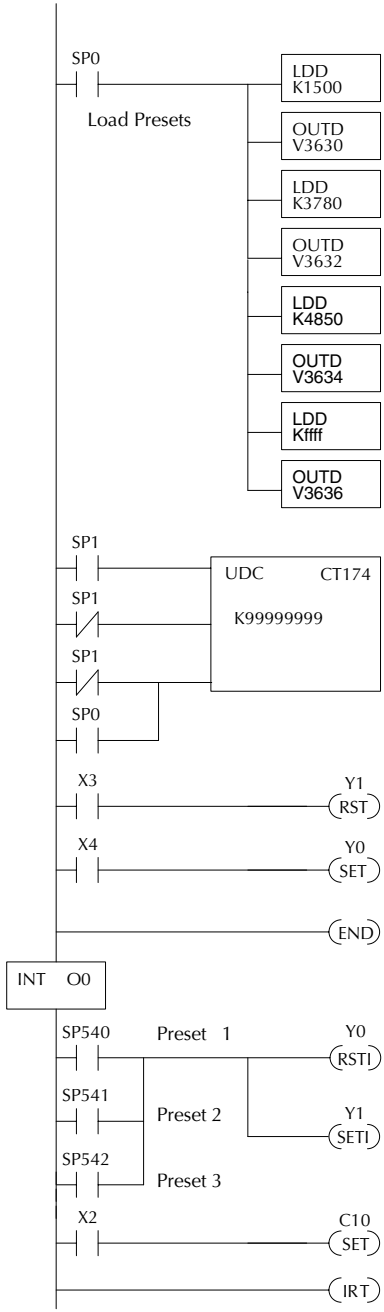


***Direct*SOFT**

SP0	ENI	Enable Interrupts before reaching a preset generates an interrupt. Special Relay SP0 is on during the first CPU scan.
SP0	LD K10	Load constant K10 into the accumulator. This selects Mode 10 as the HSIO mode.
	OUT V7633	Output the constant to V7633, the location of HSIO Mode select register.
	LD K1	Load the constant required to configure X0 as the counter clock.
	OUT V7634	Output the constant K1 to V7634, the location of the setup parameter for X0.
Select Mode 10	LD K1006	Load the constant required to configure X1 as a filtered input.
	OUT V7635	Output the constant to V7635, the location of the setup parameter for X1.
	LD K107	Load the constant required to configure X2 as an external reset with interrupt.
	OUT V7636	Output the constant to V7636, the location of the setup parameter for X2.
	LD K1006	Load the constant required to configure X3 as a filtered input.
	OUT V7637	Output the constant to V7637, the location of the setup parameter for X3.
	LDA O3630	Load the octal address O3630 into the accumulator. This instruction automatically converts the address into hex.
	OUT V7630	Output this address to V7630, the location of the pointer to the Preset Table.

continued on next page

continued from previous page



Load the preset A value into the accumulator .

Output the accumulator contents to the memory location for preset 1.

Load the preset B value into the accumulator .

Output the accumulator contents to the memory location for preset 2.

Load the preset C value into the accumulator .

Output the accumulator contents to the memory location for preset 3.

Load the constant Kffff into the accumulator. This value represents the end of the preset list.

Output the accumulator contents to the memory location for preset 4 (end of preset marker).

CT174 is the HSIO counter. The first rung's SP1 always enables the counter. The preload input in the middle is off (unused in this example).

The third rung's Reset input is normally off, because we will use the external reset. You can optionally reset the counter value on each powerup using the SP0 contact.

Input X3 energizes when the groove has finished cutting. So, we retract the cutter head.

Turn lead screw on again, after cutter head has retracted.

END coil marks the end of the main program.

The INT label marks the beginning of the interrupt service routine program.

Inside the interrupt service routine, we turn OFF the lead screw motor immediately.

These special "equal" relays turn on individually as the corresponding preset is reached. In this application, each results in the cutting of a groove (Y1), so they are logically ORed together .

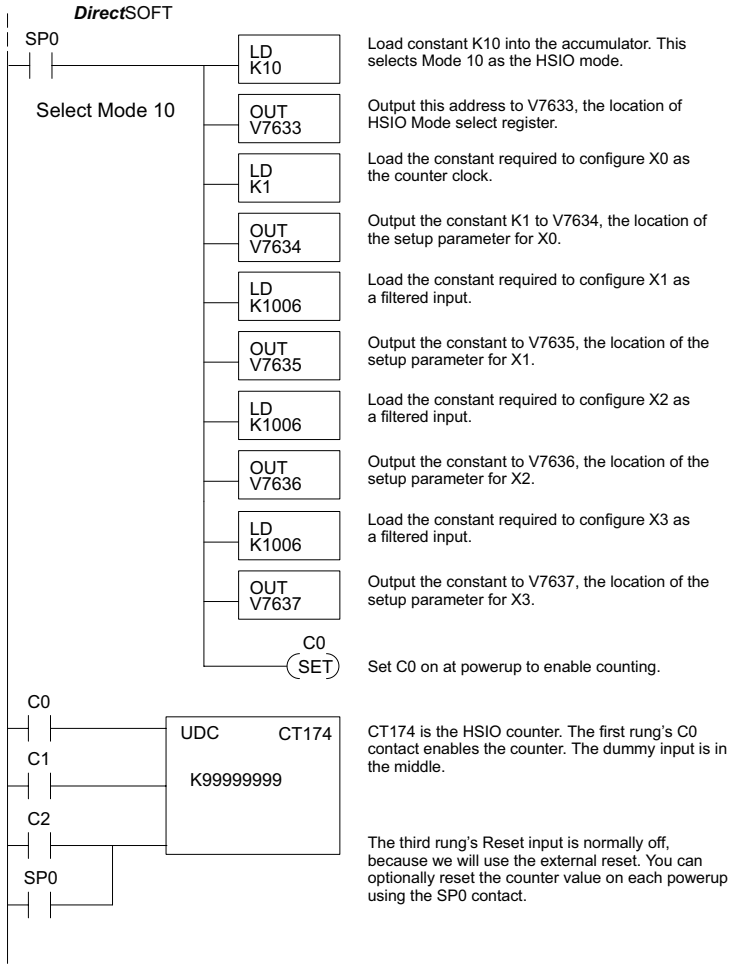
Input X2 will be energized inside the interrupt routine if X2 external interrupt was the source.

Return from the interrupt service routine.

Some applications will require a different type of action at each preset. It is possible for the interrupt routine to distinguish one preset event from another, by turning on a unique output for each equal relay contact SPxxx. We can determine the source of the interrupt by examining the equal relay contacts individually, as well as X2. The X2 contact will be on (inside the interrupt routine only) if the interrupt was caused by the external reset, X2 input.

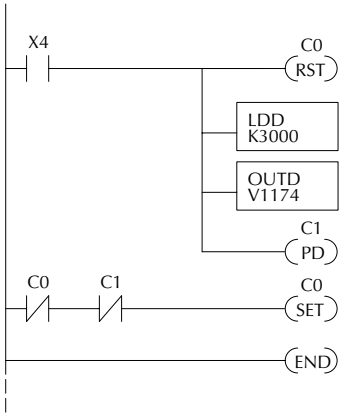
Program Example 3: Counter With Preload

The following example shows how you can preload the current count with another value. When the preload command input (X4 in this example) is energized, we disable the counter from counting with C0. Then, we write the value K3000 to the count register (V1076-V1077). We preload the current count of the counter with K3000. When the preload command (X4) is turned off, the counter resumes counting any pulses, but now starting from K3000. In this example, only one high-speed counter is used. The second counter could be used in the same manner.



continued on next page

continued from previous page



When the preload request is made, the user turns on X4. First we disable counting by resetting C0, the counter's enable input.

Load the BCD value K3000 into the accumulator.

Output the constant to V1174/V1175, the location of the accumulated count for CT76.

Generate a preload counter input pulse, which causes the counter to preload from V1174-V1175.

Enable the counter by setting C0, when the preolad pulse on C1 has occurred (C1 is off).

END coil marks the end of the main program.

Troubleshooting Guide for Mode 10

If you're having trouble with Mode 10 operation, please study the following symptoms and possible causes. The most common problems are listed below.

Symptom: The counter does not count.

Possible causes:

1. **Field sensor and wiring** – Verify that the encoder, proximity switch, or sensor actually turns on and illuminates the status LED for X0 (counter 1) and X1 (counter 2). The problem could be due to sinking-sourcing wiring problem, etc. Remember to check the signal ground connection. Also verify that the pulse on-time is long enough for the PLC to recognize it.
2. **Configuration** – use the Data View window to check the configuration parameters. V7633 must be set to 10, and V7634 must be set to 1 or 101 to enable the first high-speed counter. V7635 must be set to 1 or 101 to enable the second high-speed counter.
3. **Stuck in reset** – check the input status of the reset input, X2 and X3. If X2 is on, the counter will not count because it is being held in reset.
4. **Ladder program** – make sure you are using counter CT174 and CT176 in your program. The top input is the enable signal for the counter. It must be on before the counter will count. The middle input is the dummy input. The bottom input is the counter reset, and must be off during counting.

Symptom: The counter counts but the presets do not function.

Possible causes:

1. **Configuration** – Ensure the preset values are correct. The presets are 32-bit BCD values having a range of 0 to 99999999. Make sure you write all 32 bits to the reserved locations by using the LDD and OUTD instructions. Use only even–numbered addresses, from V3630 to V3767. If using less than 24 presets, be sure to place "0000FFFF," "0000FF00," or "000000FF" in the location after the last preset used.
2. **Interrupt routine** – Only use Interrupt #0. Make sure the interrupt has been enabled by executing an ENI instruction prior to needing the interrupt. The interrupt routine must be placed after the main program, using the INT label and ending with an interrupt return IRT.
3. **Special relays** – Check the special relay numbers in your program. Use SP540 for Preset 1, SP541 for Preset 2, etc. Remember that only one special equal relay contact is on at a time. When the counter value reaches the next preset, the SP contact which is on now goes off and the next one turns on.

Symptom: The counter counts up but will not reset.

Possible causes:

1. Check the LED status indicator for X2 (counter 1) and X3 (counter 2) to make sure it is active when you want a reset. Or, if you are using an internal reset, use the status mode of DirectSOFT to monitor the reset input to the counter.

Mode 20: Up/Down Counter

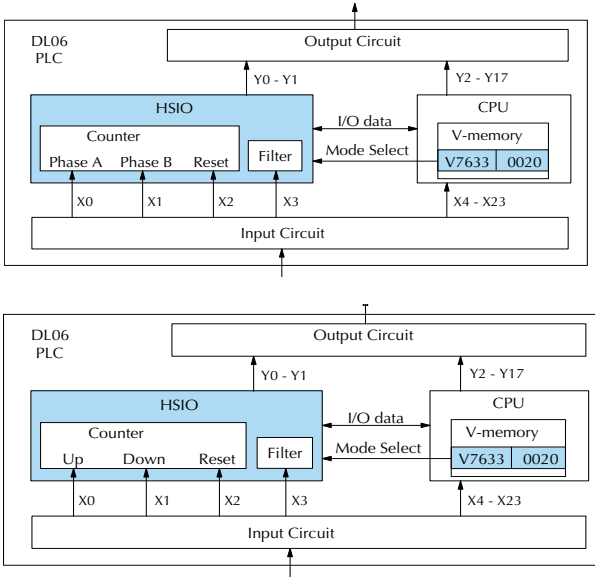
Purpose

The counter in the HSIO circuit can count up/down signals from two separate sources (i.e., two single channel encoders) or two quadrature signal pulses. Quadrature signals are commonly generated from incremental encoders, which may be rotary or linear. The up/down counter has a range from -8388608 to 8388607. Using CT174 and CT175, the quadrature counter can count at up to a 7 kHz rate.

***NOTE:** The count is 32 bit BCD sign + magnitude format.*

Functional Block Diagram

The diagram below shows HSIO functionality in Mode 20. When the lower byte of HSIO Mode register V7633 contains a BCD "20", the up/down counter in the HSIO circuit is enabled. For quadrature counting, input X0 is dedicated to the Phase A quadrature signal, and input X1 receives Phase B signal. X2 is dedicated to reset the counter to zero value when energized.



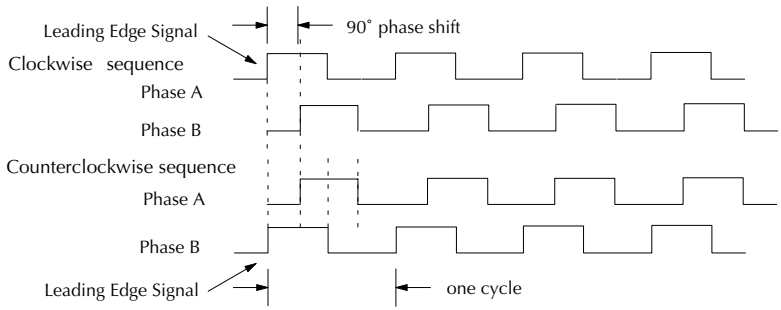
For standard up/down counting, input X0 is dedicated to the up counting signal, and input X1 is dedicated to the down counting signal. The X2 input resets the counter to zero when energized.

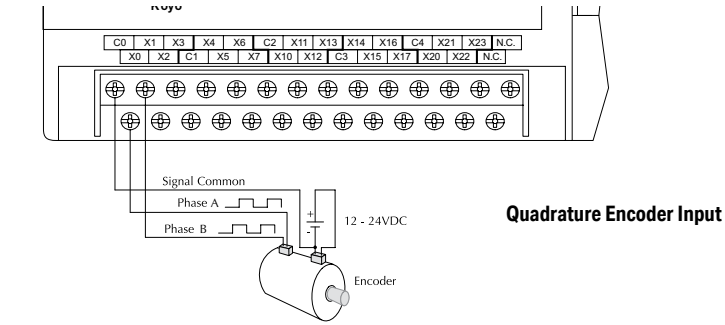
Quadrature Encoder Signals

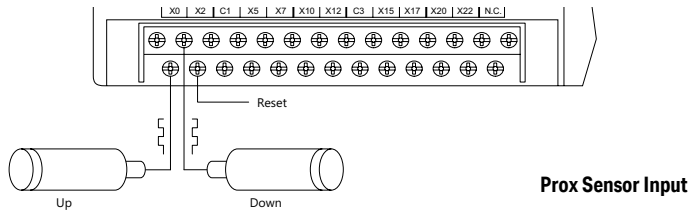
Quadrature encoder signals contain position and direction information, while their frequency represents speed of motion. Phase A and B signals shown below are phase-shifted 90 degrees, thus the quadrature name. When the rising edge of Phase A precedes Phase B's leading edge (indicates clockwise motion by convention), the HSIO counter counts UP. If Phase B's rising edge precedes Phase A's rising edge (indicates counter-clockwise motion), the counter counts DOWN.

Wiring Diagram

A general wiring diagram for encoders to the DL06 in HSIO Mode 20 is shown below. Encoders with sinking outputs (NPN open collector) are probably the best choice for interfacing. If the encoder sources to the inputs, it must output 12 to 24 VDC. Note that encoders with 5V sourcing outputs will not work with DL06 inputs.

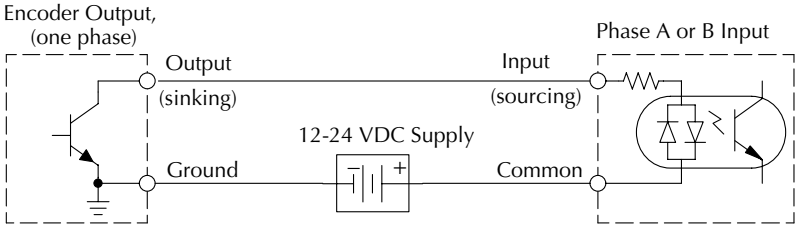




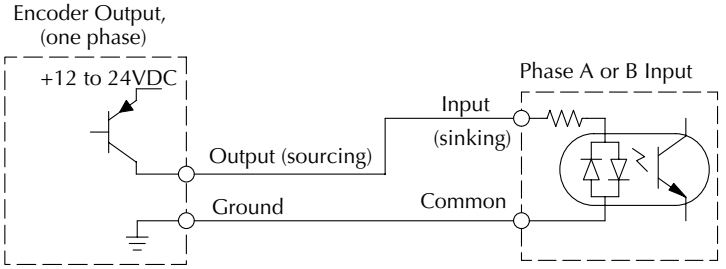


Interfacing to Encoder Outputs

The DL06's DC inputs are flexible in that they detect current flow in either direction, so they can be wired to an encoder with either sourcing or sinking outputs. In the following circuit, an encoder has open-collector NPN transistor outputs. It sinks current from the PLC input point, which sources current. The power supply can be the +24VDC auxiliary supply or another supply (+12VDC or +24VDC), as long as the input specifications are met.

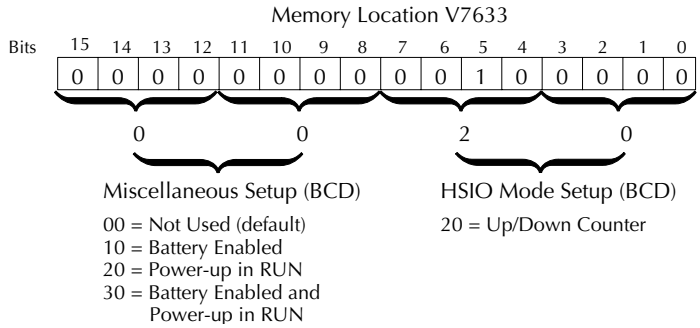


In the next circuit, an encoder has open-emitter PNP transistor outputs. It sources current to the PLC input point, which sinks the current back to ground. Since the encoder sources current, no additional power supply is required. However, note that the encoder output must be 12 to 24 volts (5V encoder outputs will not work).



Setup for Mode 20

Remember that V7633 is the HSIO Mode Select register. Refer to the diagram below. Use BCD 20 in the lower byte of V7633 to select the High-Speed Counter Mode.



Choose the most convenient method of programming V7633 from the following:

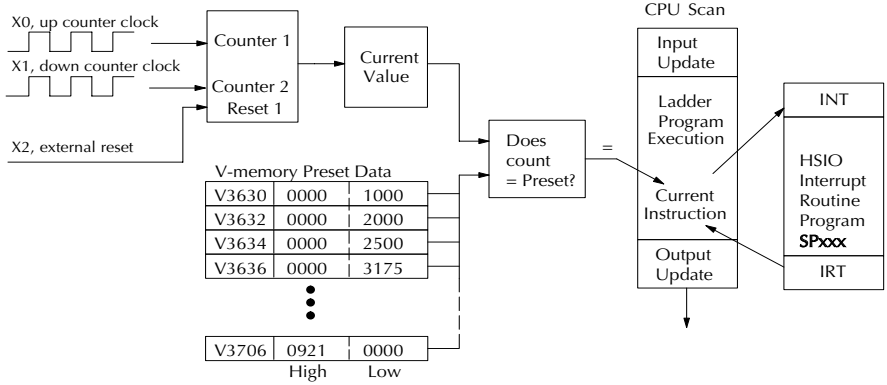
- Include load and out instructions in your ladder program
- ***Direct***SOFT's memory editor
- Use the Handheld Programmer D2–HPP

We recommend using the first method above so that the HSIO setup becomes an integral part of your application program. An example program later in this section shows how to do this.

Presets and Special Relays

The goal of counting is to cause a particular action to occur when the count reaches a preset value. Refer to the figure below. Each counter features 24 presets, which you can program. A preset is a number you select and store so that the counter will continuously compare the current count with the preset. When the two are equal, a special relay contact is energized and program execution jumps to the interrupt routine.

We recommend using the special relay(s) in the interrupt service routine to cause any immediate action you desire. After the interrupt service routine is complete, the CPU returns to the ladder program, resuming program execution from the point of interruption. The compare function is ready for the next preset event.



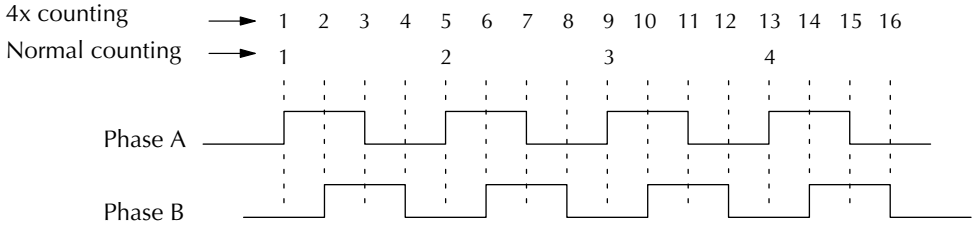
X Input Configuration

The configurable discrete input options for High-Speed Counter Mode are listed in the table below. The section on Mode 60 operation at the end of this appendix describes programming the filter time constants.

Mode 20 Up/Down Counter

Input	Configuration Register	Function	Hex Code Required
X0	V7634	Up counting	0202 (standard, absolute)
			0302 (standard, incremental)
		Phase A	0002 (quadrature, absolute) (default)
			0102 (quadrature, incremental)
			1002 (quadrature, absolute) 4x counting*
			1102 (quadrature, incremental) 4x counting*
X1	V7635	Down counting or Phase B	0000
X2	V7636	Counter Reset (no interrupt)	0007** (default) 0207**
		Counter Reset (with interrupt)	0107** 0307**
		Pulse input	0005
		Filtered input	xx06 (xx = filter time, 0 - 99ms (BCD)
X3	V7637	Pulse input	0005
		Filtered input	xx06 (xx = filter time, 0 - 99ms (BCD) (default)

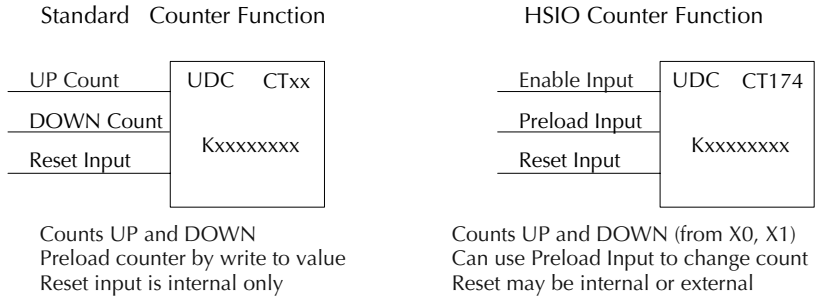
* With this feature, you can count 4 times more with the same encoder.



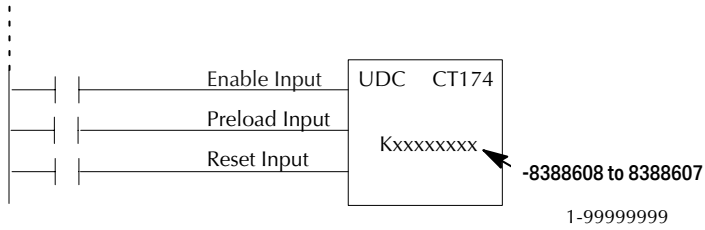
** With the counter reset you have the option of a normal reset or a faster reset. However, the fast reset does not recognize changed preset values during program execution. When '0007' or '0107' are set in V7636 and preset values are changed during program execution, the DL06 recognizes the changed preset values at the time of the reset. When '0207' or '0307' are set in V7636 the CPU does not check for changed preset values, so the DL06 has a faster reset time.

Writing Your Control Program

The mnemonic for the counter is UDC (up-down counter). The DL06 can have up to 128 counters, labeled CT0 through CT177. The quadrature counter in the HSIO circuit is accessed in ladder logic by using UDC CT174. It uses counter registers CT174 and CT175 exclusively when the HSIO mode 20 is active (otherwise, CT174 and CT175 are available for standard counter use). The HSIO counter needs two registers because it is a double-word counter. It has three inputs as shown. The first input is the enable signal, the middle is a preload (write), and the bottom is the reset. The enable input must be on before the counter will count. The enable input must be off during a preload.



The next figure shows the how the HSIO quadrature counter will appear in a ladder program.

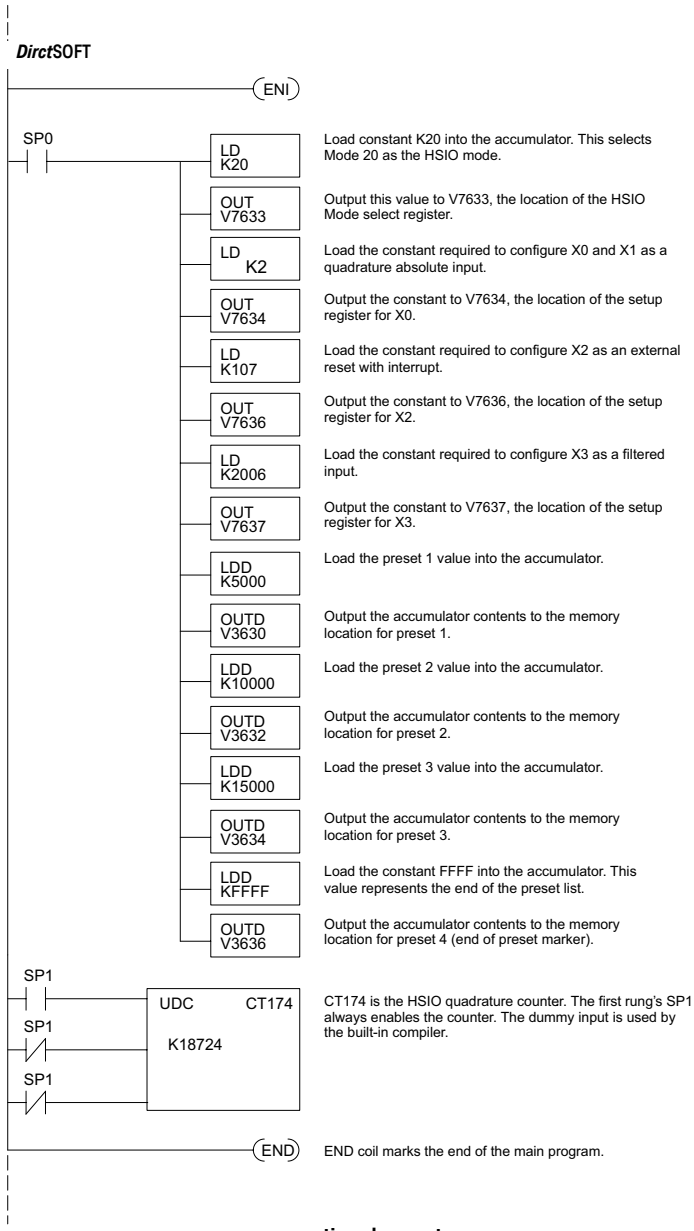


When the enable input is energized, the counter will respond to quadrature pulses on X0 and X1, incrementing or decrementing the counter at CT174 – CT175. The reset input contact behaves in a logical OR fashion with the physical reset input X2. This means the quadrature counter can receive a reset from either the contact(s) on the reset rung in the ladder, OR the external reset X2.

NOTE: *The count is 32 bit BCD sign + magnitude format.*

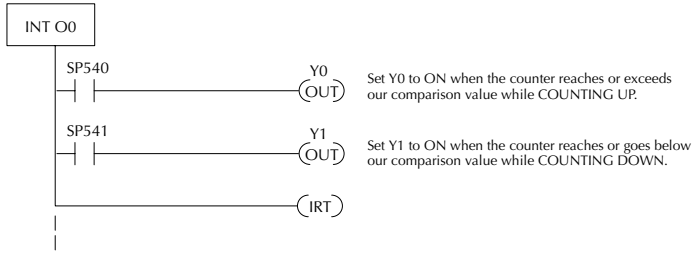
Program Example 1: Quadrature Counting with an Interrupt

Below is a simple example of how quadrature counting with an interrupt can be programmed.



continued on next page

continued from previous page



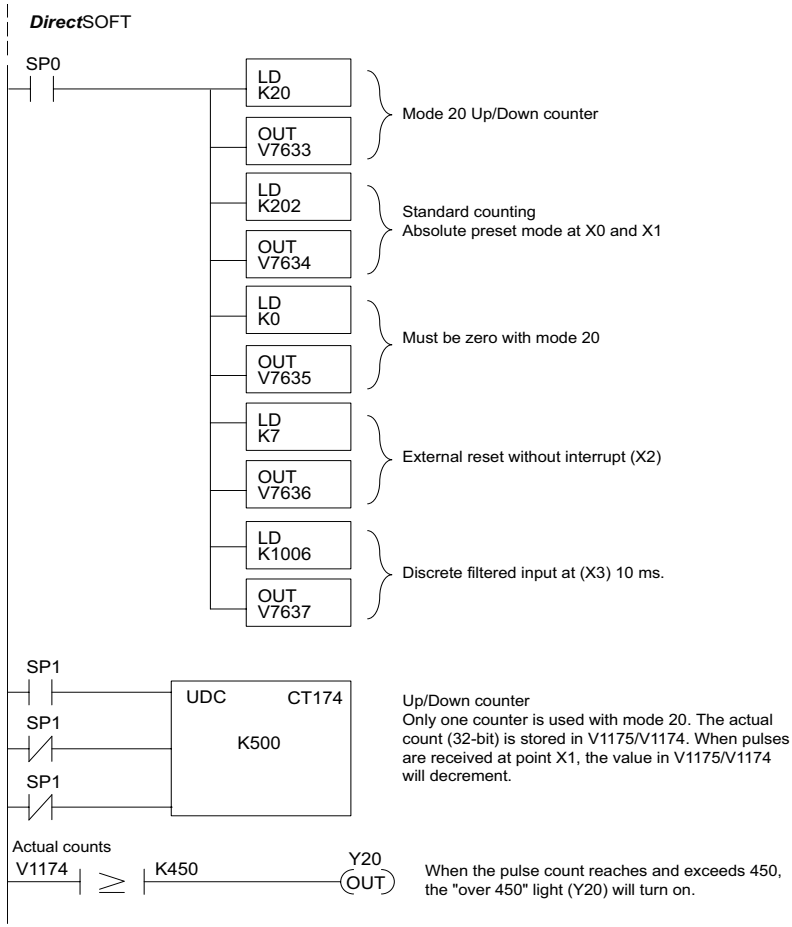
The Load Accumulator instructions have set up the V-memory as required, i.e., 20 in V7633 for the mode and 0202 in V7634 to designate the standard up/down with the absolute preset mode. By placing 0107 in V7636, an external reset for counter CT174 is selected and it will execute interrupt 0 on the rising edge of the reset. Presets for up/down counting have been stored in memory locations V3630 through V3635. The next even numbered location following this has FFFF to indicate we have no more presets.

Program Example 2: Up/Down Counting with Standard Inputs

In this example, there is a conveyor belt "A" that transports bottles to be inspected. During the course of the process, one sensor is keeping track of the bottles that are going onto belt "A" for inspection, and another sensor is keeping track of how many bottles are being removed to the finished product line.

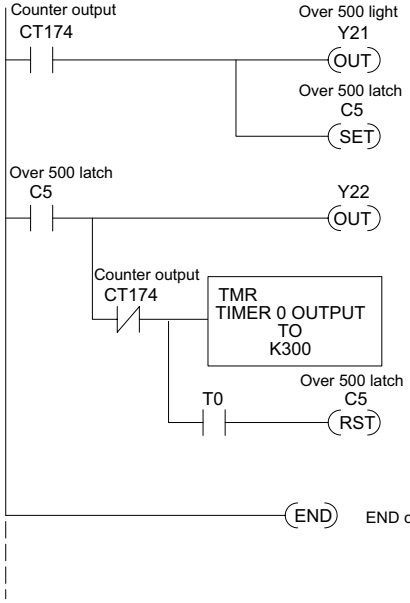
When we have reached 500 bottles in the process, an "over 500" light turns on and a rerouting gate is activated to channel the incoming bottles to conveyor belt "B". The rerouting gate will stay activated for 30 seconds after the conveyor belt "A" contains less than 500 bottles.

The program below shows how ladder logic might be written to handle the job. Note the use of V1174. This memory location stores the current count for CT174 which is used with the DL06.



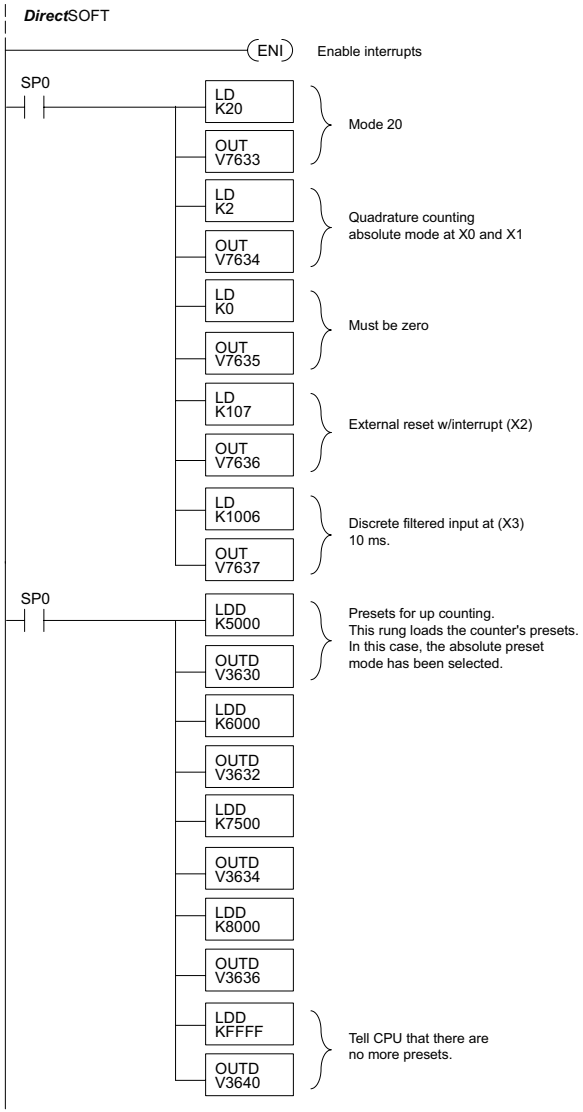
Continued on next page.

continued from previous page



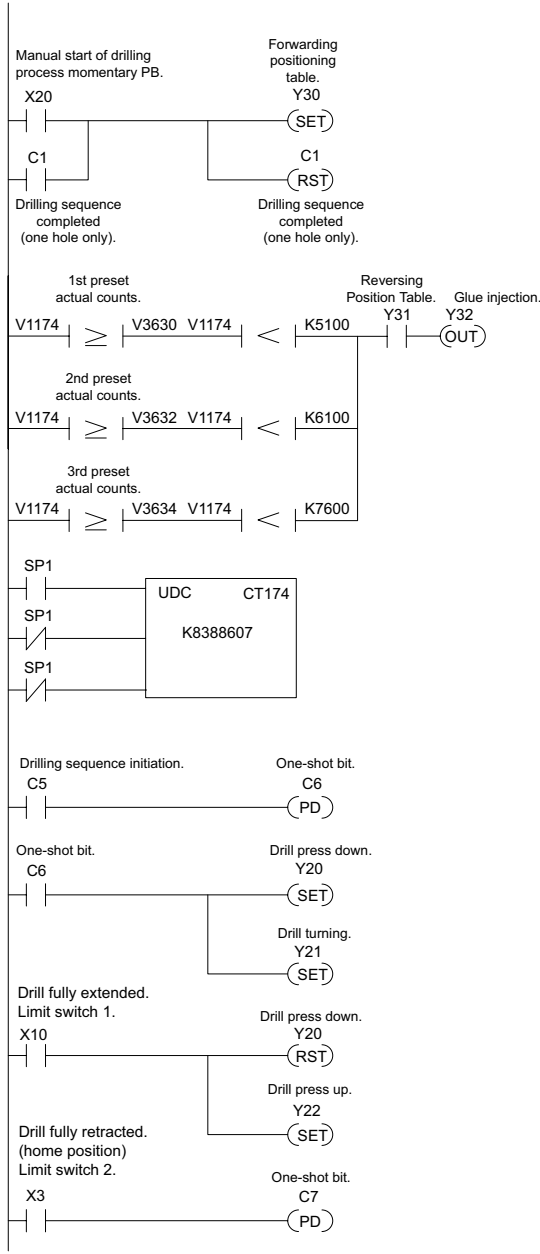
Program Example 3: Quadrature Counting

In this example, a wooden workpiece is being drilled with 3 holes and then the holes are injected with glue for dowels to be inserted at another workstation. A quadrature encoder is connected to a positioning table which is moving a drill press horizontally over the workpiece. The positioning table will stop and the drill press will lower to drill a hole in an exact location. After the three holes are drilled in the workpiece, the positioning table reverses direction and injects glue into the same holes.



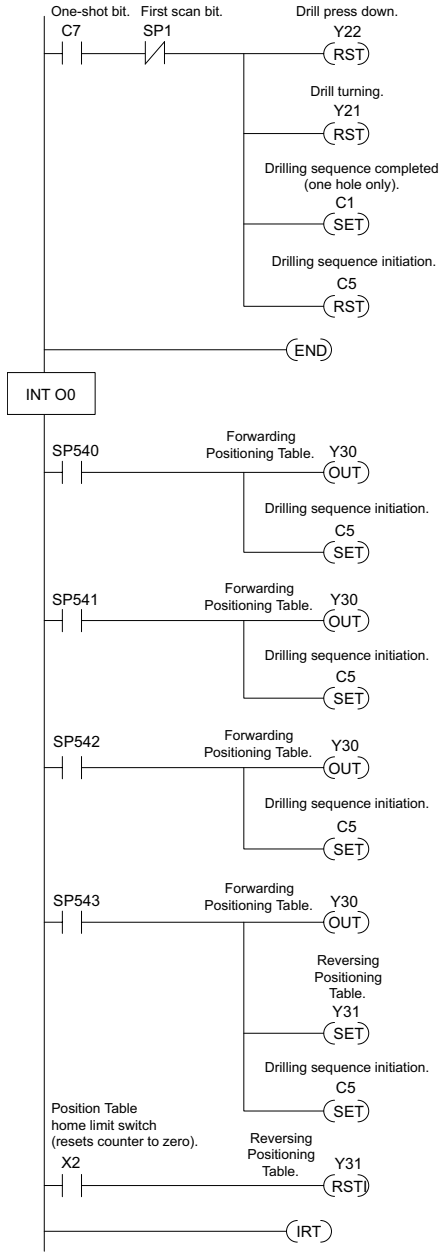
Continued on next page.

Continued from previous page.



Continued on next page.

Continued from previous page.



Troubleshooting Guide for Mode 20

If you're having trouble with Mode 20 operation, please study the following symptoms and possible causes. The most common problems are listed below.

Symptom: The counter does not count.

Possible causes:

1. **Field sensor and wiring** – Verify that the encoder or other field device inputs actually turn on and illuminate the status LEDs for X0 and X1. A standard incremental encoder will visibly, alternately turn on the LEDs for X0 and X1 when rotating slowly (1 RPM). Or, the problem could be due to a sinking-sourcing wiring problem, etc. Remember to check the signal ground connection. Also verify that the pulse on-time, duty cycle, voltage level, and frequency are within the input specifications.
2. **Configuration** – make sure all of the configuration parameters are correct. V7633 must be set to 20, and V7634 must be set to "0002" to enable the Phase A input, and V7635 must be set to "0000" to enable the Phase B input.
3. Stuck in reset – check the input status of the reset input, X2. If X2 is on, the counter will not count because it is being held in reset.
4. **Ladder program** – make sure you are using counter CT174 in your program. The top input is the enable signal for the counter. It must be on before the counter will count. The middle input is the dummy input and must be off for the counter to count. The bottom input is the counter reset, and must be off during counting.

Symptom: The counter counts in the wrong direction (up instead of down, and visa-versa).

Possible causes:

1. **Channel A and B assignment** – It's possible that Channel A and B assignments of the encoder wires is backwards from the desired rotation/counting orientation. Just swap the X0 and X1 inputs, and the counting direction will be reversed.

Symptom: The counter counts up and down but will not reset.

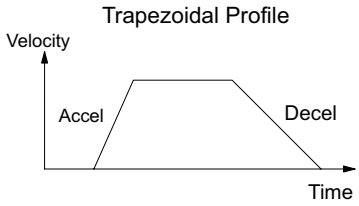
Possible causes:

1. Check the LED status indicator for X2 to make sure it is active when you want a reset. Also verify the configuration register V7636 for X2 is set to 7. Or, if you are using an internal reset, use the status mode of ***Direct***SOFT to monitor the reset input to the counter.

Mode 30: Pulse Output

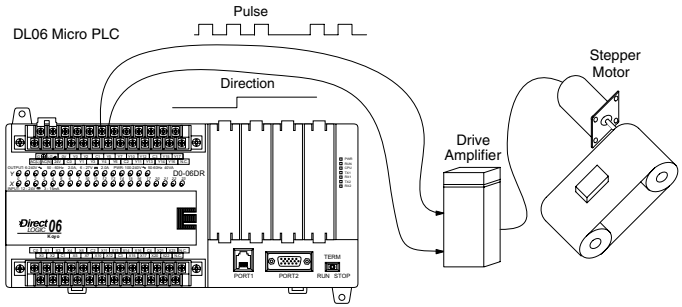
Purpose

The HSIO circuit in Mode 30 generates output pulse trains suitable for open-loop control of a single-axis motion positioning system. It generates pulse (stepper increment) and direction signals which you can connect to motor drive systems and perform various types of motion control. Using Mode 30 Pulse Output, you can select from three profile types detailed later in this appendix:

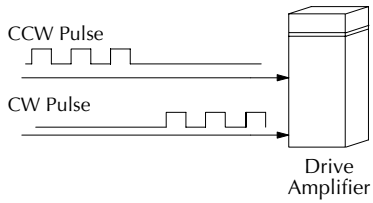


- **Automatic Trapezoidal** – Accel Slope to Target Velocity to Decel Slope
- **Step Trapezoidal** – User defined step acceleration/deceleration and target velocity
- **Velocity Control** – Speed and Direction only

The HSIO circuit becomes a high-speed pulse generator (up to 10 kHz) in Mode 30. By programming acceleration and deceleration values, position and velocity target values, the HSIO function automatically calculates the entire motion profile. The figure below shows the DL06 generating pulse and direction signals to the drive amplifier of a stepper positioning system. The pulses accomplish the profile independently and without interruption to ladder program execution in the CPU.



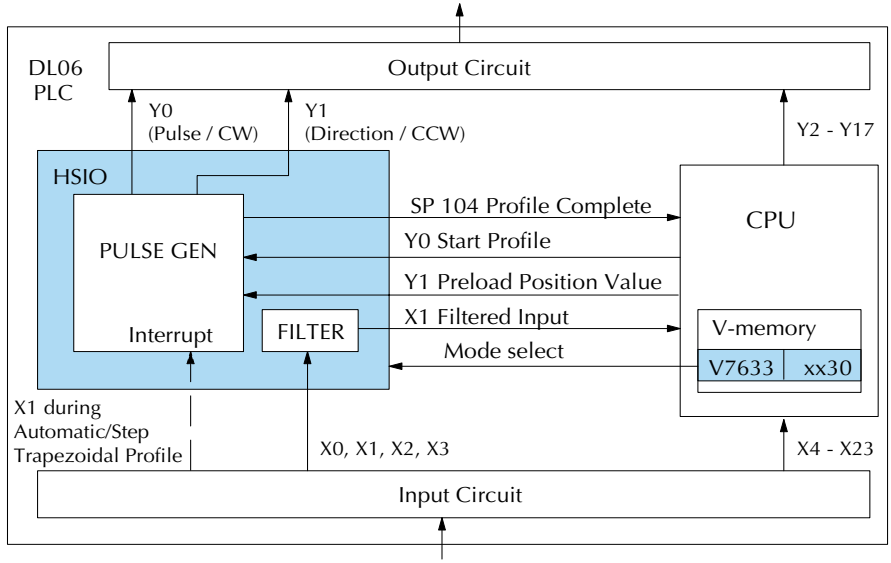
In the figure above, the DL06 generates pulse and direction signals. Each pulse represents the smallest increment of motion to the positioning system (such as one step or micro-step to a stepper system). Alternatively, the HSIO Pulse Output Mode may be configured to deliver counter clock-wise (CCW) and clock-wise (CW) pulse signals as shown to the right.



NOTE: *The pulse output is designed for open loop stepper motor systems. This, plus its minimum velocity of 40 pps, make it unsuitable for servo motor control.*

Functional Block Diagram

The diagram below shows HSIO functionality in Mode 30. When the lower byte of HSIO Mode register V7633 contains a BCD "30", the pulse output capability in the HSIO circuit is enabled. The pulse outputs use Y0 and Y1 terminals on the output connector. Remember that the outputs can only be DC type to operate.



IMPORTANT NOTE: *In Pulse Output Mode, Y0 and Y1 references are redefined or are used differently in two ways. Physical references refer to terminal screws, while logical references refer to I/O references in the ladder program. Please read the items below to understand this very crucial point.*

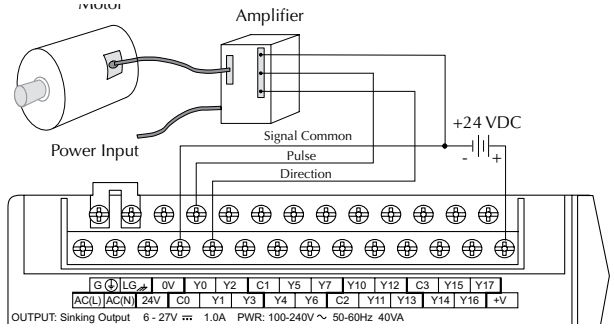
Notice the I/O point assignment and usage in the above diagram:

- X0, X1, X2 and X3 can be filtered inputs or pulse inputs in Pulse Output Mode, and they are available as input contacts to the ladder program.
- X1 behaves as an external interrupt to the pulse generator for automatic/step trapezoidal profiles. In other profile modes, it can be used as a filtered input or pulse input just like X0 (registration mode configuration shown above).
- References "Y0" and "Y1" are used in two different ways. At the discrete output connector, Y0 and Y1 terminals deliver the pulses to the motion system. The ladder program uses logical references Y0 and Y1 to initiate "Start Profile" and "Load Position Value" HSIO functions in Mode 30.

Hopefully, the above discussion will explain why some I/O reference names have dual meanings in Pulse Output Mode. **Please read the remainder of this section with care,** to avoid confusion about which actual I/O function is being discussed.

Wiring Diagram

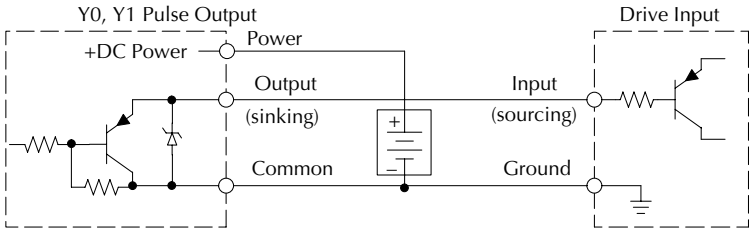
The generalized wiring diagram below shows pulse outputs Y0 and Y1 connected to the drive amplifier inputs of a motion control system.



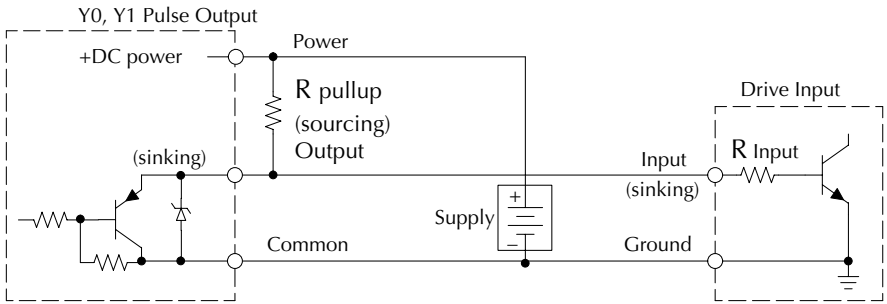
NOTE: *Do not use this drawing to wire your device. This is a general example and is not specific to any PLC model, stepper or encoder. Always refer to the device documentation for proper wiring connections.*

Interfacing to Drive Inputs

The pulse signals from Y0 and Y1 outputs will typically go to drive input circuits as shown above. It will be helpful to locate equivalent circuit schematics of the drive amplifier. The following diagram shows how to interface to a sourcing drive input circuit.



The following circuit shows how to interface to a sinking drive input using a pullup resistor. Please refer to Appendix 2 to learn how to calculate and install R pullup.



Motion Profile Specifications

The motion control profiles generated in Pulse Output Mode have the following specifications:

Motion Control Profile Specifications	
Parameter	**Specification**
Profiles	Automatic Trapezoidal – Accel Slope / Target Velocity / Decel Slope
	Step Trapezoidal - Step Acceleration / Deceleration
	Velocity Control – Speed and Direction only
Position Range	–8388608 to 8388607
Positioning	Absolute / relative command
Velocity Range	40 Hz to 10 kHz
V-memory registers	V3630 to V3652 (Profile Parameter Table)
Current Position	CT174 and CT175 (V1174 and V1175)

Physical I/O Configuration

The configurable discrete I/O options for Pulse Output Mode are listed in the table below. The CPU uses SP 104 contact to sense "profile complete". V7632 is used to select pulse/direction or CW/CCW modes for the pulse outputs. Input X1 is dedicated as the external interrupt for use in registration mode.

Physical I/O Configuration			
Input	**Configuration Register**	**Function**	**Hex Code Required**
–	V7632	Y0 = Pulse Y1 = Direction	0103
		Y0 = CW Pulse Y1 = CCW Pulse	0003 (default)
X0	V7634	pulse input	0005
		filtered input	xx06, xx = filter time, 0-9 (BCD) (default)
X1	V7635	pulse input	0005
		filtered input	xx06, xx = filter time, 0-99 (BCD) (default)
X2	V7636	pulse input	0005
		filtered input	xx06, xx = filter time, 0-99 (BCD) (default)
X3	V7637	pulse input	0005
		filtered input	xx06, xx = filter time, 0-99 (BCD) (default)

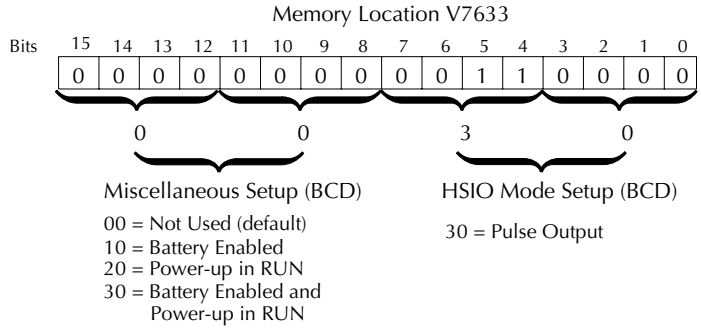
Logical I/O Functions

The following logical I/O references define functions that allow the HSI to communicate with the ladder program.

Logical I/O/ Functions	
Logical I/O	**Function**
SP104	Profile Complete – the HSIO turns on SP104 to the CPU when the profile completes, and it goes back off when Start Profile (Y0) turns on.
X1	External Interrupt - If the interrupt feature is selected for the Automatic Trapezoidal profile or the Step Trapezoidal Profile, the DL06 keeps outputting pulses until X1 turns on. After it is on, the unit outputs the pulses that are defined as the Target position.
Y0	Start Profile – the ladder program turns on Y0 to start motion. If turned off before the move completes, the pulses ramp down and motion stops. Turning it on again will start another profile, unless the current position equals the target position.
Y1	Preload Position Value – if motion is stopped and Start Profile is off, you can load a new value in CT174/CT175, and turn on Y1. At that transition, the value in CT174/CT175 becomes the current position.

Setup for Mode 30

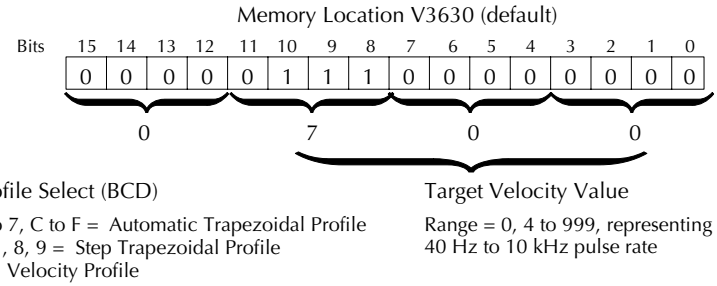
Recall that V7633 is the HSIO Mode Select register. Refer to the diagram below. Use BCD 30 in the lower byte of V7633 to select the High-Speed Counter Mode.



Choose the most convenient method of programming V7633 from the following:

- Include load and out instructions in your ladder program
- ***Direct***SOFT's memory editor
- Use the Handheld Programmer D2–HPP

We recommend using the first method above so that the HSIO setup becomes an integral part of your application program. An example program later in this section shows how to do this.



Profile / Velocity Select Register

The first location in the Profile Parameter Table stores two key pieces of information. The upper four bits (12–15) select the type of profile required. The lower 12 bits (0-11) select the Target Velocity.

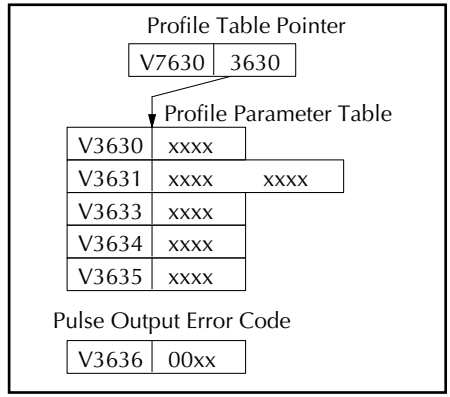
The ladder program must program this location before initiating any of the three profiles. The LD and OUT instruction will write all 16 bits, so be sure to fully specify the full four-digit BCD value for the Profile / Velocity Select Register each time.

The absolute and relative selection determines how the HSIO circuit will interpret your specified target position. Absolute position targets are referenced to zero. Relative position targets are referenced to the current position (previous target position). You may choose whichever reference method is most convenient for your application.

Profile Parameter Table

V7630 is a pointer location which points to the beginning of the Profile Parameter Table. The default starting location for the profile parameter table is V3630. However, you may change this by programming a different value in V7630. Remember to use the LDA (load address) instruction, converting octal into hex.

The HSIO uses the next V-memory register past the bottom of the profile parameter table to indicate profile errors. See the error table at the end of this section for error code definitions.



Automatic Trapezoidal Profile

V-Memory	Function	Range	Units
V3630, bits 12–15	Automatic Trapezoidal Profile without Ending Velocity (Ending Velocity is fixed to 0.)	4=absolute w/o interrupt 5=absolute with interrupt* C=relative w/o interrupt D=relative with interrupt*	–
	Automatic Trapezoidal Profile with Ending Velocity (Use V3637 to set up Ending Velocity.)	6=absolute w/o interrupt 7=absolute with interrupt* E=relative w/o interrupt F=relative with interrupt*	–
V3630, bits 0–11	Target Velocity	0, 4-999 where 0=1000	x 10pps
V3631 / V3632	Target Position**	–8388608 to 8388607	Pulses
V3633	Starting Velocity	4 to 100	x 10pps
V3634	Acceleration Time	2 to 100	x 100ms
V3635	Deceleration Time	2 to 100	x 100ms
V3636	Error Code	(see end of section)	–
V3637	Ending Velocity	4 to 100	x 10pps

* If you select to use interrupt, the DL06 will not start looking for your target count until the interrupt X1 is on.

**To set a negative number, put 8 in the most significant digit. For example: -8388608 is 88388608 in V3631 and V3632.

Step Trapezoidal Profile

V-Memory	Function	Range	Units
V3630, bits 12–15	Step Trapezoidal Profile	0=absolute w/o interrupt 7=absolute with interrupt* 8=relative w/o interrupt 9=relative with interrupt*	–
V3630, bits 0–11	Target Velocity	0, 4-999 where 0=1000	x 10 pps
V3631 / V3632	Target Position**	–8388608 to 8388607	Pulses
V3633	Step 1 Acceleration	4 to 1000	x 10 pps
V3634	Step 1 Distance	1 to 9999	Pulses
V3635	Step 2 Acceleration	4 to 1000	x 10 pps
V3636	Step 2 Distance	1 to 9999	Pulses
V3637	Step 3 Acceleration	4 to 1000	x 10 pps
V3640	Step 3 Distance	1 to 9999	Pulses
V3641	Step 4 Acceleration	4 to 1000	x 10 pps
V3642	Step 4 Distance	1 to 9999	Pulses
V3643	Step 5 Deceleration	4 to 1000	x 10 pps
V3644	Step 5 Distance	1 to 9999	Pulses
V3645	Step 6 Deceleration	4 to 1000	x 10 pps
V3646	Step 6 Distance	1 to 9999	Pulses
V3647	Step 7 Deceleration	4 to 1000	x 10 pps
V3650	Step 7 Distance	1 to 9999	Pulses
V3651	Step 8 Deceleration	4 to 1000	x 10 pps
V3652	Step 8 Distance	1 to 9999	Pulses

* If you select to use interrupt, the DL06 will not start looking for your target count until the interrupt X1 is on

**To set a negative number, put 8 in the most significant digit. For example: -8388608 is 88388608 in V3631 and V3632..

Velocity Control

V-Memory	Function	Range	Units
V3630	Velocity Profile	2000 only	–
V3631 / 3632	Direction Select	0=CW, 80000000=CCW,	Pulses
V3633	Velocity	4 to 1000	x 10 pps
V3636	Error Code	(see end of section)	–

Choosing the Profile Type

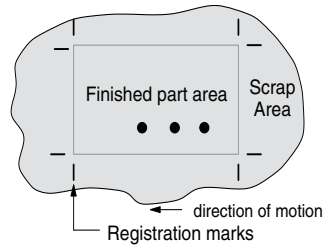
Pulse Output Mode generates three types of motion profiles. Most applications use one type for most moves. However, each move can be different if required.

- Automatic Trapezoidal – Accel Slope to Target Velocity to Decel Slope
- Step Trapezoidal – Velocity to Position Control on Interrupt
- Velocity Control – Speed and Direction only

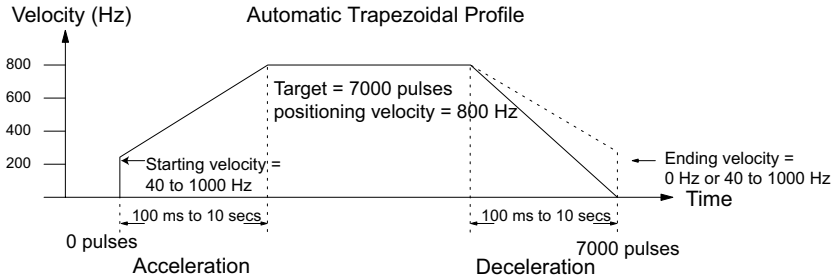
Automatic Trapezoidal Profile Defined

The automatic trapezoidal profile is the most common positioning profile. It moves the load to a pre-defined target position by creating a move profile. The acceleration slope is applied at the starting position. The deceleration slope is applied backwards from the target position. The remainder of the move in the middle is spent traveling at a defined velocity.

Registration profiles solve a class of motion control problems. In some applications, product material in work moves past a work tool such as a drill station. Shown to the right, registration marks on the scrap area of the work-piece allow a machine tool to register its position relative to the rectangle, to drill properly.



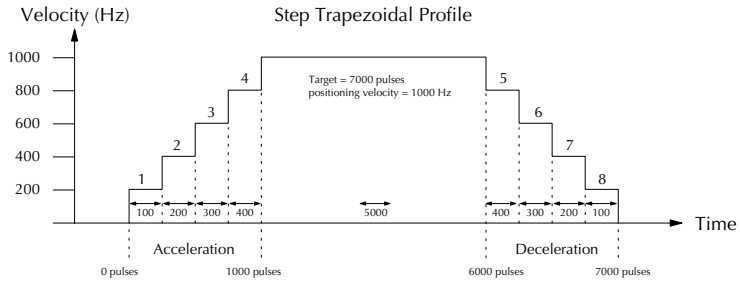
Home search moves allow open-loop motion systems to re-calibrate (preload) the current position value at powerup.



The user determines the starting velocity, the acceleration/deceleration times, and the total number of pulses. The CPU computes the profile from these inputs.

Step Trapezoidal Profiles Defined

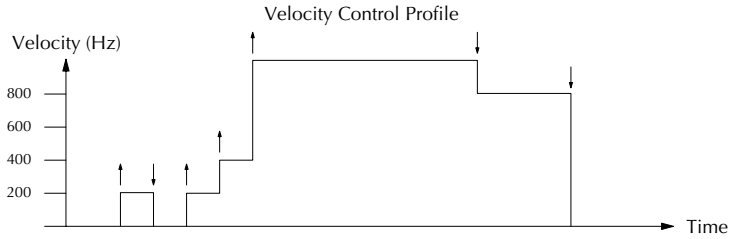
The step trapezoidal profile is a combination of velocity and position control modes. The move begins by accelerating to a programmed velocity. The velocity is sustained and the move is of indefinite duration. When an external interrupt signal occurs (due to registration sensing), the profile switches from velocity to position control. The move ends by continuing motion a pre-defined distance past the interrupt point (such as a drill hole location). The deceleration ramp is applied in advance of the target position.



Define steps 1 through 4 for gradual acceleration to the target velocity and define steps 5 through 8 for gradual deceleration from the target velocity. This type of profile is appropriate for applications involving large stepper motors and/or large inertia loads. It can, however, be used to provide gradual ramping in applications involving smaller motors and loads.

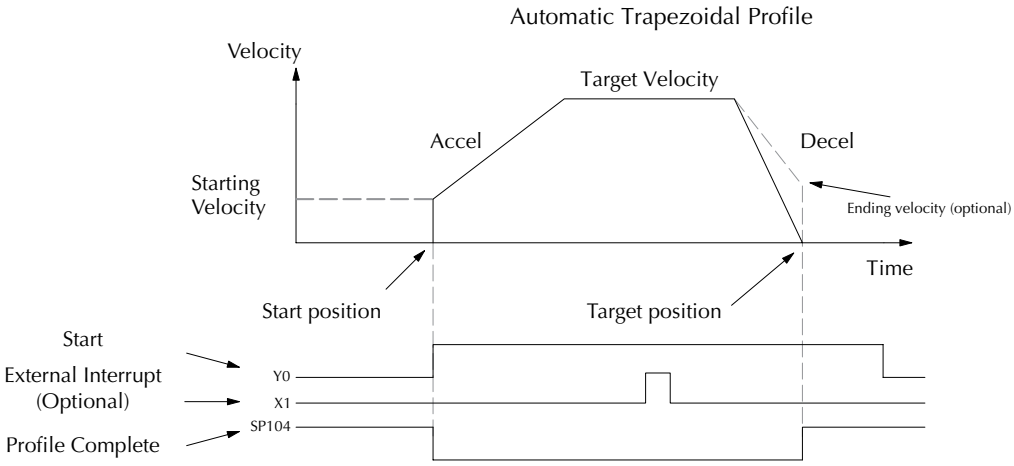
Velocity Control Defined

The Velocity Control defines only the direction and speed of motion. There is no target position specified, so the move can be of indefinite length. Only the first velocity value needs to be defined. The remaining velocity values can be created while motion is in progress. Arrows in the profile shown indicate velocity changes.



Automatic Trapezoidal Profile Operation

Starting velocities must be within the range of 40 pps to 1k pps. The remainder of the profile parameters are in the profile parameter table.



The time line of signal traces below the profile indicates the order of events. The HSIO uses logical output Y0 as the Start input to the HSIO, which starts the profile. Immediately the HSIO turns off the Profile Complete signal (SP104), so the ladder program can monitor the progress of the move. Typically, a ladder program will monitor this bit so it knows when to initiate the next profile move.

You can also use the external interrupt (X1). Once the external interrupt feature is selected for the profile, the DL06 keeps outputting the pulses until X1 turns on. Then, the DL06 outputs the pulses defined as the target position.

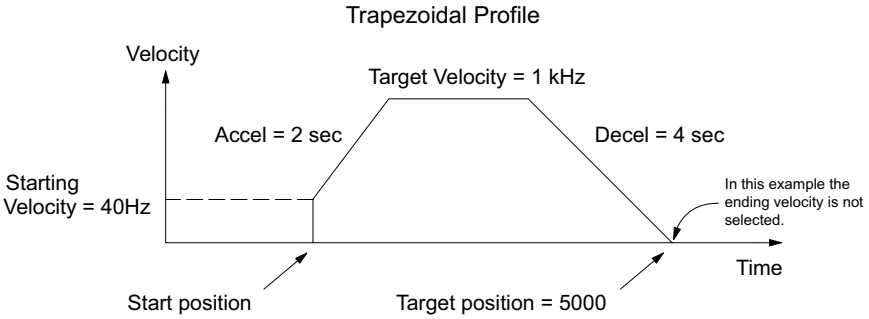
If you are familiar with motion control, you'll notice that we do not have to specify the direction of the move. The HSIO function examines the target position relative to the current position, and automatically outputs the correct direction information to the motor drive.

Notice that the motion accelerates immediately to the starting velocity. This segment is useful in stepper systems so we can jump past low speed areas when low-torque problems or a resonant point in the motor might cause a stall. (When a stepper motor stalls, we have lost the position of the load in open-loop positioning systems). However, it is preferable not to make the starting velocity too large, because the stepper motor will also *slip* some pulses due to the inertia of the system. You can also set up the ending velocity for the same reason.

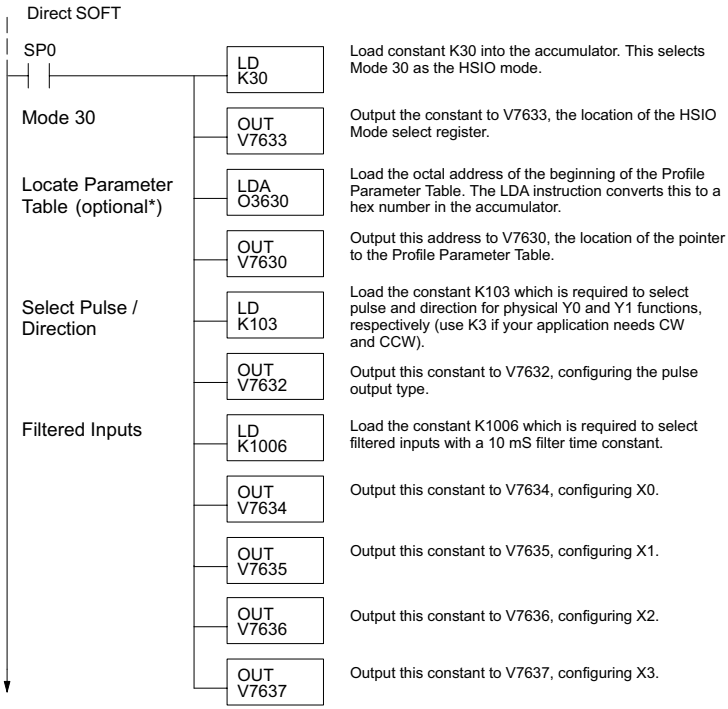
When you need to change the current position value, use logical Y1 output coil to load a new value into the HSIO counter. If the ladder program loads a new value in CT174/CT175 (V1174/V1175), then energizing Y1 will copy that value into the HSIO circuit counter. This must occur before the profile begins, because the HSIO ignores Y1 during motion.

Program Example 1: Automatic Trapezoidal Profile without External Interrupt

The Automatic Trapezoidal Profile we want to perform is drawn and labeled in the following figure. It consists of a non-zero starting velocity, and moderate target velocity.



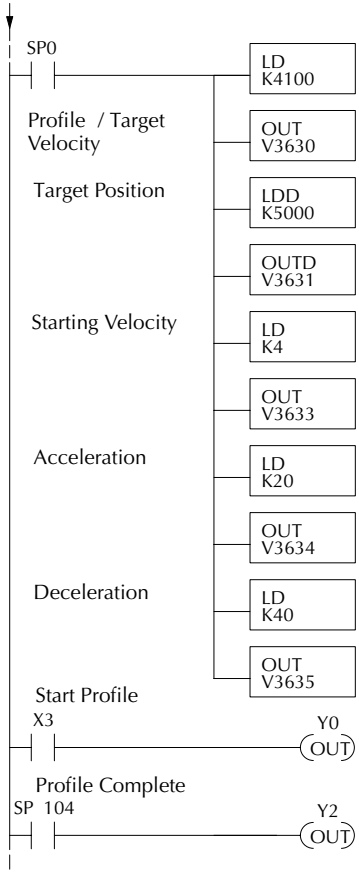
The following program will realize the profile drawn above, when executed. The beginning of the program contains all the necessary setup parameters for Pulse Output Mode 30. We only have to do this once in the program, so we use first-scan contact SP0 to trigger the setup.



* If the Locate Parameter Table is not used you must remove both the LDA function and the OUT function below it.

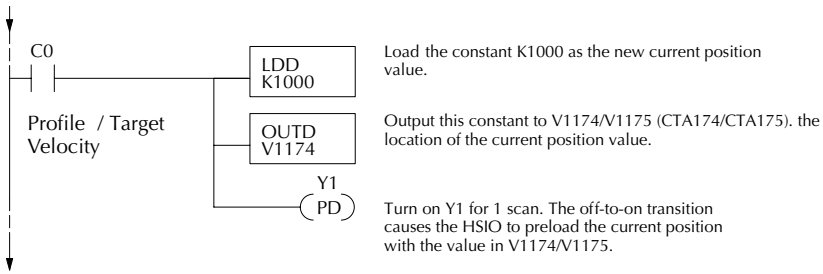
Continued on next page.

Continued from previous page.



Preload Position Value

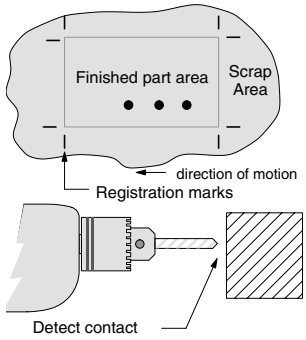
At any time you can write (preload) a new position into the current position value. This is often done after a home search (see the registration example programs).



Program Example 2: Automatic Trapezoidal Profile with External Interrupt

Registration Applications:

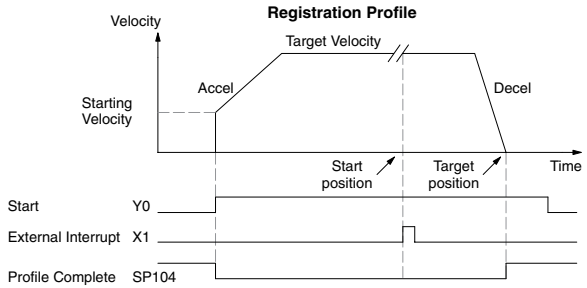
1. In a typical application shown to the right, product material in work moves past a work tool such as a drill. Registration marks on the scrap area of the work-piece allow a machine tool to register its position relative to the rectangle, to drill properly.



2. In other examples of registration, the work piece is stationary and the tool moves. A drill bit may approach the surface of a part in work, preparing to drill a hole of precise depth. However, the drill bit length gradually decreases due to tool wear. A method to overcome this is to detect the moment the drill makes contact with the surface of the part each time a part is drilled. The bit can then drill a constant depth after making contact with the part's surface.

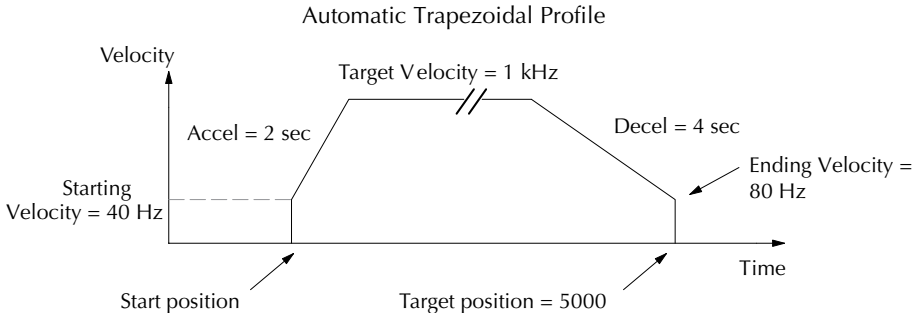
3. The home search move allows a motion system to calibrate its position on startup. In this case, the positioning system makes an indefinite move and waits for the load to pass by a home limit switch. This creates an interrupt at the moment when the load is in a known position. We then stop motion and preload the position value with a number which equates to the physical "home position".

When an interrupt pulse occurs on physical input X1, the starting position is declared to be the present count (current load position). The velocity control switches to position control, moving the load to the target position. Note that the minimum starting velocity is 40 pps. This instantaneous velocity accommodates stepper motors that can stall at low speeds.

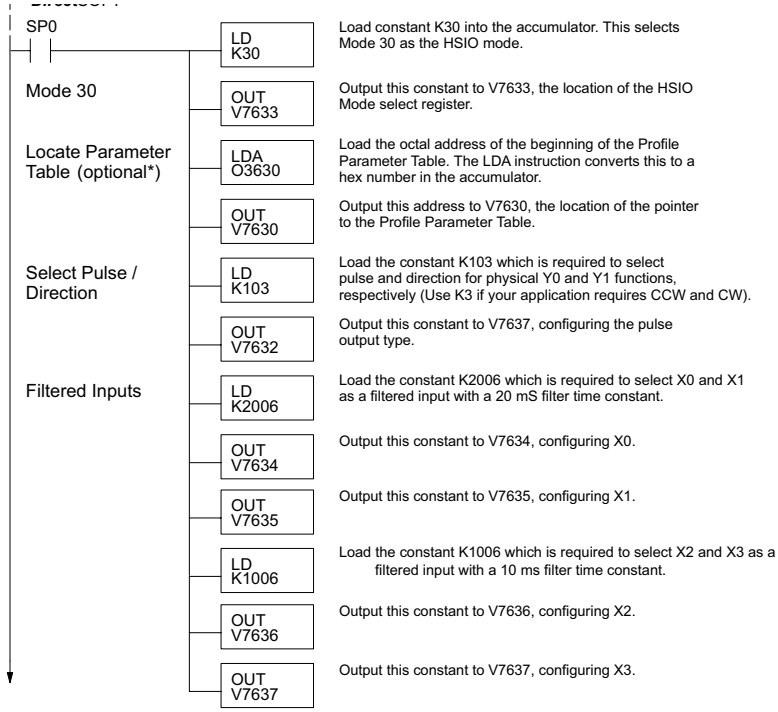


The time line of signal traces below the profile indicates the order of events. The CPU uses logical output Y0 to start the profile. Immediately the HSIO turns off the Profile Complete signal (SP104), so the ladder program can monitor the move's completion by sensing the signal's on state.

The Automatic Trapezoidal profile we want to perform is drawn and labeled in the following figure. It consists of a non-zero starting velocity, and moderate target velocity.

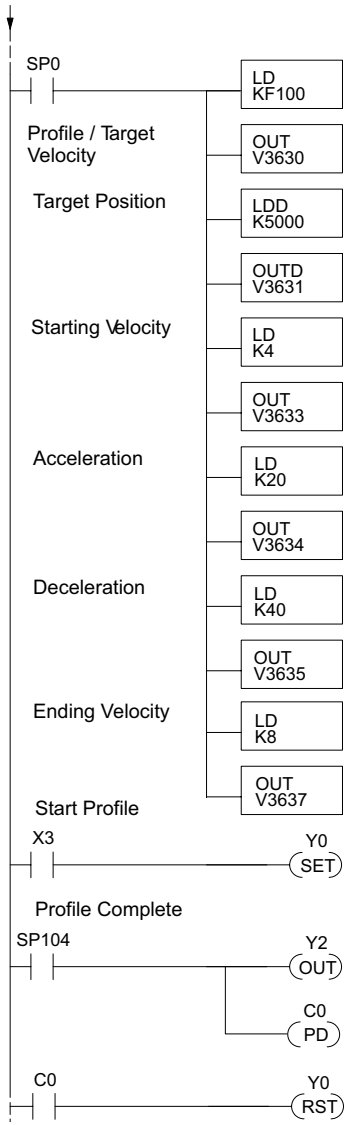


The following program will realize the profile drawn above, when executed. The first program rung contains all the necessary setup parameters. We only have to do this once in the program, so we use first-scan contact SP0 to trigger the setup.



* If the Locate Parameter Table is not used you must remove both the LDA function and the OUT function below it.

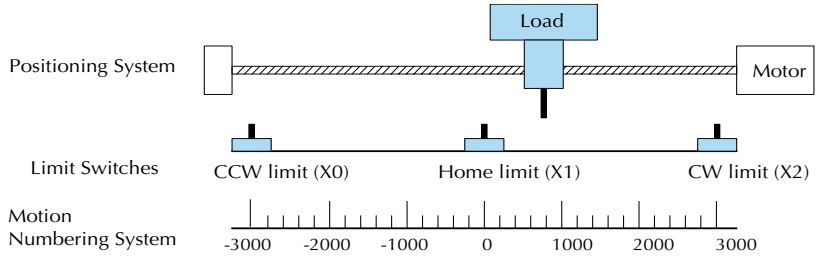
Continued from previous page



The profile will begin when the start input (X3) is given. Then the motion begins an indefinite move, which lasts until an external interrupt on X1 occurs. Then the motion continues on for 5000 more pulses before stopping.

Program Example 3: Automatic Trapezoidal Profile with Home Search

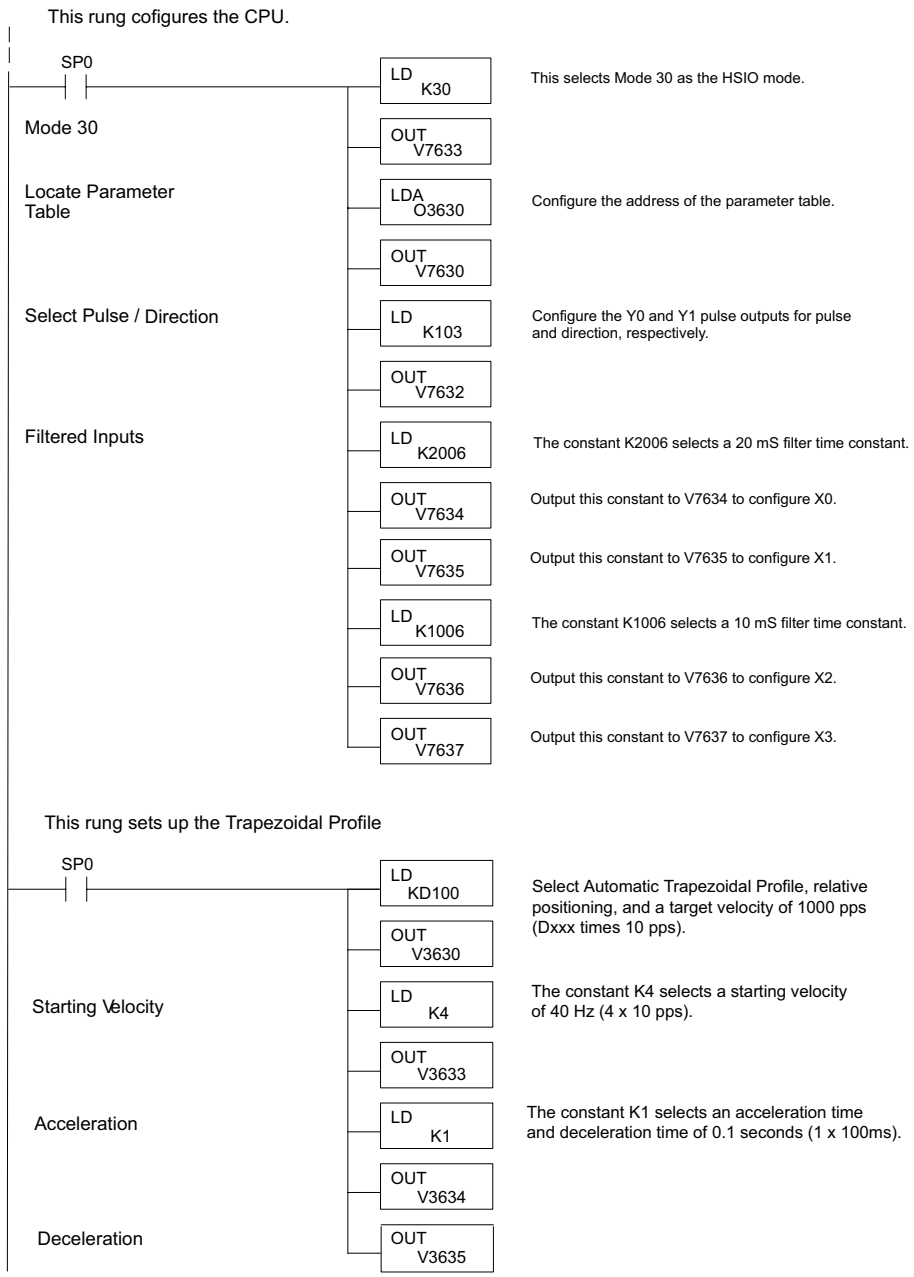
One of the more challenging aspects of motion control is the establishment of actual position at powerup. This is especially true for open-loop systems which do not have a position feedback device. However, a simple limit switch located at an exact location on the positioning mechanism can provide "position feedback" at one point. For most stepper control systems, this method is a good and economical solution.



In the drawing above, the load moves left or right depending on the CW/CCW direction of motor rotation. The PLC ladder program senses the CW and CCW limit switches to stop the motor, before the load moves out-of-bounds and damages the machine. The home limit switch is used at powerup to establish the actual position. The numbering system is arbitrary, depending on a machine's engineering units.

At powerup, we do not know whether the load is located to the left or to the right of the home limit switch. Therefore, we will initiate a home search profile, using the registration mode. The home limit switch is wired to X1, causing the interrupt. The example, beginning on the next page, preferentially starts in one direction only (CW), and pulses until it reaches the first Overtravel Limit (CW Limit). It will ignore the Home Switch if it passes it. This means that Homing is always accomplished from the same direction for better consistency.

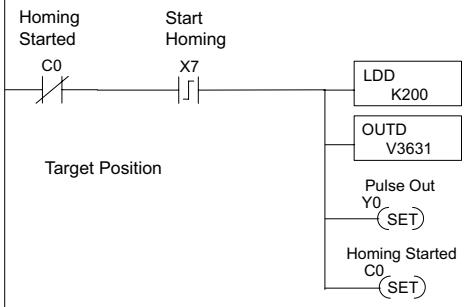
The CPU will then reverse the direction of pulses (loads 80000200 into V3631) and travel away from the CW Limit until it hits the Home Switch, then travels past it and reverse slowly back to the Home Switch. A value of 0 will be loaded to V1174 (CTA174) to represent the Home position.



Continued on next page

Continued from previous page

"Start Homing" is assigned as X7. This will set Y0, which starts the Pulse Output. Pulses will continue until the CCW Limit is reached.

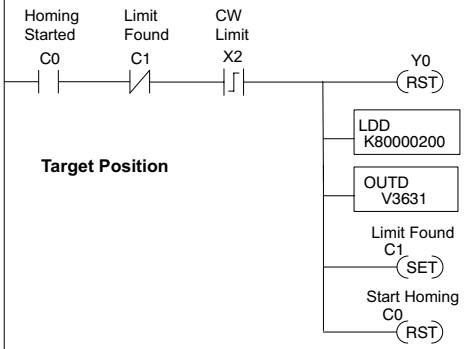


This constant K200 selects the target position 200 pulses (CW direction).

Turn on Pulse Out, searching for the CW Limit.

Set C0 to indicate the CW Limit search has begun.

Once the CW Limit is found, the DL06 will stop the Pulse Output, load a negative value, thus reversing direction. It does this by LDD K80000200 into V3631. The "8" in the left-most position of the value loaded (8xxxxxxx) will cause Y1 to turn on. This is how the PLC reverses direction.



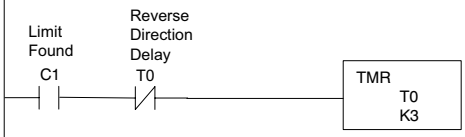
Turn off Start Profile.

The constant -K200 selects a target position of -200 pulses, which is in the CCW direction.

The CW limit has been reached.

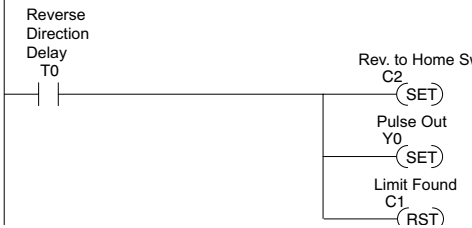
CW Limit was found, reset C0.

This rung will activate timer, T0, for a short 0.3 second delay. When T0 times out, the next rung is activated.



Add a timer to create a short delay before reversing the motor.

The Pulse Output is activated again, but in the reverse direction.



Set C2 to indicate the CCW home search has begun.

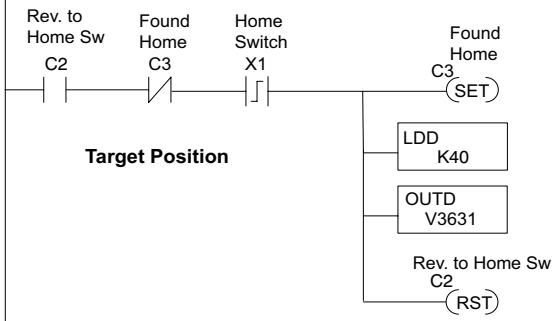
Turn on Pulse Out again.

Reset C1.

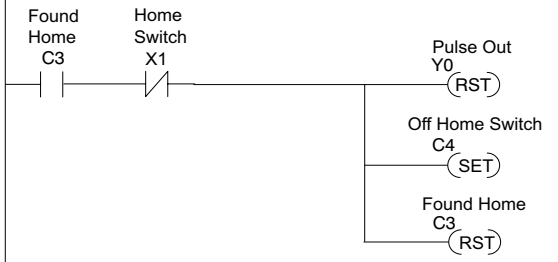
Continued on next page

Continued from previous page

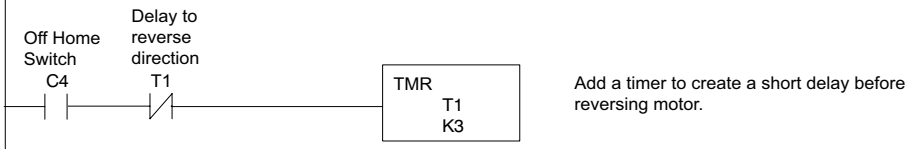
A positve position is loaded into V3631 to slowly move off the X1 Home Switch. Since the position is small, the move will be in the Accel portion, and will be modest speed.
Notice that the value loaded is not in the form of 8xxxxxxx.



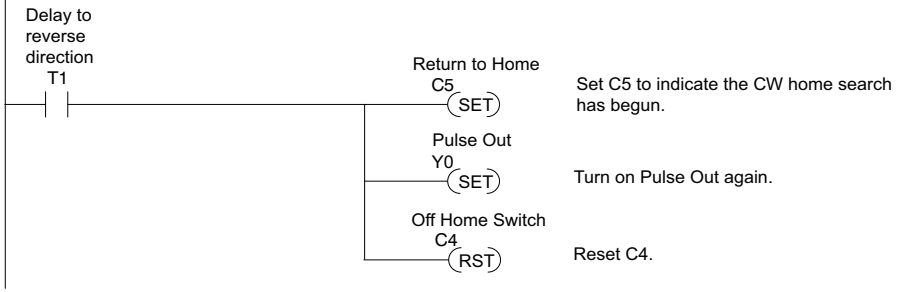
Once the Home Switch, X1, is deactivated, the Pulse Output is stopped.



This rung adds a short delay of 0.3 seconds to delay the reversing of the motor.

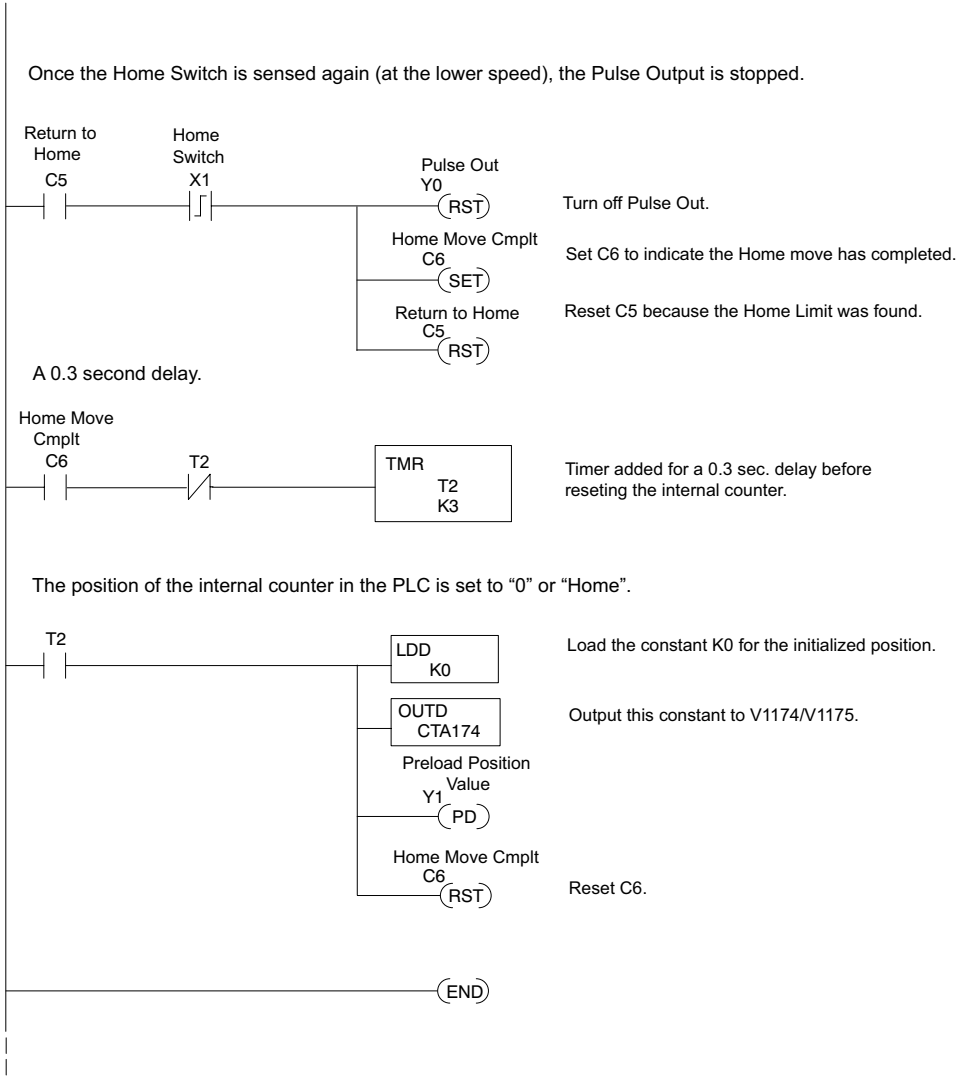


The Pulse Output is now activated again, but in the direction the motor was initially started.



Continued on next page

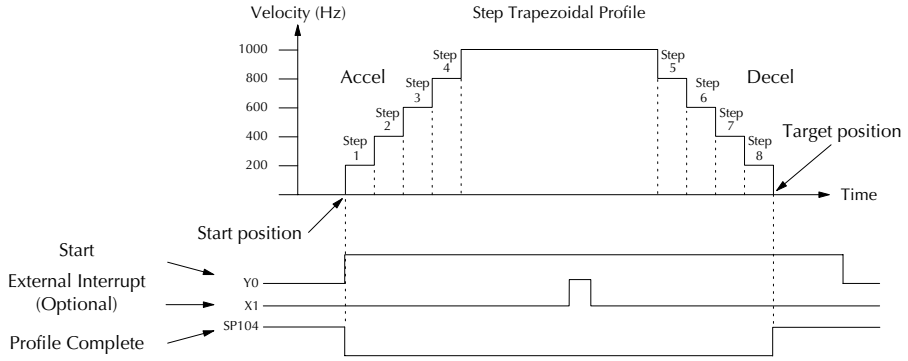
Continued from previous page



The home search profile will execute specific parts of the program, based on the order of detection of the limit switches. Ladder logic sets C0 to initiate a home search in the CW direction. If the CW limit is encountered, the program searches for home in the CCW direction, passes it slightly, and does the final CW search for home. After reaching home, the last ladder rung preloads the current position to “0”.

Step Trapezoidal Profile Operation

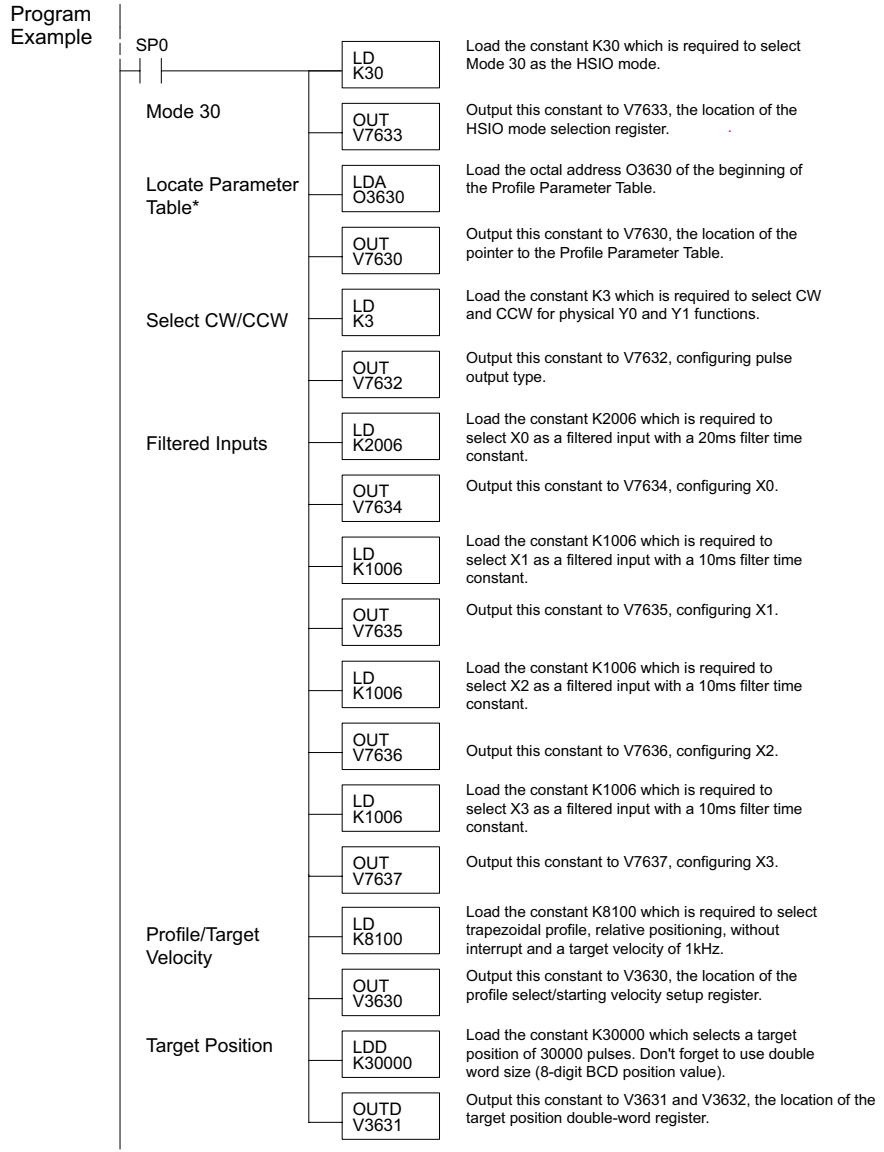
With this step trapezoidal profile, you can control the acceleration and deceleration slopes as you want.



The time line of signal traces below the profile indicates the order of events. The HSIO uses logical output Y0 as the start input to the HSIO, which starts the profile. Immediately, the HSIO turns off the Profile Complete signal (SP104), so the ladder program can monitor the progress of the move. Typically, a ladder program will monitor this bit so it knows when to initiate the next profile move. You can also use the external interrupt (X1). Once the external interrupt feature selected for the profile, the DL06 keeps outputting the pulses until X1 turns on. Then the DL06 outputs the pulses defined as the target position.

Each acceleration and deceleration slope consists of 4 steps. You can set up the velocity and the distance (number of pulses) of each step. You don't need to use all 4 steps of each slope. For instance, if you want to use only 2 steps, just set zero to the velocity and the distance of the 3rd and 4th step. If the acceleration slope and the deceleration slope are identical, you can just put zero into all the velocity and the distance parameters for the deceleration slope.

Program Example 4: Step Trapezoidal Profile

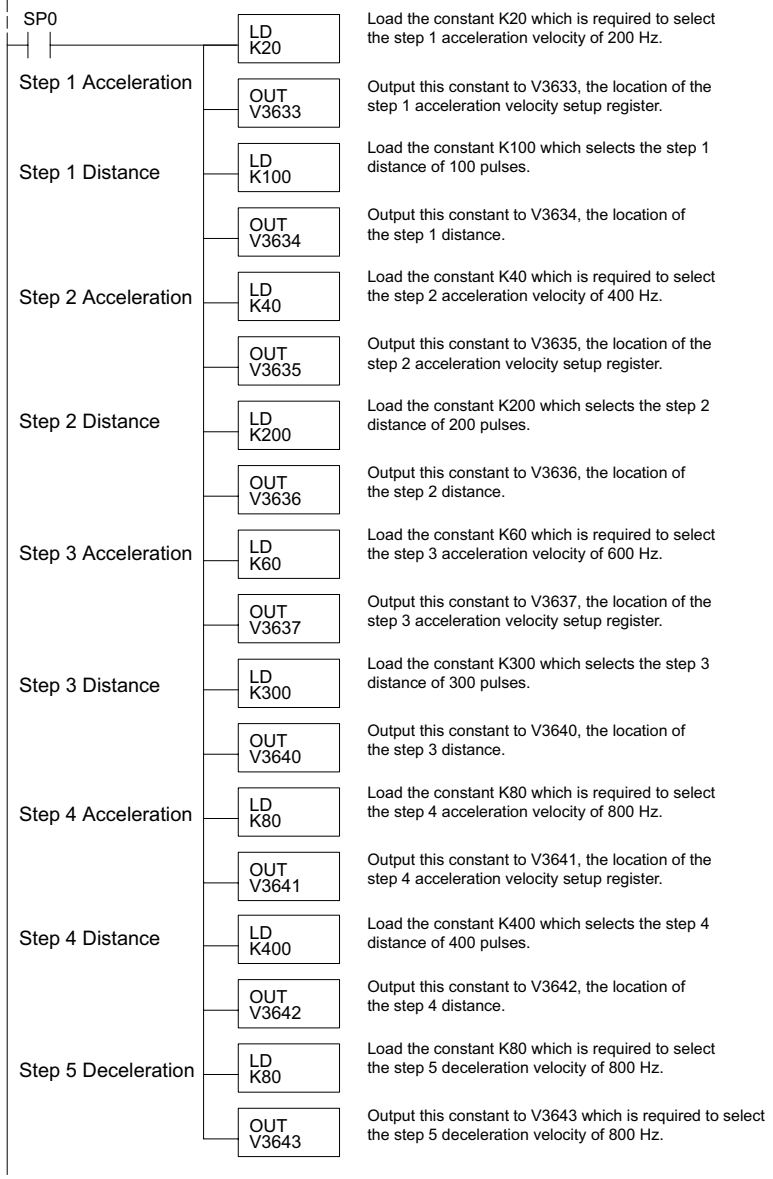


* If the Locate Parameter Table is not used you must remove both the LDA function and the OUT function below it.

Continued on next page

Continued from previous page

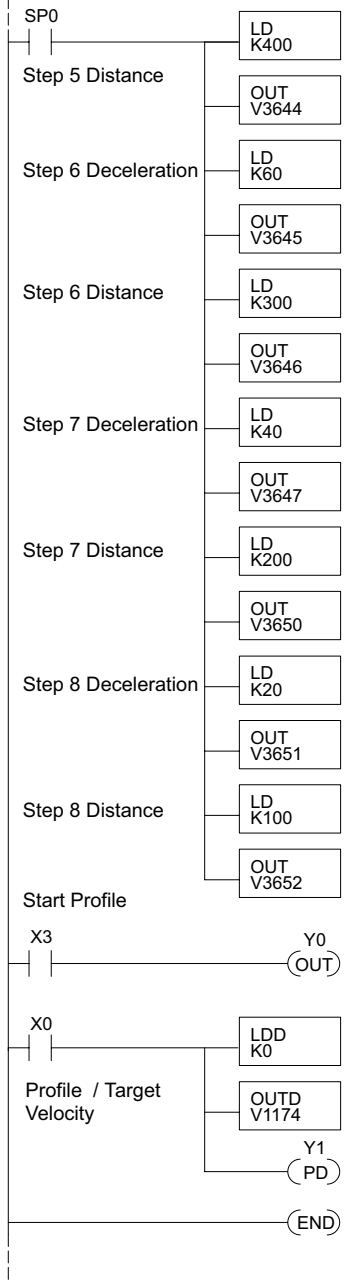
Program Example



Continued on next page

Continued from previous page

Program Example



Load the constant K400 which selects the step 5 distance of 400 pulses.

Output this constant to V3644, the location of the step 5 distance.

Load the constant K60 which is required to select the step 6 deceleration velocity of 600 Hz.

Output this constant to V3645, which is required to select the step 6 deceleration velocity of 600 Hz.

Load the constant K300 which selects the step 6 distance of 300 pulses.

Output this constant to V3646, the location of the step 6 distance.

Load the constant K40 which is required to select the step 6 deceleration velocity of 400 Hz.

Output this constant to V3647, which is required to select the step 6 deceleration velocity of 400 Hz.

Load the constant K200 which selects the step 7 distance of 200 pulses.

Output this constant to V3650, the location of the step 7 distance.

Load the constant K20 which is required to select the step 8 deceleration velocity of 200 Hz.

Output this constant to V3651, which is required to select the step 8 deceleration velocity of 200 Hz.

Load the constant K100 which selects the step 8 distance of 100 pulses.

Output this constant to V3652, the location of the step 8 distance.

We use a spare filtered input to allow the operator to start the profile. When the operator turns X3 ON, logical output Y0 starts the profile. The operator must maintain X3 on until the profile is complete.

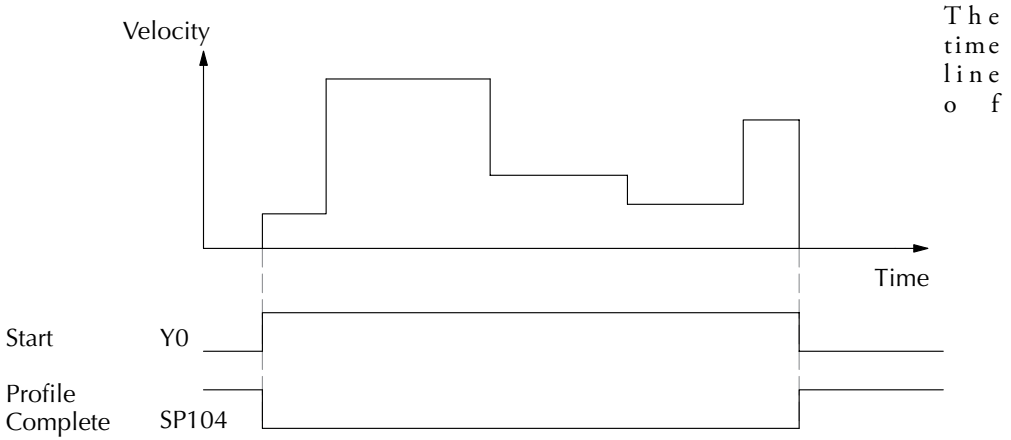
Load value.

Output this constant to V1174/V1175 (CT174/CT175), the location of the current position value.

Turn on Y1 for 1 scan. The of-to-on transition causes the HSIO to preload the current position with the value in V1174/V1175.

Velocity Profile Operation

The velocity profile is best suited for applications which involve motion but do not require moves to specific points. Conveyor speed control is a typical example.



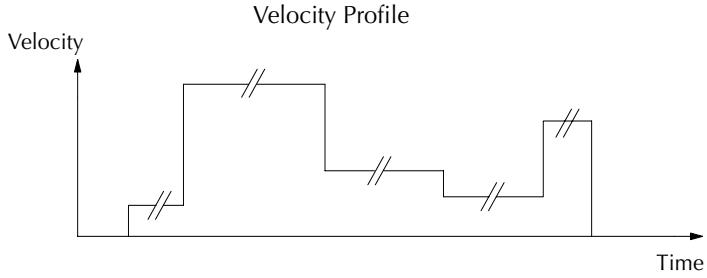
The timeline of signal traces below the profile indicates the order of events. Assuming the velocity is set greater than zero, motion begins when the Start input (Y0) energizes. Since there is no end position target, the profile is considered in progress as long as the Start input remains active. The profile complete logical input to ladder logic (X0) correlates directly to the Start input status when velocity profiles are in use.

While the Start input is active, the ladder program can command a velocity change by writing a new value to the velocity register (V3633 by default). The full speed range of 40 Hz to 10 kHz is available. Notice from the drawing that there are no acceleration or deceleration ramps between velocity updates. This is how velocity profiling works with the HSIO. However, the ladder program can command more gradual velocity changes by incrementing or decrementing the velocity value more slowly. A counter or timer can be useful in creating your own acceleration/deceleration ramps. Unless the load must do a very complex move, it is easier to let the HSIO function generate the accel/decel ramps by selecting the trapezoidal or registration profiles, instead.

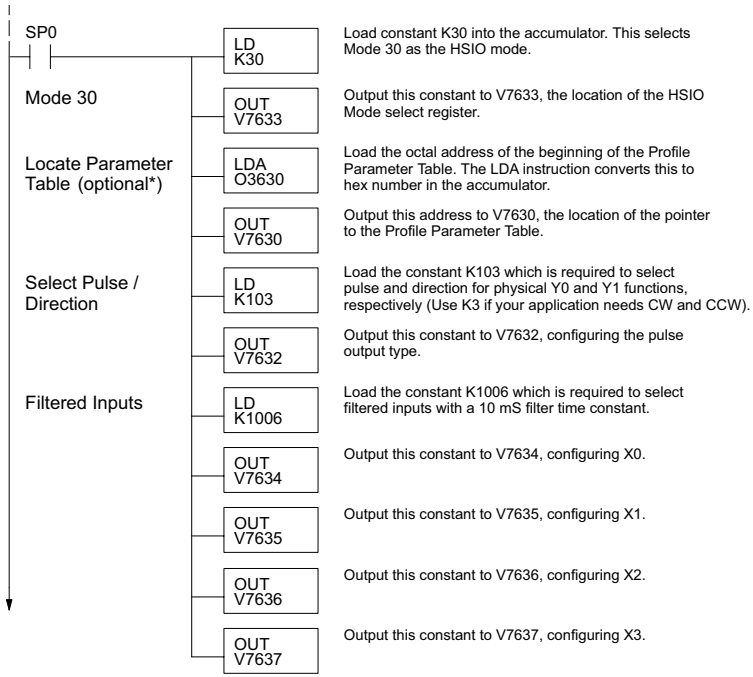
Unlike the trapezoidal and registration profiles, you must specify the desired direction of travel with velocity profiles. Load the direction select register (V3631/V3632 by default) with 8000 0000 hex for CCW direction, or 0 for CW direction.

Program Example 5: Velocity Profile

The velocity profile we want to perform is drawn and labeled in the following figure. Each velocity segment is of indefinite length. The velocity only changes when ladder logic (or other device writing to V-memory) updates the velocity parameter.

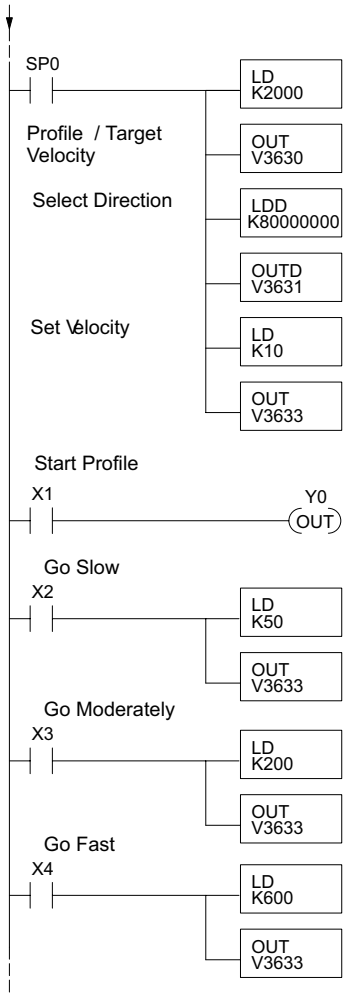


The following program uses dedicated discrete inputs to load in new velocity values. This program is fun to try, because you can create an infinite variety of profiles with just two or three input switches. The intent is to turn on only one of X2, X3, or X4 at a time. The beginning of the program contains all the necessary setup parameters for Pulse Output Mode 30. We only have to do this once in the program, so we use first-scan contact SP0 to trigger the setup.



* If the Locate Parameter Table is not used you must remove both the LDA function and the OUT function below it.

Program Example Cont'd



Load the constant K2000 which is required to select Velocity Profile. This data word contains no velocity information in the case of velocity mode.

Output this constant to V3630, the location of the Profile Select setup register.

Load the constant K80000000 which selects CCW direction for Velocity Profiles. Don't forget to use double word size (8-digit BCD position value).

Output this constant to V3631 and V3632, the location of the Target Position double-word register.

Load the constant K10 which is required to select an initial velocity of 100 pps (uses x10 multiplier).

Output this constant to V3633, the location of the velocity parameter register. After the program is running, we can write here again, using discrete input switches.

We use a spare filtered input to allow the operator to start the profile. When the operator turns X1 ON, logical output Y0 starts the profile. The operator must maintain X1 on until the profile is complete.

Load the constant K50 which is required to select a velocity of 500 pps when the operator closes X2.

Output this constant to V3633, the location of the velocity parameter register. The speed will change immediately.

Load the constant K200 which is required to select a velocity of 2000 pps when the operator closes X3.

Output this constant to V3633, the location of the velocity parameter register. The speed will change immediately.

Load the constant K600 which is required to select a velocity of 6000 pps when the operator closes X4.

Output this constant to V3633, the location of the velocity parameter register. The speed will change immediately.

Automatic Trapezoidal Profile Error Codes

The Profile Parameter Table starting at V3630 (default location) defines the profile. Certain numbers will result in an error when the HSIO attempts to use the parameters to execute a move profile. When an error occurs, the HSIO writes an error code in V3636.

Most errors can be corrected by rechecking the Profile Parameter Table values. The error is automatically cleared at powerup and at Program-to-Run Mode transitions.

Error Code	Error Description
0000	No error
0010	Requested profile type code is invalid (must use 4 to 6 or C to F)
0011	Interrupt is selected for absolute mode
0020	Target Velocity is not in BCD
0021	Target Velocity is specified to be less than 40 pps
0022	Target Velocity is specified to be greater than 10,000 pps
0030	Target Position value is not in BCD
0031	Target Position value is zero
0032	Direction Select is not 0 or 80000000.
0040	Starting Velocity is not in BCD
0041	Starting Velocity is specified to be less than 40 pps
0042	Starting Velocity is specified to be greater than 10,000 pps
0050	Acceleration Time is not in BCD
0051	Acceleration Time is zero
0052	Acceleration Time is greater than 10 seconds
0060	Deceleration Time is not in BCD
0061	Deceleration Time is zero
0062	Deceleration Time is greater than 10 seconds
0070	Ending Velocity is not BCD
0071	Ending Velocity is specified to be less than 40 pps
0072	Ending Velocity is specified to be greater than 1,000 pps
0073	Ending Velocity is specified to be greater than Target Velocity

Troubleshooting Guide for Mode 30

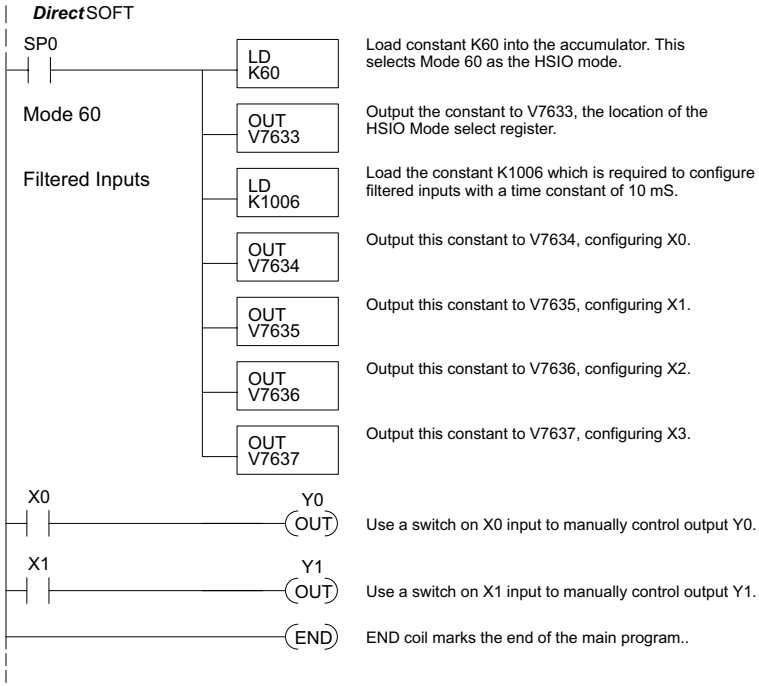
If you're having trouble with Mode 30 operation, please study the following symptoms and possible causes. The most common problems are listed below.

Symptom: The stepper motor does not rotate.

Possible causes:

1. **Configuration –** Verify that the HSIO actually generates pulses on outputs Y0 and Y1. Watch the status LEDs for Y0 and Y1 when you start a motion profile. If the LEDs flicker on and off or are steadily on, the configuration is probably correct.
2. **Programming error –** If there are no pulses on Y0 or Y1 you may have a programming error. Check the contents of V3636 for an error code that may be generated when the PLC attempts to do the move profile. Error code descriptions are given above.
3. **Check target value –** The profile will not pulse if the count value is equal to the target value (ex. count =0, target=0).

4. **Wiring** – Verify the wiring to the stepper motor is correct. Remember the signal ground connection from the PLC to the motion system is required.
5. **Motion system** – Verify that the drive is powered and enabled. To verify the motion system is working, you can use Mode 60 operation (normal PLC inputs/outputs) as shown in the test program below. With it, you can manually control Y0 and Y1 with X0 and X1, respectively. Using an input simulator is ideal for this type of manual debugging. With the switches you can single-step the motor in either direction. If the motor will not move with this simple control, Mode 30 operation will not be possible until the problem with the motor drive system or wiring is corrected.



6. **Memory Error** – HSIO configuration parameters are stored in the CPU system memory. Corrupted data in this memory area can sometimes interfere with proper HSIO operation. If all other corrective actions fail, initializing the scratchpad memory may solve the problem. With DirectSOFT, select PLC > Setup > Initialize Scratch Pad from the Menu bar.

Symptom: The motor turns in the wrong direction.

Possible causes:

1. **Wiring** – If you have selected CW and CCW type operation, just swap the wires on Y0 and Y1 outputs.
2. **Direction control** – If you have selected Pulse and Direction type operation, just change the direction bit to the opposite state.

Mode 40: High-Speed Interrupts

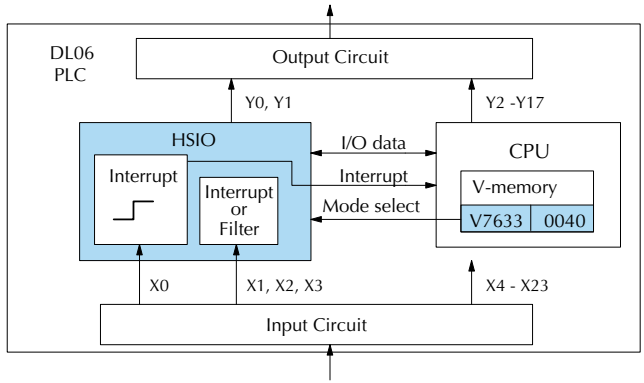
Purpose

The HSIO Mode 40 provides a high-speed interrupt to the ladder program. This capability is provided for your choice of the following application scenarios:

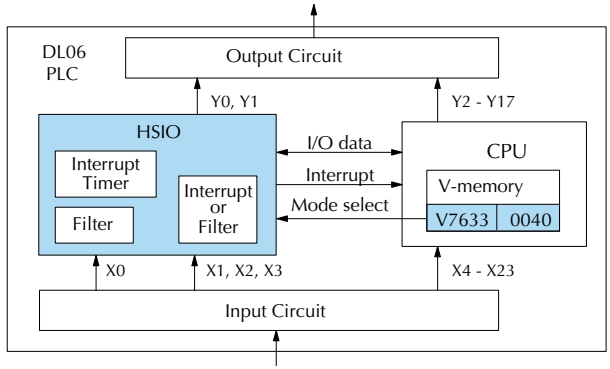
- External events need to trigger an interrupt subroutine in the CPU. Using immediate I/O instructions in the subroutine is typical.
- An interrupt routine needs to occur on a timed basis which is different from the CPU scan time (either faster or slower). The timed interrupt is programmable, from 5 to 999 mS.

Functional Block Diagram

The HSIO circuit creates the high-speed interrupt to the CPU. The following diagram shows the external interrupt option, which uses X0. In this configuration X1, X2 and X3 are external interrupts or normal filtered inputs.

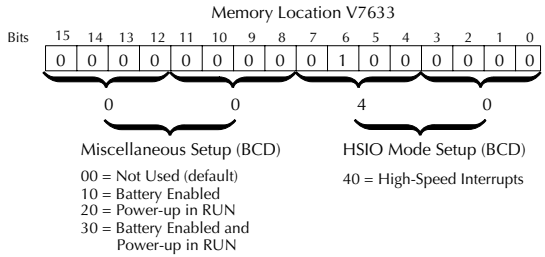


Alternately, you may configure the HSIO circuit to generate interrupts based on a timer, as shown below. In this configuration, inputs X0 is a filtered input.



Setup for Mode 40

Recall that V7633 is the HSIO Mode Select register. Refer to the diagram below. Use BCD 40 in the lower byte of V7633 to select high-speed interrupts.



Choose the most convenient method of programming V7633 from the following:

- Include load and out instructions in your ladder program
- ***Direct***SOFT's memory editor
- Use the Handheld Programmer D2–HPP

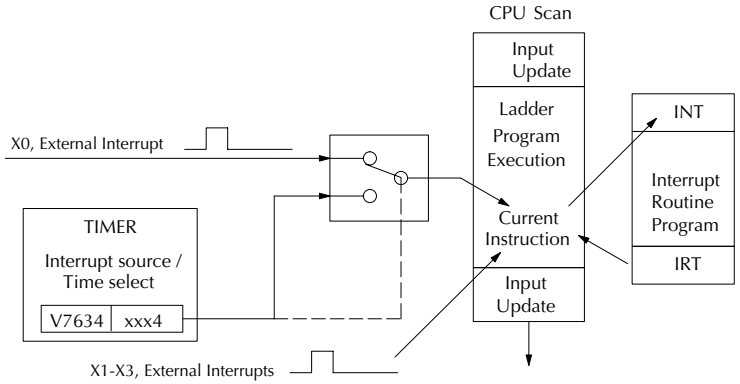
We recommend using the first method above so that the HSIO setup becomes an integral part of your application program. An example program later in this section shows how to do this.

Interrupts and the Ladder Program

Refer to the drawing below. The source of the interrupt may be external (X0 - X3). An internal timer can be used instead of X0 as the interrupt source. The setup parameter in V7634 serves a dual purpose:

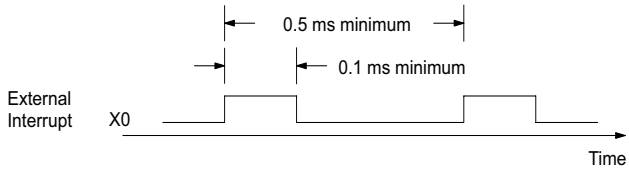
- It selects between the two interrupt sources (external or internal timer). The timed interrupt can only be used with X0.
- In the case of the timer interrupt, it programs the interrupt timebase between 5 and 999 mS.

The resulting interrupt uses label INT 0, 1, 2 or 3 in the ladder program. Be sure to include the Enable Interrupt (ENI) instruction at the beginning of your program. Otherwise, the interrupt routine will not be executed.



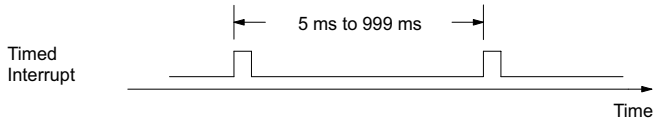
External Interrupt Timing Parameters

External interrupt signals must meet certain timing criteria to guarantee an interrupt will result. Refer to the timing diagram below. The minimum pulse width is 0.1 ms. There must be some delay before the next interrupt pulse arrives, such that the interrupt period cannot be smaller than 0.5 ms.



Timed Interrupt Parameters

When the timed interrupt is selected, the HSIO generates the interrupt to ladder logic. There is no interrupt pulse width in this case, but the interrupt period can be adjusted from 5 to 999 ms.



X Input / Timed INT Configuration

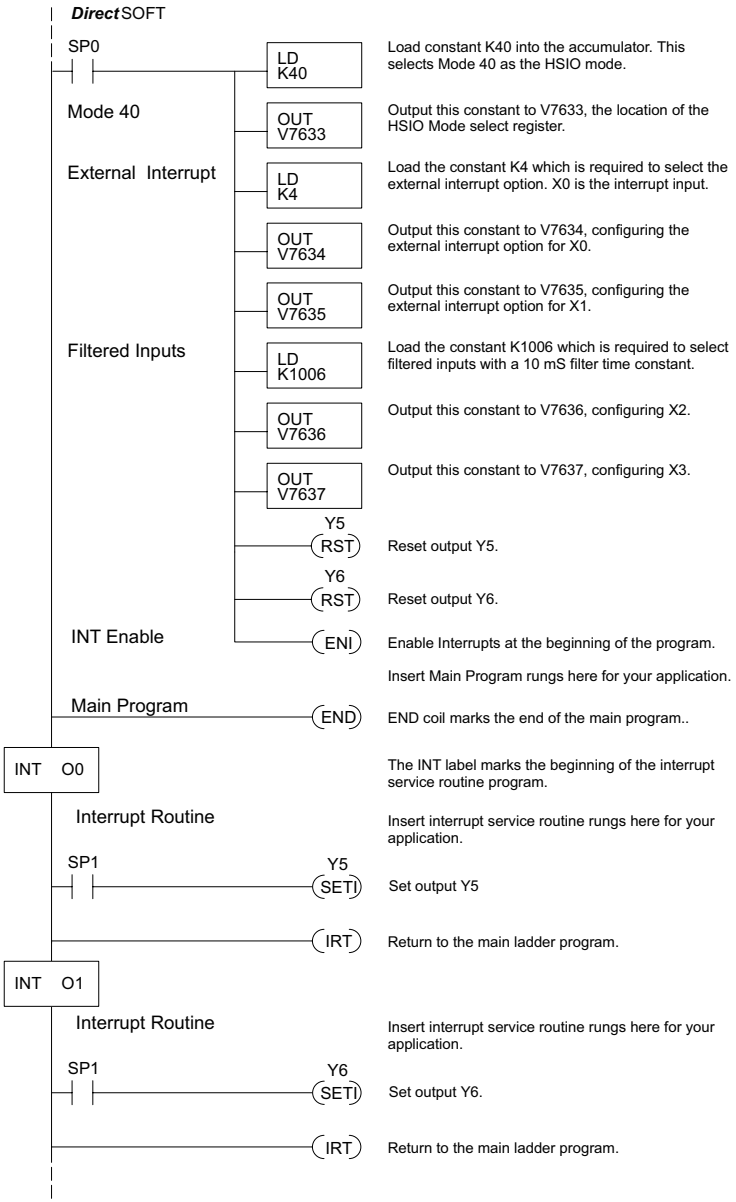
The configurable discrete input options for High-Speed Interrupt Mode are listed in the table below. Input X0 is the external interrupt when "0004" is in V7634. If you need a timed interrupt instead, then V7634 contains the interrupt time period, and input X0 becomes a filtered input (uses X1's filter time constant by default). Inputs X0, X1, X2, and X3, can be filtered inputs, having individual configuration registers and filter time constants, interrupt inputs or counter inputs.

Input	Configuration Register	Function	Hex Code Required
X0	V7634	External Interrupt	0004 (default)
		Timed Interrupt	xxx4, xxx = INT timebase 5 - 999 ms (BCD)
X1	V7635	Interrupt	0004 (default)
		Pulse Input	0005
		Filtered Input	xx06 (xx = filter time) 0 - 99 ms (BCD)
X2	V7636	Interrupt	0004 (default)
		Pulse Input	0005
		Filtered Input	xx06 (xx = filter time) 0 - 99 ms (BCD)
X3	V7637	Interrupt	0004 (default)
		Pulse Input	0005
		Filtered Input	xx06 (xx = filter time) 0 - 99 ms (BCD)

If you are only using *one* of the points for an interrupt, you may want to select a different *main* mode (i.e. 10, 20, 30, 50, or 60); and then, just configure one of the terminals not taken as an interrupt. For example, you might want to configure your CPU for the UP counter mode (Mode 10) and use point 03 for a high speed interrupt. You should read the individual sections for any alternate mode you might choose. There you will find instructions on how to select a high speed interrupt as a secondary feature.

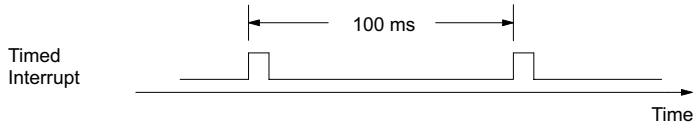
Program Example 1: External Interrupt

The following program selects Mode 40, then selects the external interrupt option for inputs X0 and X1. Inputs X2 and X3 are configured as filtered inputs with a 10 ms time constant. The program is otherwise generic, and may be adapted to your application.

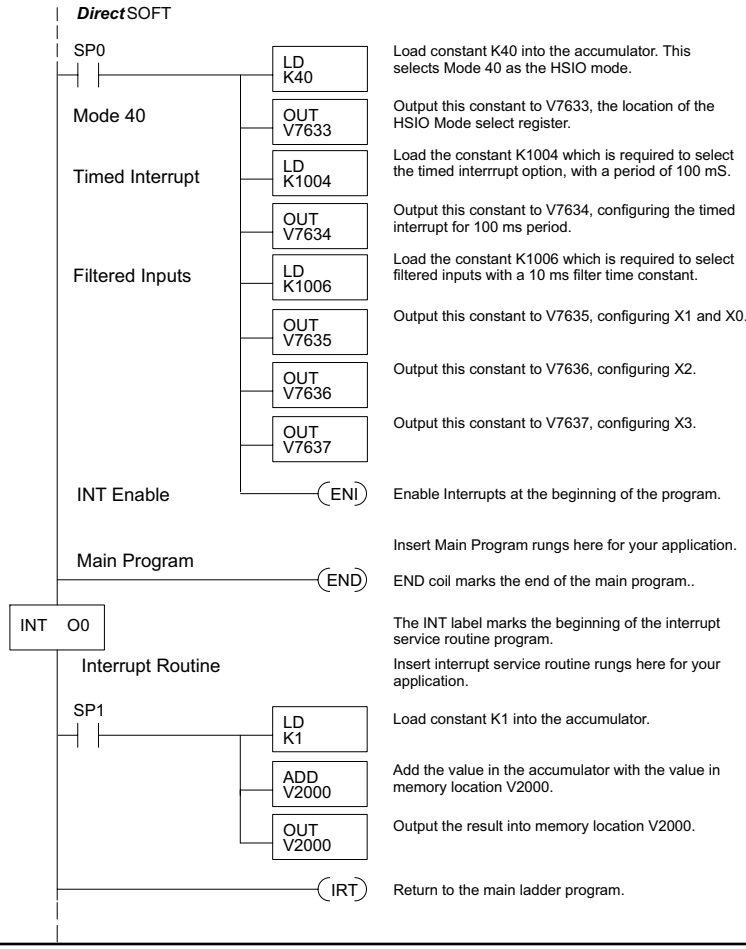


Program Example 2: Timed Interrupt

The following program selects Mode 40, then selects the timed interrupt option, with an interrupt period of 100 ms.



Inputs X0, X1, X2, and X3, are configured as filtered inputs with a 10 ms time constant. Note that X0 uses the time constant from X1. The program is otherwise generic, and may be adapted to your application.



NOTE: *X0 Cannot be used in the main program logic; however, using X0 to set C10, for instance, will allow the use of C10 in the main program logic. Do not forget to reset C10.*

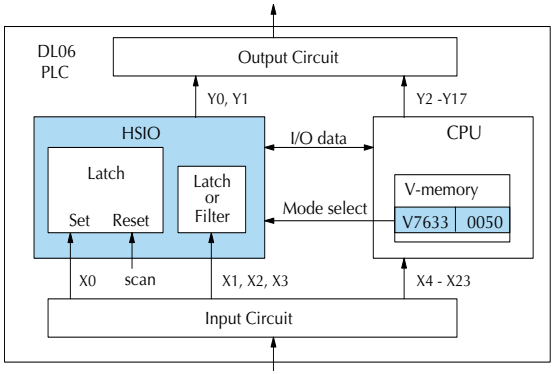
Mode 50: Pulse Catch Input

Purpose

The HSIO circuit has a pulse-catch mode of operation. It monitors the signal on inputs X0 - X3, preserving the occurrence of a narrow pulse. The purpose of the pulse catch mode is to enable the ladder program to *see* an input pulse which is shorter in duration than the current scan time. The HSIO circuit latches the input event on input X0 - X3 for one scan. This contact automatically goes off after one scan.

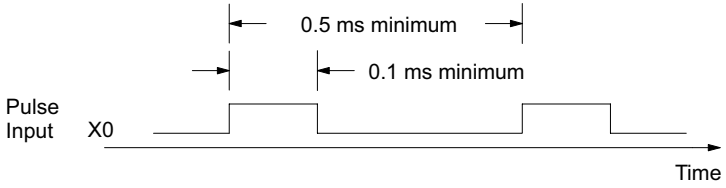
Functional Block Diagram

Refer to the block diagram below. When the lower byte of HSIO Mode register V7633 contains a BCD "50", the pulse catch mode in the HSIO circuit is enabled. X0 - X3 automatically become the pulse catch inputs, which set the latch on each rising edge. The HSIO resets the latch at the end of the next CPU scan. Inputs X1 X2, and X3 can be filtered discrete inputs, also.



Pulse Catch Timing Parameters

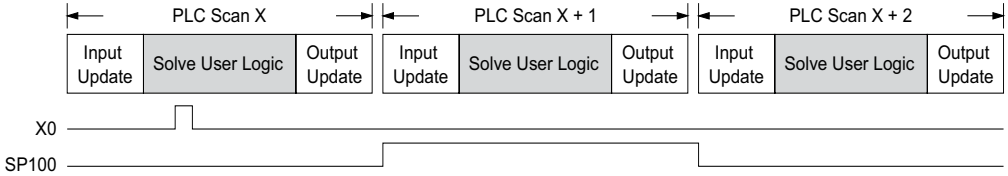
Signal pulses at X0 - X3 must meet certain timing criteria to guarantee a pulse capture will result. Refer to the timing diagram below. The input characteristics of X0 are fixed (it is not a programmable filtered input). The minimum pulse width is 0.1 ms. There must be some delay before the next pulse arrives, such that the pulse period cannot be smaller than 0.5 ms. If the pulse period is smaller than 0.5 ms, the next pulse will be considered part of the current pulse



NOTE: *The pulse catch and filtered input functions are opposite in nature. The pulse catch feature seeks to capture narrow pulses, while the filter input feature seeks to reject narrow pulses.*

When to use Pulse Catch Mode

Use the pulse catch mode for applications where the input (e.g. X0) can not be used in the user program because the pulse width is very narrow. Use SP100 instead of X0. The SP100 contact stays on through the next scan, as shown above. If X0 is on for more than scan, SP100 will remain on until X0 transitions off.

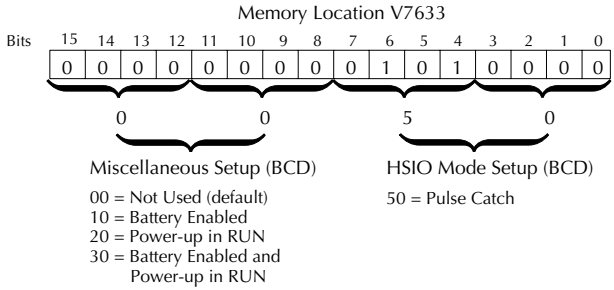


The status relay for X0 is SP100. The other status relays are shown in the table below.

Input	Status Relay
X0	SP100
X1	SP101
X2	SP102
X3	SP103

Setup for Mode 50

Recall that V7633 is the HSIO Mode Select register. Refer to the diagram below. Use BCD 50 in the lower byte of V7633 to select the High-Speed Counter Mode.



Choose the most convenient method of programming V7633 from the following:

- Include load and out instructions in your ladder program
- ***Direct***SOFT's memory editor
- Use the Handheld Programmer D2–HPP

We recommend using the first method above so that the HSIO setup becomes an integral part of your application program. An example program later in this section shows how to do this.

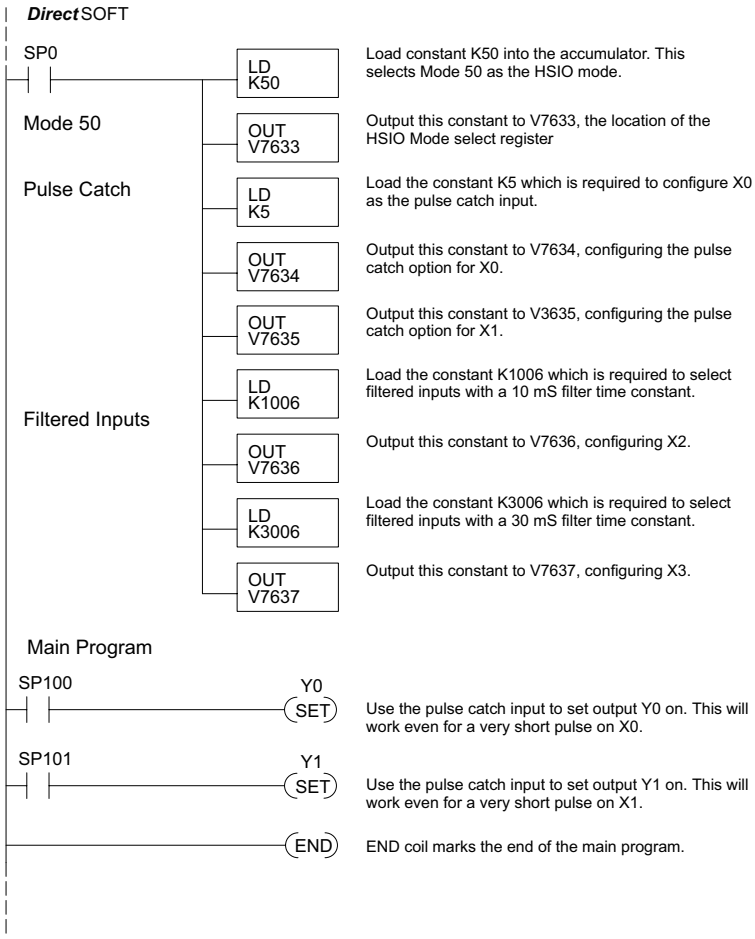
X Input Configuration

The configurable discrete input options for Pulse Catch Mode are listed in the table below. Each input has its own configuration register and filter time constant.

Input	Configuration Register	Function	Hex Code Required
X0	V7634	Pulse Catch Input	0005 (default)
X1	V7635	Interrupt	0004
		Pulse Catch Input	0005 (default)
		Filtered Input	xx06 (xx = filter time) 0 - 99 ms (BCD)
X2	V7636	Interrupt	0004
		Pulse Catch Input	0005 (default)
		Filtered Input	xx06 (xx = filter time) 0 - 99 ms (BCD)
X3	V7637	Interrupt	0004
		Pulse Catch Input	0005 (default)
		Filtered Input	xx06 (xx = filter time) 0 - 99 ms (BCD)

Program Example 1: Pulse Catch

The following program selects Mode 50, then programs the pulse catch code for X0 and X1. Inputs X2, and X3 are configured as filtered inputs with 10 and 30 mS time constants respectively. The program is otherwise generic, and may be adapted to your application.



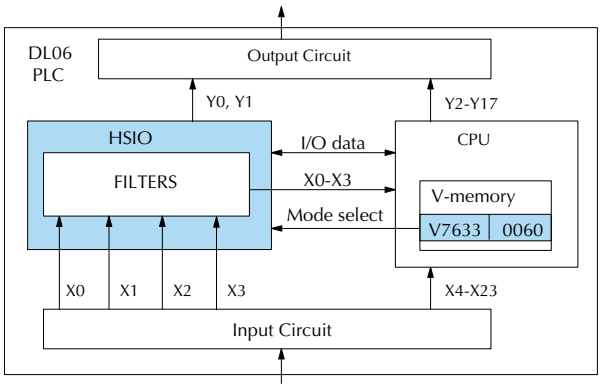
Mode 60: Discrete Inputs with Filter

Purpose

The last mode we will discuss for the HSIO circuit is Mode 60, Discrete Inputs with Filter. The purpose of this mode is to allow the input circuit to reject narrow pulses and accept wide ones, as viewed from the ladder program. This is useful in especially noisy environments or other applications where pulse width is important. In all other modes in this appendix, X0 to X3 usually support the mode functions as special inputs. Only spare inputs operate as filtered inputs by default. Now in Mode 60, all four inputs X0 through X3 function only as discrete filtered inputs.

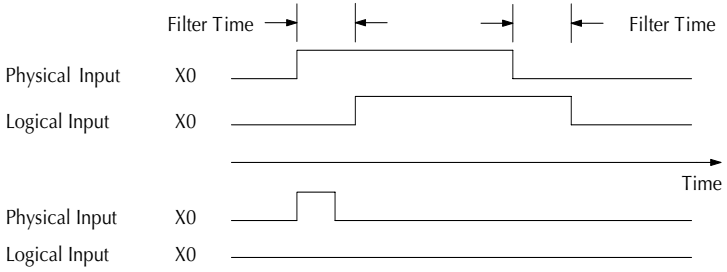
Functional Block Diagram

Refer to the block diagram below. When the lower byte of HSIO Mode register V7633 contains a BCD "60", the input filter in the HSIO circuit is enabled. Each input X0 through X3 has its own filter time constant. The filter circuit assigns the outputs of the filters as logical references X0 through X3.



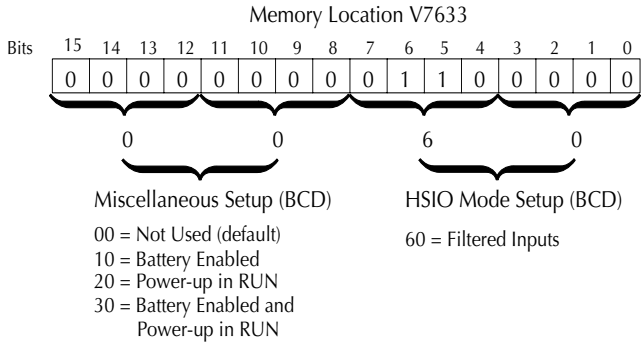
Input Filter Timing Parameters

Signal pulses at inputs X0 – X3 are filtered by using a delay time. In the figure below, the input pulse on the top line is longer than the filter time. The resultant logical input to ladder is phase-shifted (delayed) by the filter time on both rising and falling edges. In the bottom waveforms, the physical input pulse width is smaller than the filter time. In this case, the logical input to the ladder program remains in the OFF state (input pulse was filtered out).



Setup for Mode 60

Recall that V7633 is the HSIO Mode Select register. Refer to the diagram below. Use BCD 60 in the lower byte of V7633 to select the High-Speed Counter Mode.



Choose the most convenient method of programming V7633 from the following:

- Include load and out instructions in your ladder program
- *Direct*SOFT's memory editor
- Use the Handheld Programmer D2–HPP

We recommend using the first method above so that the HSIO setup becomes an integral part of your application program. An example program later in this section shows how to do this.

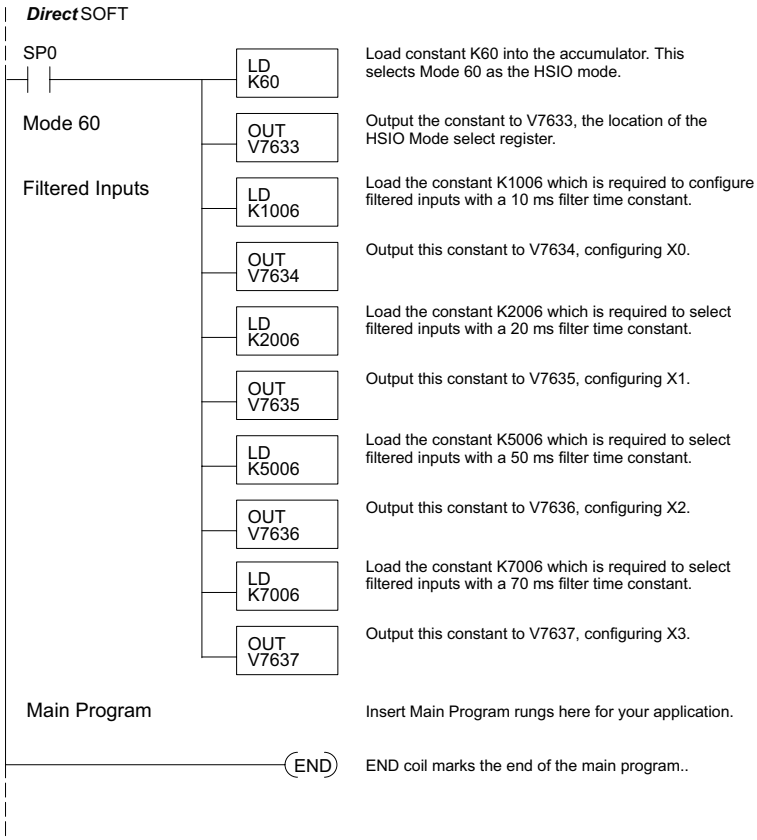
X Input Configuration

The configurable discrete input options for Discrete Filtered Inputs Mode are listed in the table below. The filter time constant (delay) is programmable from 0 to 99 mS (the input acts as a normal discrete input when the time constant is set to 0). The code for this selection occupies the upper byte of the configuration register in BCD. We combine this number with the required "06" in the lower byte to get "xx06", where xx = 0 to 99. Input X0, X1, X2, and X3 can only be filtered inputs. Each input has its own configuration register and filter time constant.

Input	Configuration Register	Function	Hex Code Required
X0	V7634	Filtered Input	xx06 (xx = filter delay time) 0 - 99 ms (BCD) (default)
X1	V7635	Filtered Input	xx06 (xx = filter delay time) 0 - 99 ms (BCD) (default)
X2	V7636	Filtered Input	xx06 (xx = filter delay time) 0 - 99 ms (BCD) (default)
X3	V7637	Filtered Input	xx06 (xx = filter delay time) 0 - 99 ms (BCD) (default)

Program Example: Filtered Inputs

The following program selects Mode 60, then programs the filter delay time constants for inputs X0, X1, X2, and X3. Each filter time constant is different, for illustration purposes. The program is otherwise generic, and may be adapted to your application.



PLC Memory

In This Appendix...

DL06 PLC Memory

When designing a PLC application, it is important for the PLC user to understand the different types of memory in the PLC. The DL06 CPU uses two types of memory: RAM and EEPROM. RAM is Random Access Memory and EEPROM is Electrically Erasable Programmable Read Only Memory. The PLC program is stored in EEPROM, and the PLC V-memory data is stored in RAM. There is also a small range of V-memory that can be copied to EEPROM which will be explained later.

The V-memory in RAM can be configured as either retentive or non-retentive.

Retentive memory is memory that is configured by the user to maintain values through a power cycle or a PROGRAM to RUN transition. Non-retentive memory is memory that is configured by the PLC user to clear data after a power cycle or a PROGRAM to RUN transition. The retentive ranges can be configured with either the handheld programmer using AUX57 or *Direct*SOFT (PLC Setup).

The contents of RAM memory can be written to and read from an infinite number of times, but RAM requires a power source to maintain the contents of memory. The contents of RAM are maintained by the internal power supply (5VDC) only while the PLC is powered by an external source, normally 120VAC. When power to the PLC is turned off, the contents of RAM are maintained by a Super-Capacitor. If the Super-Capacitor ever discharges, the contents of RAM will be lost. The data retention time of the Super-Capacitor backed RAM is 3 weeks maximum, and 4 1/2 days minimum (at 60° C). An optional battery, D2-BAT-1, can be added to maintain RAM retentive memory if the DL06 is ever without external power (see Volume I, page 3-8 for a detailed explanation).

The contents of EEPROM memory can be read from an infinite number of times, but there is a limit to the number of times it can be written to (typical specification is 100,000 writes). EEPROM does not require a power source to maintain the memory contents. It will retain the contents of memory indefinitely.

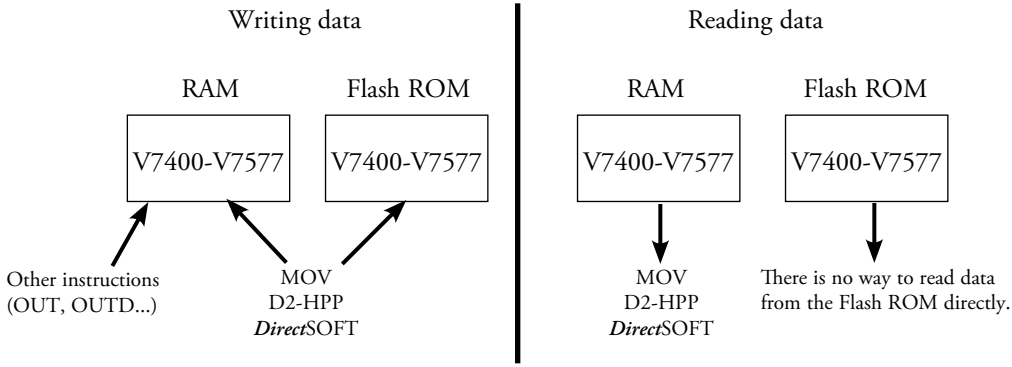
PLC user V-memory is stored in both volatile RAM and non-volatile EEPROM memory. Data being stored in RAM uses V400-V677, V1200-V7377 and V10000-V17777 which is volatile. Data stored in EEPROM uses V7400-V7577 and V700-V777, V7600-V7777 and V36000-V37777 are non-volatile.

Data values that must be retained for long periods of time, when the PLC is powered off, should be stored in EEPROM based V-memory. Since EEPROM is limited to the number of times it can be written to, it is suggested that transitional logic, such as a one-shot, be used to write the data one time instead on each CPU scan. The MOV instruction must be used to move the data into EEPROM.

Data values that are continually changing or which can be initialized with program logic should be stored in RAM based V-memory.

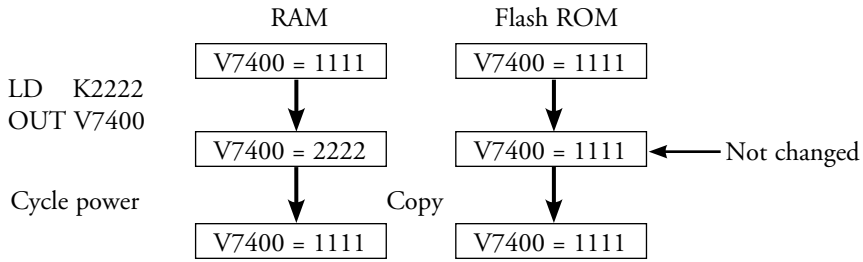
Non-volatile V-memory in the DL06

There are 2 types of memory assigned for the non-volatile V-memory area. They are RAM and flash ROM (EEPROM). They are sharing the same V-memory addresses; however, **you can only use the MOV instruction, D2-HPP and *Direct*SOFT or a device that uses K-Sequence or DirectNet protocol to write data to the flash ROM.** When you write data to the flash ROM, the same data is also written to RAM. If you use other instructions, you can only write data to RAM. When you read data from the non-volatile V-memory area, the data is always read from RAM.



After a power cycle, the PLC always copies the data in the flash ROM to the RAM.

If you use the instructions except for the MOV instruction to write data into the non-volatile V-memory area, you only update the data in RAM. After a power cycle, the PLC copies the previous data from the flash memory to the RAM, so you may think the data you changed has disappeared. To avoid trouble such as this, we recommend that you use the MOV instruction.



This appears to be previous data returning.

ASCII TABLE

In This Appendix...

ASCII Conversion Table

DECIMAL TO HEX TO ASCII CONVERTER											
DEC	HEX	ASCII	DEC	HEX	ASCII	DEC	HEX	ASCII	DEC	HEX	ASCII
0	00	NUL	32	20	space	64	40	@	96	60	`
1	01	SOH	33	21	!	65	41	A	97	61	a
2	02	STX	34	22	"	66	42	B	98	62	b
3	03	ETX	35	23	#	67	43	C	99	63	c
4	04	EOT	36	24	$	68	44	D	100	64	d
5	05	ENQ	37	25	%	69	45	E	101	65	e
6	06	ACK	38	26	&	70	46	F	102	66	f
7	07	BEL	39	27	'	71	47	G	103	67	g
8	08	BS	40	28	(	72	48	H	104	68	h
9	09	TAB	41	29	)	73	49	I	105	69	i
10	0A	LF	42	2A	*	74	4A	J	106	6A	j
11	0B	VT	43	2B	+	75	4B	K	107	6B	k
12	0C	FF	44	2C	,	76	4C	L	108	6C	l
13	0D	CR	45	2D	-	77	4D	M	109	6D	m
14	0E	SO	46	2E	.	78	4E	N	110	6E	n
15	0F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	DEL

Product Weights

In This Appendix...

Product Weight Table

PLC	Weight
D0–06AR	1.78 lb / 807g
D0–06DR	1.76 lb / 798g
D0–06DR-D	1.72 lb / 780g
D0–06AA	1.78 lb / 807g
D0–06DA	1.76 lb / 798g
D0–06DD1	1.68 lb / 762g
D0–06DD1-D	1.64 lb / 743g
D0–06DD2-D	1.64 lb / 743 g
D0–06DD2	1.68 lb / 798g
D0–06LCD	0.12 lb / 54.4 g

NUMBERING SYSTEMS

In This Appendix...

Introduction

As almost anyone who uses a computer is somewhat aware, the actual operations of a computer are done with a binary number system. Traditionally, the two possible states for a binary system are represented by the digits for "zero" (0) and "one" (1) although "off" and "on" or sometimes "no" and yes" are closer to what is actually involved. Most of the time a typical PC user has no need to think about this aspect of computers, but every now and then one gets confronted with the underlying nature of the binary system.

A PLC user, specifically the PLC programmer, should be familiar with the binary system. This appendix will provide an explanation of the numbering systems most commonly used by a PLC.

Binary Numbering System

Computers, including PLCs, use the Base 2 numbering system, which is called Binary and often called Decimal. As with a computer, there are only two valid digits a PLC relies on, zero and one, or off and on respectively. You would think that it would be hard to have a numbering system built on Base 2 with only two possible values, but the secret is by encoding using several digits.

Each digit in the base 2 system, when referenced by a computer, is called a bit. When four bits are grouped together, they form what is known as a nibble. Eight bits or two nibbles would be a byte. Sixteen bits or two bytes would be a word (Table 1). Thirty-two bits or two words is a double word.

Word															
Byte								Byte							
Nibble				Nibble				Nibble				Nibble			
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 1

Binary is not natural for us to use since we grow up using the base 10 system. Base 10 uses the numbers 0-9, as we are all well aware. From now on, the different bases will be shown as a subscripted number following the number. Example; 10 decimal would be 10_{10}.

Table 2 shows how base 2 numbers relate to their decimal equivalents.

A nibble of 1001_2 would be equal to a decimal number 9 ($1*2^3 + 1*2^0$ or $8_{10} + 1_{10}$). A byte of 11010101_2 would be equal to 213 ($1*2^7 + 1*2^6 + 1*2^4 + 1*2^2 + 1*2^0$ or $128_{10} + 64_{10} + 16_{10} + 4_{10} + 1_{10}$).

Binary/Decimal Bit Pattern																
Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Power	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Decimal Bit Value	32678	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
Max Value	65535_{10}															

Table 2

Hexadecimal Numbering System

The binary numbering system can be difficult and cumbersome to interpret for some users. Therefore, the hexadecimal numbering system was developed as a convenience for humans since the PLC (computer) only understands pure binary. The hexadecimal system is useful because it can represent every byte (8 bits) as two consecutive hexadecimal digits. It is easier for us to read hexadecimal numbers than binary numbers.

The hexadecimal numbering system uses 16 characters (base 16) to represent values. The first ten characters are the same as our decimal system (0-9), and the first six letters of the alphabet (A-F). Table 3 lists the first eighteen decimal numbers; 0-17 in the left column and the equivalent hexadecimal numbers are shown in the right column.

Decimal	Hex	Decimal	Hex
0	0	9	9
1	1	10	A
2	2	11	B
3	3	12	C
4	4	13	D
5	5	14	E
6	6	15	F
7	7	16	10
8	8	17	11

Table 3

Note that "10" and "11" in hex are not the same as "10" and "11" in decimal. Only the first ten numbers 0-9 are the same in the two representations. For example, consider the hex number "D8AF". To evaluate this hex number use the same method used to write decimal numbers. Each digit in a decimal number represents a multiple of a power of ten (base 10). Powers of ten increase from right to left. For example, the decimal number 365 means $3x10^2 + 6x10 + 5$. In hex each digit represents a multiple of a power of sixteen (base 16). Therefore, the hex number D8AF translated to decimal means $13x16^3 + 8x16^2 + 10x16 + 15 = 55471$. However, going through the arithmetic for hex numbers in order to evaluate them is not really necessary. The easier way is to use the calculator that comes as an accessory in Windows. It can convert between decimal and hex when in "Scientific" view.

Note that a hex number such as "365" is not the same as the decimal number "365". Its actual value in decimal terms is $3x16^2 + 6x16 + 5 = 869$. To avoid confusion, hex numbers are often labeled or tagged so that their meaning is clear. One method of tagging hex numbers is to append a lower case "h" at the end. Another method of labeling is to precede the number with 0x. Thus, the hex number "D8AF" can also be written "D8AFh", where the lower case "h" at the end is just a label to make sure we know that it is a hex number. Also, D8AF can be written with a labeling prefix as "0xD8AF".

Octal Numbering System

Many of the early computers used the octal numbering system for compiled printouts. Today, the PLC is about the only device that uses the Octal numbering system. The octal numbering system uses 8 values to represent numbers. The values are 0-7 being Base 8. Table 4 shows the first 31 decimal digits in octal. Note that the octal values are 0-7, 10-17, 20-27, and 30-37.

This follows the *Direct*LOGIC PLCs. Refer to Chapter 3 bit maps and notice that the memory addresses are numbered in octal, as well as each bit. The octal system is much like counting in the decimal system without the digits 8 and 9 being available.

Octal	Decimal	Octal	Decimal
0	0	20	16
1	1	21	17
2	2	22	18
3	3	23	19
4	4	24	20
5	5	25	21
6	6	26	22
7	7	27	23
10	8	30	24
11	9	31	25
12	10	32	26
13	11	33	27
14	12	34	28
15	13	35	29
16	14	36	30
17	15	37	31

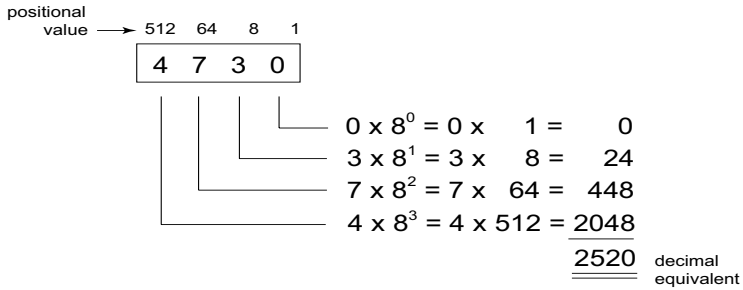
Table 4

The general format for four digits of the octal number is:

$$(d \times 8^0) + (d \times 8^1) + (d \times 8^2) + (d \times 8^3)$$

where "d" means digit. This is the same format used in the binary, decimal, or hexadecimal systems except that the base number for octal is 8.

Using the powers of expansion, the example below shows octal 4730 converted to decimal.



Binary Coded Decimal (BCD) Numbering System

BCD is a numbering system where four bits are used to represent each decimal digit. The binary codes corresponding to the hexadecimal digits A-F are not used in the BCD system. For this reason numbers cannot be coded as efficiently using the BCD system. For example, a byte can represent a maximum of 256 different numbers (i.e. 0-255) using normal binary, whereas only 100 distinct numbers (i.e. 0-99) could be coded using BCD. Also, note that BCD is a subset of hexadecimal and neither one does negative numbers.

	BCD Bit Pattern															
Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Power	10^3				10^2				10^1				10^0			
Bit Value	8	4	2	1	8	4	2	1	8	4	2	1	8	4	2	1
Max Value	9				9				9				9			

Table 5

One plus for BCD is that it reads like a decimal number, for example, 867 in BCD would be expressed the same as 867 decimal. No conversion is needed; however, within the PLC, BCD calculations can be performed if numbers are adjusted to BCD after normal binary arithmetic.

Real (Floating Point) Numbering System

The terms Real and floating-point both describe IEEE-754 floating point arithmetic. This standard specifies how single precision (32 bit) and double precision (64 bit) floating point numbers are to be represented as well as how arithmetic should be carried out on them.

	Real (Floating Point 32) Bit Pattern															
Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Sign	Exponent								Mantissa						
Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Mantissa (continues from above)															

Table 6

Most PLCs use the 32-bit format for floating point (or Real) numbers.

Floating point numbers which ***Direct***LOGIC PLCs use have three basic components: sign, exponent and mantissa. The 32 bit word required for the IEEE standard floating point numbers is shown in Table 6. It is represented as a number from 0 to 31, left to right. The first bit (31) is the sign bit, the next eight bits (30-23) are the exponent bits and the final 23 bits (22-0) are the fraction bits. In summary:

The sign bit is either "0" for positive or "1" for negative;

The exponent uses base 2;

The first bit of the mantissa is typically assumed to be "1.*fff*", where "*f*" is the field of fraction bits.

The Internet can provide a more in-depth explanation of the floating point numbering system. One website to look at is:

http://www.psc.edu/general/software/packages/ieee/ieee.html

BCD/Binary/Decimal/Hex/Octal - What is the Difference?

Sometimes there is confusion about the differences between the data types used in a PLC. The PLC's native data format is BCD, while the I/O numbering system is octal. Other numbering formats used are binary and Real. Although data is stored in the same manner (0's and 1's), there are differences in the way that the PLC interprets it.

While all of the formats rely on the base 2 numbering system and bit-coded data, the format of the data is dissimilar. Table 7 below shows the bit patterns and values for various formats.

Binary/Decimal Bit Pattern																
Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Decimal Bit Value	32678	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
Max Value	65535															

	Hexadecimal Bit Pattern															
Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Decimal Bit Value	8	4	2	1	8	4	2	1	8	4	2	1	8	4	2	1
Max Value	F				F				F				F			

	BCD Bit Pattern															
Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Decimal Bit Value	8	4	2	1	8	4	2	1	8	4	2	1	8	4	2	1
Max Value	9				9				9				9			

	Octal Bit Pattern															
Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Bit Value	1	4	2	1	4	2	1	4	2	1	4	2	1	4	2	1
Max Value	1	7			7			7			7			7		

	Real (Floating Point 32) Bit Pattern															
Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Sign	Exponent								Mantissa						
Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Mantissa (continued from above)															

Table 7

As seen in Table 7, the BCD and hexadecimal formats are similar, although the maximum number for each grouping is different (9 for BCD and F for hexadecimal). This allows both formats to use the same display method. The unfortunate side effect is that unless the data type is documented, it's difficult to know what the data type is unless it contains the letters A-F.

Data Type Mismatch

Data type mismatching is a common problem when using an operator interface. Diagnosing it can be a challenge until you identify the symptoms. Since the PLC uses BCD as the native format, many people tend to think it is interchangeable with binary (unsigned integer) format. This is true to some extent, but not in this case. Table 8 shows how BCD and binary numbers differ.

Data Type Mismatch												
Decimal	0	1	2	3	4	5	6	7	8	9	10	11
BCD	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	0001 0000	0001 0001
Binary	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	0000 1010	0000 1011

Table 8

As the table shows, BCD and binary share the same bit pattern up until you get to the decimal number 10. Once you get past 10, the bit pattern changes. The BCD bit pattern for the decimal 10 is actually equal to a value of 16 in binary, causing the number to jump six digits by when viewing it as the BCD. With larger numbers, the error multiplies. Binary values from 10 to 15 Decimal are actually invalid for the BCD data type.

Looking at a larger number, such as the value shown in Table 9, both the BCD bit pattern and the decimal bit pattern correspond to a base 10 value of 4095_{10}. If bit patterns are read, or interpreted, in a different format than what is used to write them, the data will not be correct. For instance, if the BCD bit pattern is interpreted as a decimal (binary) bit pattern, the result is a base 10 value of 16533_{10}. Similarly, if you try to view the decimal (binary) bit pattern as a BCD value, it is not a valid BCD value at all, but could be represented in hexadecimal as 0xFFF.

Base 10 Value	BCD Bit Pattern	Binary Bit Pattern
4095	0100 0000 1001 0101	1111 1111 1111

Table 9

Look at the following example and note the same value represented by the different numbering systems.

0100	0011	Binary		0001	0010	0011	0100	Binary
6	7	Decimal		4	6	6	0	Decimal
4	3	Hex		1	2	3	4	Hex
0110	0111	BCD		0100	0110	0110	0000	BCD
1	0 3	Octal		1	1	0	6	4	Octal

Signed vs. Unsigned Integers

So far, we have dealt with unsigned data types only. Now we will deal with signed data types (negative numbers). The BCD and hexadecimal numbering systems do not use signed data types.

In order to signify that a number is negative or positive, we must assign a bit to it. Usually, this is the Most Significant Bit (MSB) as shown in Table 10. For a 16-bit number, this is bit 15. This means that for 16-bit numbers we have a range of -32767 to 32767.

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	MSB															LSB

Table 10

There are two ways to encode a negative number: two's complement and Magnitude Plus sign. The two methods are not compatible.

The simplest method to represent a negative number is to use one bit of the PLC word as the sign of a number while the remainder of the word gives its magnitude. It is general convention to use the most significant bit (MSD) as the sign bit: a 1 will indicate a negative, and a 0 a positive number. Thus, a 16 bit word allows numbers in the range ±32767. Table 12 shows a representations of 100 and a representation of -100 in this format.

Magnitude Plus Sign	
Decimal	**Binary**
100	0000 0000 0110 0100
-100	1000 0000 0110 0100

Table 11

Two's complement is a bit more complicated. Without getting involved with a full explanation, a simple formula for two's complement is to invert the binary and add one (see Table 12). Basically, 1's are being changed to 0's and all 0's are being changed to 1.

Two's Compliment	
Decimal	**Binary**
100	0000 0000 0110 0100
-100	1111 1111 1001 1100

Table 12

More information about 2's complement can be found on the Internet at the following websites:
http://www.evergreen.edu/biophysics/technotes/program/2s_comp.htm

AutomationDirect.com Products and Data Types

DirectLOGIC PLCs

The *Direct*LOGIC PLC family uses the octal numbering system for all addressing which includes: inputs, outputs, internal V-memory locations, timers, counters, internal control relays (bits), etc. Most data in the PLC, including timer and counter current values, is in BCD format by default. User data in V-memory locations may be stored in other data types if it is changed by the programmer, or comes from some external source, such as an operator interface. Any manipulation of data must use instructions appropriate for that data type which includes: Load instructions, Math instructions, Out box instructions, comparison instructions, etc. In many cases, the data can be changed from one data type to another, but be aware of the limitations of the various data types when doing so. For example, to change a value from BCD to decimal (binary), use a BIN instruction box. To change from BCD to a real number, use a BIN and a BTOR instruction box. When using Math instructions, ***the data types must match.*** For example, a BCD or decimal (binary) number cannot be added to a real number, and a BCD number cannot be added to a decimal (binary) number. If the data types are mismatched, the results of any math operation will be meaningless.

To simplify making, number conversions Intelligent Box (IBox) Instructions are available with *Direct*SOFT. These instruction descriptions are located in Volume 1, page 5-230, in the Math IBox group.

Most *Direct*LOGIC analog modules can be setup to give the raw data in decimal (binary) format or in BCD format, so it is necessary to know how the module is being used. *Direct*LOGIC PID is another area where not all values are in BCD. In fact, nearly all of the PID parameters are stored in the PLC memory as decimal (binary) numbers.

NOTE: *The PID algorithm uses magnitude plus sign for negative decimal (binary) numbers, whereas the standard math functions use two's complement. This can cause confusion while debugging a PID loop.*

When using the Data View in *Direct*SOFT, be certain that the proper format is selected for the element to be viewed. The data type and length is selected using the drop-down boxes at the top of the Data View window. Also notice that BCD is called BCD/Hex. Remember that BCD is a subset of hexadecimal so they share a display format even though the values may be different. This is where good documentation of the data type stored in memory is crucial.

C-more/C-more Micro-Graphic Panels

In the C-more and C-more Micro-Graphic HMI operator panels, the 16-bit BCD format is listed as "BCD int 16". Binary format is either "Unsigned int 16" or "Signed int 16" depending on whether or not the value can be negative. Real number format is "Floating PT 32".

More available formats are, "BCD int 32", "Unsigned int 32" and "Signed int 32".

EUROPEAN UNION DIRECTIVES (CE)

In This Appendix...

European Union (EU) Directives

NOTE: *The information contained in this section is intended as a guideline and is based on our interpretation of the various standards and requirements. Since the actual standards are issued by other parties, and in some cases governmental agencies, the requirements can change over time without advance warning or notice. Changes or additions to the standards can possibly invalidate any part of the information provided in this section.*

This area of certification and approval is absolutely vital to anyone who wants to do business in Europe. One of the key tasks that faced the EU member countries and the European Economic Area (EEA) was the requirement to bring several similar yet distinct standards together into one common standard for all members. The primary purpose of a single standard was to make it easier to sell and transport goods between the various countries and to maintain a safe working and living environment. The Directives that resulted from this merging of standards are now legal requirements for doing business in Europe. Products that meet these Directives are required to have a CE mark to signify compliance.

Member Countries

As of January 1, 2007, the members of the EU are Austria, Belgium, Bulgaria, Cyprus, Czech Republic, Denmark, Estonia, Finland, France, Germany, Greece, Hungary, Ireland, Italy, Latvia, Lithonia, Luxembourg, Malta, Netherlands, Poland, Portugal, Romania, Slovakia, Slovenia, Spain, Sweden, and United Kingdom. Iceland, Liechtenstein, and Norway together with the EU members make up the European Economic Area (EEA) and all are covered by the Directives.

Applicable Directives

There are several Directives that apply to our products. Directives may be amended or added as required.

- **Electromagnetic Compatibility Directive (EMC)** — this Directive attempts to ensure that devices, equipment and systems have the ability to function satisfactorily in an electromagnetic environment without introducing intolerable electromagnetic disturbance to anything in that environment.
- **Machinery Safety Directive** — this Directive covers the safety aspects of the equipment, installation, etc. There are several areas involved, including testing standards covering both electrical noise immunity and noise generation.
- **Low Voltage Directive** — this Directive is also safety related and covers electrical equipment that has voltage ranges of 50–1000VAC and/or 75–1500VDC.
- **Battery Directive** — this Directive covers the production, recycling and disposal of batteries.

Compliance

Certain standards within each Directive already require mandatory compliance. The EMC Directive, which has gained the most attention, became mandatory as of January 1, 1996. The Low Voltage Directive became mandatory as of January 1, 1997.

Ultimately, we are all responsible for our various pieces of the puzzle. As manufacturers, we must test our products and document any test results and/or installation procedures that are necessary to comply with the Directives. As a machine builder, you are responsible for

installing the products in a manner which will ensure compliance. You are also responsible for testing any combination of products that may (or may not) comply with the Directives when used together. The end user of the products must comply with any Directives that may cover maintenance, disposal, etc., of equipment or various components. *Although we strive to provide the best assistance available, it is impossible for us to test all possible configurations of our products with respect to any specific Directive. Because of this, it is ultimately your responsibility to ensure that your machinery (as a whole) complies with these Directives and to keep up with applicable Directives and/or practices that are required for compliance.*

As of January 1, 1999, the DL05, DL06, DL205, DL305, and DL405 PLC systems manufactured by Koyo Electronics Industries or FACTS Engineering, when properly installed and used, conform to the Electromagnetic Compatibility (EMC), Low Voltage Directive, and Machinery Directive requirements of the following standards.

- **EMC Directive Standards Relevant to PLCs**
  EN50081–1 Generic emission standard for residential, commercial, and light industry
  EN50081–2 Generic emission standard for industrial environment.
  EN50082–1 Generic immunity standard for residential, commercial, and light industry
  EN50082–2 Generic immunity standard for industrial environment.
- **Low Voltage Directive Standards Applicable to PLCs**
  EN61010–1 Safety requirements for electrical equipment for measurement, control, and laboratory use.
- **Product Specific Standard for PLCs**
  EN61131–2 Programmable controllers, equipment requirements and tests. This standard replaces the above generic standards for immunity and safety. However, the generic emissions standards must still be used in conjunction with the following standards:
  -EN 61000-3-2 Harmonics
  -EN 61000-3-2 Fluctuations
- **Warning on Electrostatic Discharge (ESD)**
  We recommend that all personnel take necessary precautions to avoid the risk of transferring static charges to inside the control cabinet, and clear warnings and instructions should be provided on the cabinet exterior. Such precautions may include the use of earth straps, similar devices or the powering off of the equipment inside the enclosure before the door is opened.
- **Warning on Radio Interference (RFI)**
  This is a class A product. In a domestic environment, this product may cause radio interference in which case the user may be required to take adequate measures.

General Safety

- External switches, circuit breaker or external fusing, are required for these devices.
- The switch or circuit breaker should be mounted near the PLC equipment.

AutomationDirect is currently in the process of changing testing procedures from the generic standards to the product specific standards.

Other Sources of Information

Although the EMC Directive gets the most attention, other basic Directives, such as the Machinery Directive and the Low Voltage Directive, also place restrictions on the control panel builder. Because of these additional requirements it is recommended that the following publications be purchased and used as guidelines:

- BSI publication TH 42073: February 1996 – covers the safety and electrical aspects of the Machinery Directive
- EN 60204–1:1992 – General electrical requirements for machinery, including Low Voltage and EMC considerations
- IEC 1000–5–2: EMC earthing and cabling requirements
- IEC 1000–5–1: EMC general considerations

It may be possible for you to obtain this information locally; however, the official source of applicable Directives and related standards is:

The Office for Official Publications of the European Communities L–2985 Luxembourg; quickest contact is via the World Wide Web at http://euro–op.eu.int/indexn.htm

Another source is:

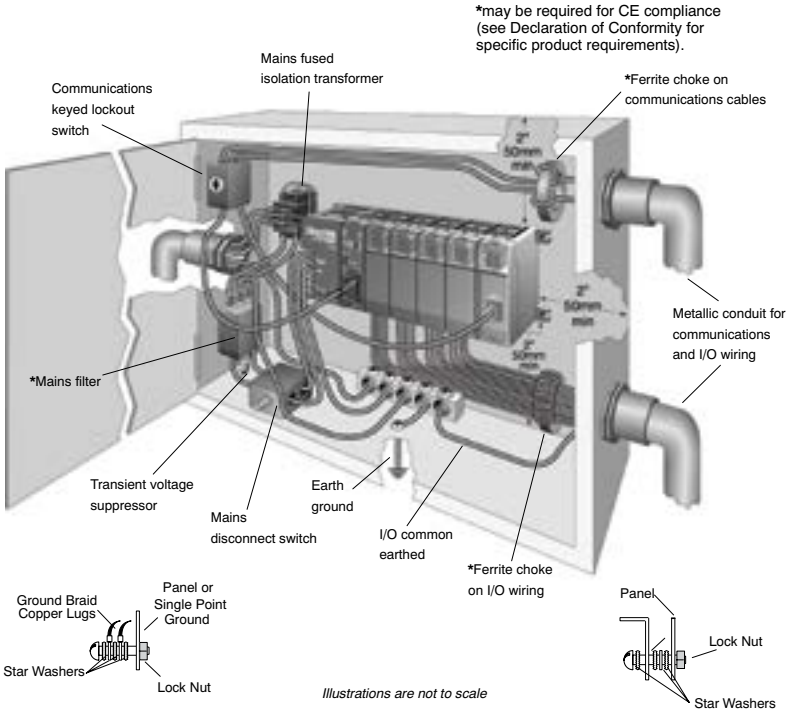
The 'Blue Guide' on the implementation of EU product rules 2016

http://ec.europa.eu/DocsRooms/documents/18027/

Basic EMC Installation Guidelines

Enclosures

The following diagram illustrates good engineering practices supporting the requirements of the Machinery and Low Voltage Directives. House all control equipment in an industry standard lockable steel enclosure and use metallic conduit for wire runs and cables.



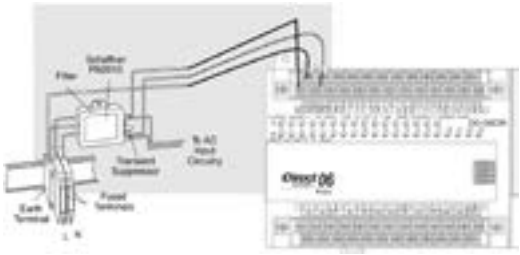
Electrostatic Discharge (ESD)

We specify in all declarations of conformity that our products are installed inside an industrial enclosure using metallic conduit for external wire runs; therefore, we test the products in a typical enclosure. However, we would like to point out that although our products operate normally in the presence of ESD, this is only the case when mounted within an enclosed industrial control cabinet. When the cabinet is open during installation or maintenance, the equipment and or programs may be at risk of damage from ESD carried by personnel.

We therefore recommend that all personnel take necessary precautions to avoid the risk of transferring static electricity to components inside the control cabinet. If necessary, clear warnings and instructions should be provided on the cabinet exterior, such as recommending the use of earth straps of similar devices, or the powering off of equipment inside the enclosure.

Mains Filters

DL05, DL06, and DL205 AC powered base power supplies require extra mains filtering to comply with the EMC Directive on conducted RF emissions. All PLC equipment has been tested with filters from Schaffner, which reduce emissions levels if the filters are properly grounded (earth ground). A filter with a current rating suitable to supply all PLC power supplies and AC input modules should be selected. We suggest the FN2010 for DL05/DL06/ DL205 systems.



NOTE: *Very few mains filters can reduce problem emissions to negligible levels. In some cases, filters may increase conducted emissions if not properly matched to the problem emissions.*

Suppression and Fusing

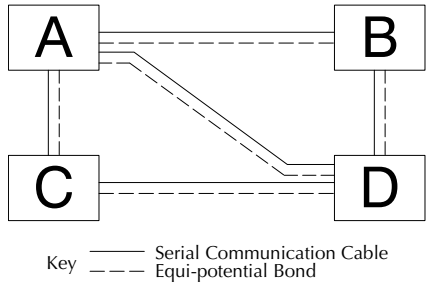
In order to comply with the fire risk requirements of the Low Voltage and Machinery Directive electrical standards EN 61010–1, and EN 60204–1, by limiting the power into "unlimited" mains circuits with power leads reversed, it is necessary to fuse both AC and DC supply inputs. You should also install a transient voltage suppressor across the power input connections of the PLC. Choose a suppressor such as a metal oxide varistor, with a rating of 275VAC working voltage for 230V nominal supplies (150VAC working voltage for 115V supplies) and high energy capacity (eg. 140 joules).

Transient suppressors must be protected by fuses and the capacity of the transient suppressor must be greater than the blow characteristics of the fuses or circuit breakers to avoid a fire risk. A recommended AC supply input arrangement for Koyo PLCs is to use twin 3 amp TT fused terminals with fuse blown indication, such as DINnectors DN–F10L terminals, or twin circuit breakers, wired to a Schaffner FN2010 filter or equivalent, with high energy transient suppressor soldered directly across the output terminals of the filter. PLC system inputs should also be protected from voltage impulses by deriving their power from the same fused, filtered, and surge-suppressed supply.

Internal Enclosure Grounding

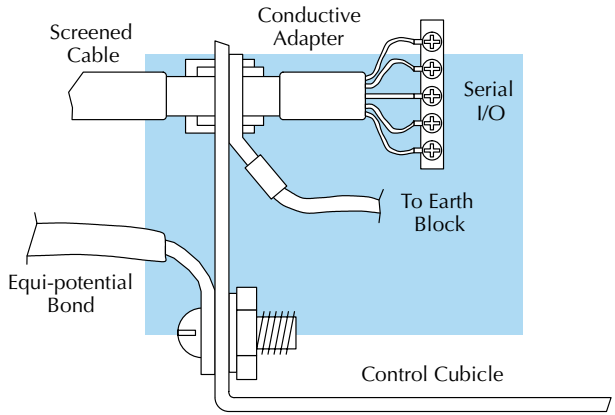
A heavy-duty star earth terminal block should be provided in every cubicle for the connection of all earth ground straps, protective earth ground connections, mains filter earth ground wires, and mechanical assembly earth ground connections. This should be installed to comply with safety and EMC requirements, local standards, and the requirements found in IEC 1000-5-2. The Machinery Directive also requires that the common terminals of PLC input modules and common supply side of loads driven from PLC output modules be connected to the protective earth ground terminal.

Equipotential Grounding



Adequate site earth grounding must be provided for equipment containing modern electronic circuitry. The use of isolated earth electrodes for electronic systems is forbidden in some countries. Make sure you check any requirements for your particular destination. IEC 1000–5–2 covers equipotential bonding of earth grids adequately, but special attention should be given to apparatus and control cubicles that contain I/O devices, remote I/O racks, or have inter-system communications with the primary PLC system enclosure. An equipotential bond wire must be provided alongside all serial communications cables, and to any separate items of the plant which contain I/O devices connected to the PLC. The diagram shows an example of four physical locations connected by a communications cable.

Communications and Shielded Cables



Good quality 24 AWG minimum twisted-pair shielded cables, with overall foil and braid shields are recommended for analog cabling and communications cabling outside of the PLC enclosure. To date, it has been a common practice to provide an earth ground for one end of the cable shield only in order to minimize the risk of noise caused by earth ground loop currents between apparatus. The procedure of grounding only one end, which primarily originated as a result of trying to reduce hum in audio systems, is no longer applicable to the complex industrial environment. Shielded cables are also efficient emitters of RF noise from the PLC system, and can interact in a parasitic manner in networks and between multiple sources of interference.

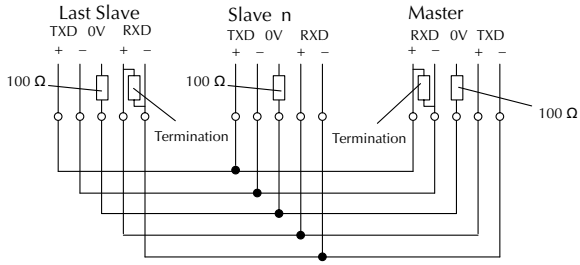
The recommendation is to use shielded cables as electrostatic "pipes" between apparatus and systems, and to run heavy gauge equipotential bond wires alongside all shielded cables. When a shielded cable runs through the metallic wall of an enclosure or machine, it is recommended in IEC 1000–5–2 that the shield be connected over its full perimeter to the wall, preferably using a conducting adapter, and not via a pigtail wire connection to an earth ground bolt. Shields must be connected to every enclosure wall or machine cover that they pass through.

Analog and RS232 Cables

Providing an earth ground for both ends of the shield for analog circuits provides the perfect electrical environment for the twisted pair cable as the loop consists of signal and return, in a perfectly balanced circuit arrangement, with connection to the common of the input circuitry made at the module terminals. RS232 cables are handled in the same way.

Multidrop Cables

RS422 twin twisted pair, and RS485 single twisted pair cables also require a 0V link, which has often been provided in the past by the cable shield. It is now recommended that you use triple twisted pair cabling for RS422 links, and twin twisted pair cable for RS485 links. This is because the extra pair can be used as the 0V inter-system link. With loop DC power supplies earth grounded in both systems, earth loops are created in this manner via the inter-system 0v link. The installation guides encourage earth loops, which are maintained at a low impedance by using heavy equipotential bond wires. **To account for non–European installations using single-end earth grounds, and sites with far from ideal earth ground characteristics, we recommend the addition of 100 ohm resistors at each 0V link connection in network and communications cables.**



Shielded Cables within Enclosures

When you run cables between PLC items within an enclosure which also contains susceptible electronic equipment from other manufacturers, remember that these cables may be a source of RF emissions. There are ways to minimize this risk. Standard data cables connecting PLCs and/or operator interfaces should be routed well away from other equipment and their associated cabling. You can make special serial cables where the cable shield is connected to the enclosure's earth ground at both ends, the same way as external cables are connected.

Analog Modules and RF Interference

All Automationdirect products are tested to withstand field strength levels up to 10V/m, which is the maximum required by the relevant EU standards. While all products pass this test, analog modules will typically exhibit deviations of their readings. This is quite normal; however, systems designers should be aware of this and plan accordingly.

When assembling a control system using analog modules, these standards must be followed and should be integrated into the system design. This is the responsibility of the system builder/ commissioner.

Network Isolation

For safety reasons, it is a specific requirement of the Machinery Directive that **a keyswitch must be provided** that isolates any network input signal during maintenance, so that remote commands cannot be received that could result in the operation of the machinery. To avoid the introduction of noise into the system, any keyswitch assembly should be housed in its own earth grounded steel box and the integrity of the shielded cable must be maintained.

Also, if you are connected to the World Wide Web, you can check the EU commission's official site at: http://ec.europa.eu/index_en.htm.

DC Powered Versions

Due to slightly higher emissions radiated by the DC powered versions of the DL06, and the differing emissions performance for different DC supply voltages, the following stipulations must be met:

- The PLC must be housed within a metallic enclosure with a minimum number of orifices.
- I/O and communications cabling exiting the cabinet must be contained within metallic conduit/trunking.

Items Specific to the DL06

- The rating between all circuits in this product is **as basic insulation only**, as appropriate for single fault conditions.
- There is no isolation offered between the PLC and the analog inputs of this product.
- It is the responsibility of the system designer to earth one side of all control and power circuits, and to earth the braid of screened cables.
- This equipment must be properly installed while adhering to the guidelines and the installation standards IEC 1000–5–1, IEC 1000–5–2 and IEC 1131–4.
- It is a requirement that all PLC equipment must be housed in a protective steel enclosure, which limits access to operators by a lock and power breaker. If access is required by operators or untrained personnel, the equipment must be installed inside an internal cover or secondary enclosure. A warning label must be used on the front door of the installation cabinet as follows: **Warning: Exposed terminals and hazardous voltages inside**
- It should be noted that the safety requirements of the machinery directive standard EN60204–1 state that all equipment power circuits must be wired through isolation transformers or isolating power supplies, and that one side of all AC or DC control circuits must be earthed.
- Both power input connections to the PLC must be separately fused using 3 amp T-type anti–surge fuses, and a transient suppressor fitted to limit supply overvoltages.
- **Warning: If the equipment is used in a manner not specified by the manufacturer, the protection provided by the equipment may be impaired.**

INTRODUCTION TO SERIAL COMMUNICATIONS

In This Appendix...

Introduction to Serial Communications

*Direct*LOGIC® PLCs have two built-in serial communication ports which can be used to communicate to other PLCs or to other serial devices. To fully understand the capabilities and limitations of the serial ports, a brief introduction to serial communications is in order.

There are three major components to any serial communications setup:

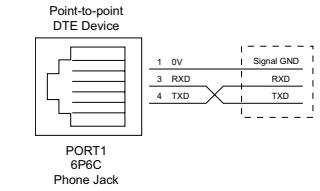
- Wiring standard
- Communications protocol
- Communications parameters

Each of these will be discussed in more detail as they apply to *Direct*LOGIC PLCs.

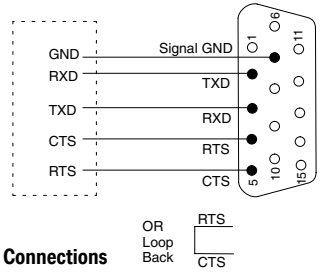
Wiring Standards

There are three different wiring standards that can be used with *Direct*LOGIC PLCs: RS-232C, RS-422 and RS-485.

RS-232C is a point-to-point wiring standard with a practical wiring distance of 15 meters (50 feet) maximum. This means that only two devices can communicate on an RS-232C network – a single master device and a single slave device. The total cable length cannot exceed 50 feet. AutomationDirect L19772 (Belden® 8102), or equivalent, is recommended for RS-232C networks.

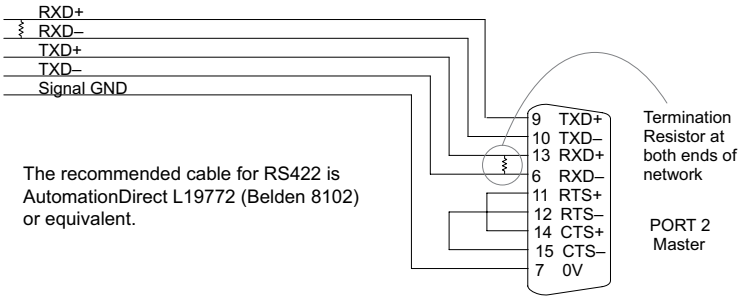


Port 1 Connections

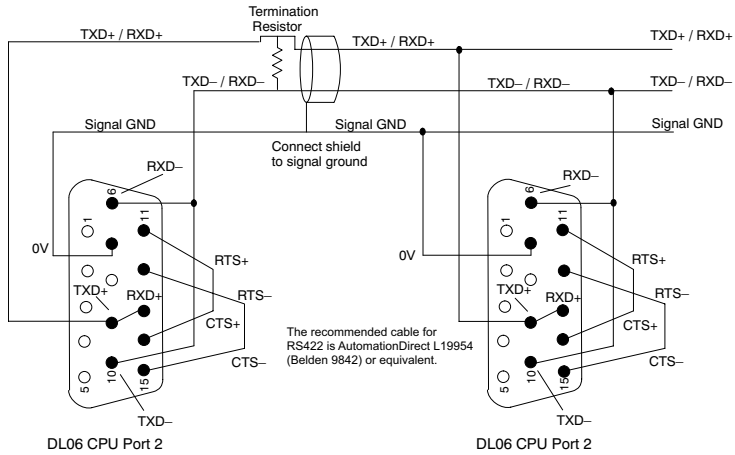


Port 2 Connections

RS-422 is a multi-point wiring standard with a practical wiring distance of 1000 meters (3280 feet) . The RS-422 wiring standard does not specify a network topology, but in practice, a daisy-chain topology with the master at one end is the only way to ensure network reliability. AutomationDirect L19772 (Belden® 8102), or equivalent, is recommended for RS-422 networks. Use a terminating resistor equal in value to the characteristic impedance of the cable being used (100 Ω for AutomationDirect L19772 [Belden® 8102]).



RS-485 is a multi-point wiring standard with a practical wiring distance of 4000 feet maximum. This wiring standard provides for the possibility of up to 32 masters communicating to up to 32 slaves, all within the maximum distance of 4000 feet. Note that while the RS-485 wiring standard provides for multiple masters on the same network, **the *Direct*LOGIC PLCs do not support multiple masters on a single network.** The RS-485 wiring standard does not specify a network topology, but in practice, a daisy-chain topology with the master at one end is the only way to ensure network reliability. AutomationDirect L19954 (Belden 9842), or equivalent is recommended for RS-485 networks. Use a terminating resistor equal in value to the characteristic impedance of the cable being used (120Ω for AutomationDirect L19954 [Belden 9842]).



Communications Protocols

A communications protocol is the language the devices on a network use to communicate with each other. All the devices on the network must use the same communications protocol in order to be able to communicate with each other. The protocols available in the *Direct*LOGIC PLCs are listed in the following table.

Communications Protocols							
Protocol	**Master**	**Slave**	**Port 1***	**Port 2**	**RS-232C**	**RS-422**	**RS-485****
K-Sequence	No	Yes	Yes	Yes	Yes	Yes	No
*Direct*NET	Yes	Yes	Yes	Yes	Yes	Yes	No
MODBUS RTU	Yes	Yes	Yes	Yes	Yes	Yes	Yes
ASCII (Non-Sequence)	Out	In	No	Yes	Yes	Yes	Yes

*** Port 1 supports slave only and is only RS-232C with fixed communications parameters of 9600 baud, 8 data bits, 1 start bit, 1 stop bit, odd parity and station address 1. It is an asynchronous, half-duplex DTE port and auto-selects between K-Sequence, DirectNET and MODBUS RTU protocols.**

**** RS-485 is available on Port 2 for MODBUS RTU and Non-Sequence protocol.**

K-Sequence protocol is not available for use by a master DL06 PLC. Therefore, it cannot be used for networking between PLCs. Its primary use in the DL06 PLC is as a slave to *Direct*SOFT programming software and to an operator interface.

***Direct*NET** protocol is available for use by a master or by a slave DL06 PLC. This, and the fact that it is *native* protocol, makes it ideal for PLC-to-PLC communication over a point-to-point to multipoint network using the RX and WX instructions.

MODBUS RTU protocol is a very common industry standard protocol, and can be used by a master or slave DL06 to communicate with a wide variety of industrial devices which support this protocol.

ASCII (Non-Sequence) is another very common industry standard protocol, and is commonly used where alpha-numeric character data is to be transferred. Many input devices, such as, barcode readers and electronic scales use ASCII protocol, and many output devices accept ASCII commands, as well.

It doesn't matter which wiring standard or protocol is used, there are several communications parameters to select for each device before they will be able to communicate. These parameters include:

Baud Rate	Flow Control
Data Bits	Echo Suppression
Parity	Timeouts
Stop Bits	Delay Times
Station Address	Format

All of these parameters may not be necessary, or available, for your application. The parameters used will depend on the protocol being used and whether the device is a master or slave.

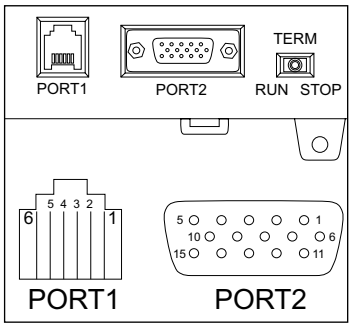
***NOTE:** REMEMBER, When the same parameter is available in the master and in the slave (i.e. Baud Rate, Parity, Stop Bits, etc.) the settings must match.*

DL06 Port Specifications

Communications Port 1	
Port 1	Connects to HPP, *Direct*SOFT, operator interfaces, etc.
	6-pin, RS232C
	Communication speed (baud): 9600 (fixed)
	Parity: odd (fixed)
	Station Address: 1 (fixed)
	8 data bits
	1 start, 1 stop bit
	Asynchronous, half-duplex, DTE
	Protocol (auto-select): K-sequence (slave only), *Direct*NET (slave only), MODBUS (slave only)

Communications Port 2	
Port 2	Connects to HPP, *Direct*SOFT, operator interfaces, etc.
	15-pin, multifunction port, RS232C, RS422, RS485 (RS485 with 2-wire is only available for MODBUS and Non-sequence)
	Communication speed (baud): 300, 600, 1200, 2400, 4800, 9600, 19200, 38400
	Parity: odd (default), even, none
	Station Address: 1 (default)
	8 data bits
	1 start, 1 stop bit
	Asynchronous, half-duplex, DTE
	Protocols can be pre-selected: K-sequence (slave only), *Direct*NET (master/slave), MODBUS (master/slave), Non-Sequence/Print/ASCII in/out

DL06 Port Pinouts



Port 1 Pin Descriptions		
1	0V	Power (-) connection (GND)
2	5V	Power (+) connection
3	RXD	Receive data (RS-232C)
4	TXD	Transmit data (RS-232C)
5	5V	Power (+) connection
6	0V	Power (-) connection (GND)

Port 2 Pin Descriptions		
1	5V	Power (+) connection
2	TXD	Transmit data (RS-232C)
3	RXD	Receive data (RS-232C)
4	RTS	Ready to send (RS-232C)
5	CTS	Clear to send (RS232C)
6	RXD-	Receive data (-) (RS-422/485)
7	0V	Power (-) connection (GND)
8	0V	Power (-) connection (GND)
9	TXD+	Transmit data (+) (RS-422/485)
10	TXD-	Transmit data (-) (RS-422/485)
11	RTS+	Ready to send (+) (RS-422/485)
12	RTS-	Ready to send (-) (RS-422/485)
13	RXD+	Receive data (+) (RS-422/485)
14	CTS+	Clear to send (+) (RS-422/485)
15	CTS-	Clear to send (-) (RS-422/485)

Note that the default configuration for port 2 is:

- Auto-detect among K-Sequence, DirectNET, and MODBUS RTU protocols
- Timeout = Base Timeout x 1 (800 ms)
- RTS on delay time = 0 ms
- RTS off delay time = 0 ms
- Station Number = 1
- Baud rate = 19200
- Stop bits = 1
- Parity = odd
- Format = Hex
- Echo Suppression = RS-422/485 (4-wire) or RS-232C

Port Setup Using DirectSOFT or Ladder Logic Instructions

Port 2 on the DL06 can be configured for communications using the various protocols which have been previously mentioned. Also, the communications parameters can be configured to match the parameters in the other device(s) with which the PLC will be communicating. The port may be configured using the ***Direct***SOFT PLC programming software, or by using ladder logic within the PLC program. It is important to note that the settings for port 2 are never saved to disk with ***Direct***SOFT, so if you are using port 2 in other than its default configuration (see page K-6) it is a good idea to include the port setup in the ladder program, typically on a first scan bit, or in an initialization subroutine.

To setup port 2 using ***Direct***SOFT, the PLC must be turned on and connected to ***Direct***SOFT. If the PLC Setup toolbar is displayed, either select the **Port 2** button or select **PLC > Setup > Setup Sec. Comm Port...** from the menu bar located at the top of the programming window. A dialog box like the one below will appear. Make the appropriate settings and write them to the PLC.

In order to setup port 2 in your relay ladder logic, the appropriate values must be written to V7655 (Word 1), V7656 (Word 2) and V7650 (Word 3, for ASCII only) to specify the settings for the port. Then write the 'setup complete' flag (K0500) to V7657 (Word 4) to request the CPU to accept the port settings. Once the CPU sees the 'setup complete' flag in V7657 it will test the port settings which have selected for validity, and then change the value in V7657 according to the results of this test. If the port settings are valid, the CPU will change the value in V7657 to 0A00 (**A** for Accepted). If there was an error in the port settings, the CPU will change the value in V7657 to 0E00 (**E** for Error).

NOTE: This is a Helpful Hint: *Rather than build the setup words manually from the tables, use* ***Direct****SOFT to setup the port as desired then use a Dataview to view the setup words as BCD/HEX. Then simply use these numbers in the setup code.*

The data that is written to the port setup words has two formats. The format that is used depends on whether K-Sequence, ***Direct***NET, MODBUS RTU (method 1) or ASCII (method 2) is selected.

Port 2 Setup for RLL Using K-Sequence, DirectNET or MODBUS RTU

V7655 (Word 1)	RTS On-delay	Timeout (% of timeout)	Protocol	RTS Off-delay
0yyy 0ttt mmmm mxxx	yyy	ttt	mmmmm	xxx
	000 = 0ms	000 = 100%	10000 = K-Sequence	000 = 0ms
	001 = 2ms	001 = 120%	01000 = *Direct*NET	001 = 2ms
	010 = 5ms	010 = 150%	00100 = MODBUS RTU	010 = 5ms
	011 = 10ms	011 = 200%		011 = 10ms
	100 = 20ms	100 = 500%		100 = 20ms
	101 = 50ms	101 = 1000%		101 = 50ms
	110 = 100ms	110 = 2000%		110 = 100ms
	111 = 500ms	111 = 5000%		111 = 500ms

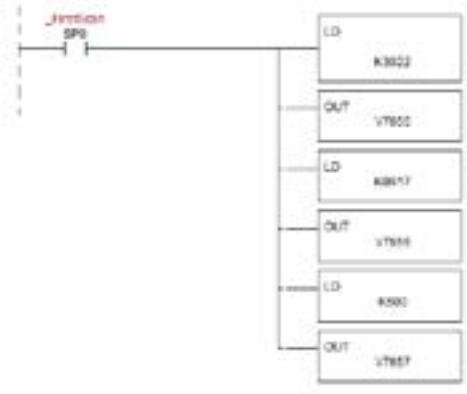
V7656 (Word 2)	Parity	Stop Bits	Echo Suppression	Baud Rate
K-Sequence, *Direct*NET & MODBUS RTU				
pps0 ebbb xaaa aaaa	pp	s	e	bbb
	00 = None	0 = 1 bit	0 = RS232/RS422/RS485 (4-wire)	000 = 300
	10 = Odd	1 = 2 bits	1 = RS485, 2 wire	001 = 600
	11 = Even			010 = 1200
				011 = 2400
				100 = 4800
				101 = 9600
				110 = 19200
				111 = 38400

V7656 (Word 2) cont'd	Protocol	Secondary Address
K-Sequence, *Direct*NET & MODBUS RTU	(*Direct*NET)	xaaaaaaa (*Direct*NET)
pps0 ebbb xaaa aaaa	x	aaaaaaa (K-Sequence & MODBUS RTU)
	0 = Hex	K-Sequence: 1-90
	1 = ASCII	*Direct*NET: 1-90
		MODBUS: 1-247

V7650 (Word 3)	V-memory address for data
DL05/DL06	For Non-Sequence (ASCII) only

V7657 (Word 4)	Setup and Completion Code
DL05/DL06	Write K0500 to accept Port 2 setup. When PLC accepts the changes, it changes the value to K0A00 in the same location. If there is an error it changes the value to K0E00 in the same location.

To setup port 2 for MODBUS protocol for the following: RTS On-delay of 10ms, Base timeout x1, RTS Off-delay of 5ms, Odd parity, 1 Stop bit, echo suppression for RS232-C/ RS422, 19,200 baud, Station Number 23 you would use the relay ladder logic shown below.



Port 2 Setup for RLL Using ASCII (Non-Sequence)

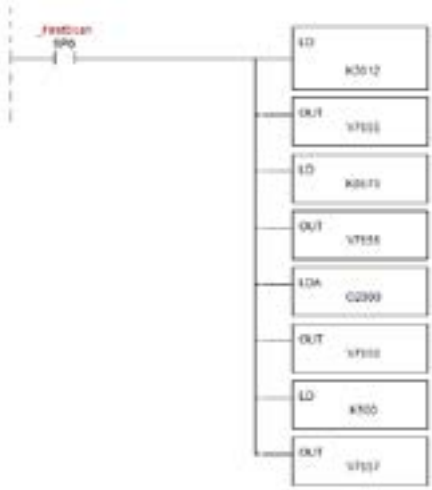
Word 1	RTS On-delay	Timeout (in% of std. timeout)	Protocol	RTS Off-delay
0yyy 0ttt mmmm mxxx	yyy	ttt	mmmmm	xxx
DL05/06: V7655	000 = 0ms	000 = Base timeout	00010 = Non-Sequence	000 = 0ms
	001 = 2ms	001 = Base timeout + 2ms		001 = 2ms
	010 = 5ms	010 = Base timeout + 5ms		010 = 5ms
	011 = 10ms	011 = Base timeout + 10ms		011 = 10ms
	100 = 20ms	100 = Base timeout + 20ms		100 = 20ms
	101 = 50ms	101 = Base timeout + 50ms		101 = 50ms
	110 = 100ms	110 = Base timeout + 100ms		110 = 100ms
	111 = 500ms	111 = Base timeout + 500ms		111 = 500ms

Word 2	Parity	Stop Bits	Data Bits	Echo Suppression (valid for DL06 only)	Baud Rate	Protocol Mode
ppsd ebbb xaaa aaaa	pp	s	d	e	bbb	aaaa aaaa
DL05/06: V7656	00 = None	0 = 1 bit	0 = 8 bits	0 = RS-232/RS-422/ RS-485 (4 wire)	000 = 300	0111 0000 = No flow control
	10 = Odd	1 = 2 bits	1 = 7 bits	1 = RS-485 (2 wire)	001 = 600	0111 0001 = Xon/Xoff flow control
	11 = Even				010 = 1200	0111 0010 RTS flow control
					011 = 2400	0111 0011 = Xon/Xoff and RTS flow control
					100 = 4800	
					101 = 9600	
					110 = 19200	
					111 = 38400	

Word 3	V-memory address for data
DL05/06: V7650	Hex value of the V-memory location to temporarily store the ASCII data coming into the PLC. Set this parameter to the start of a contiguous block of 64 unused words.

Word 4	Setup and Completion Code
DL05/06: V7657	Write K0500 to accept Port 2 setup. When PLC accepts the changes, it changes the value to K0A00 in the same location. If there is an error, it changes the value to K0E00 in the same location.

To setup port 2 for Non-sequence (ASCII) communications with the following: RTS On-delay of 10ms; Base timeout x1; RTS Off-delay of 5ms; Odd parity; 1 Stop bit; echo suppression for RS232-C/RS422; 19,200 baud; 8 data bits; V-memory buffer starting at V2000; and no flow control, you would use the relay ladder logic shown below.



K-Sequence Communications

The K-Sequence protocol can be used for communication with *Direct*SOFT, an operator interface or any other device that can be a K-Sequence master. The DL06 PLC can be a K-Sequence **slave** on either port 1 or port 2. The DL06 PLC cannot be a K-Sequence **master.**

In order to use port 2 for K-Sequence communications you first need to set up the port using either *Direct*SOFT or ladder logic as described above.

DirectNET Communications

The *Direct*NET protocol can be used to communicate to another PLC or to other devices that can use the *Direct*NET protocol. The DL06 can be used as either a master, using port 2 or as a slave, using either port 1 or port 2.

In order to use port 2 for *Direct*NET communications you must first set up the port using either *Direct*SOFT or ladder logic as previously described.

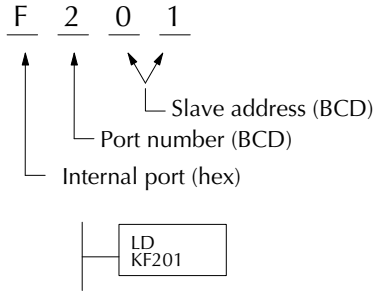
For network slave operation, nothing more needs to be done. Port 2 will function as a slave unless network communications instructions are executed by the ladder logic program.

For a network master operation you will simply need to add some ladder rungs using the network communication instructions RX and/or WX. If more than one network communication instruction is used, the rungs need to be interlocked to ensure that only one communication instruction is executed at any given time. If you have just a few network communications instructions in your program, you can use discrete bits to interlock them. If you are using many network communications instructions, a counter or a shift register will be a more convenient way to interlock the instructions.

The following step-by-step procedure will provide the information necessary to set up your ladder program to receive data from a network slave.

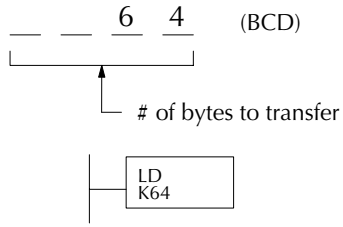
Step 1: Identify Master Port # and Slave

The first Load (LD) instruction identifies the communications port number on the network master (DL06) and the address of the slave station. This instruction can address up to 99 MODBUS slaves, or 90 *Direct*NET slaves. The format of the word is shown to the right. The **F2** in the upper byte indicates the use of the right port of the DL06 PLC, port number 2. The lower byte contains the slave address number in BCD (01 to 99).



Step 2: Load Number of Bytes to Transfer

The second Load (LD) instruction determines the number of bytes which will be transferred between the master and slave in the subsequent WX or RX instruction. The value to be loaded is in BCD format (decimal), from 1 to 128 bytes.



The number of bytes specified also depends on the type of data you want to obtain. For example, the DL06 Input points can be accessed by V-memory locations or as X input locations. However, if you only want X0 – X27, you'll have to use the X input data type because the V-memory locations can only be accessed in 2-byte increments. The following table shows the byte ranges for the various types of *Direct*LOGIC products.

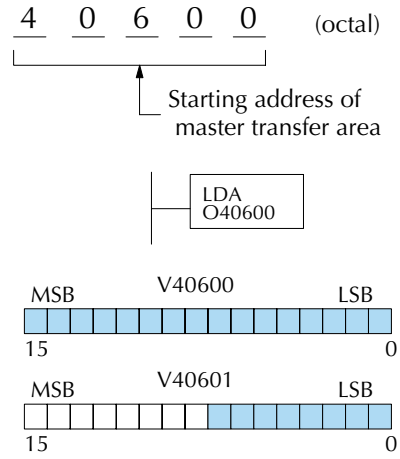
DL05 / 06 / 205 / 350 / 405 Memory	Bits per unit	Bytes
V-memory T / C current value	16 16	2 2
Inputs (X, SP)	8	1
Outputs (Y, C, Stage, T/C bits)	8	1
Scratch Pad Memory	8	1
Diagnostic Status	8	1

DL330 / 340 Memory	Bits per unit	Bytes
Data registers T / C accumulator	8 16	1 2
I/O, internal relays, shift register bits, T/C bits, stage bits	1	1
Scratch Pad Memory	8	1
Diagnostic Status(5 word R/W)	16	10

The third instruction in the RX or WX sequence is a Load Address (LDA) instruction. Its purpose is to load the starting address of the memory area to be transferred. Entered as an octal number, the LDA instruction converts it to hex and places the result in the accumulator.

For a WX instruction, the DL06 CPU sends the number of bytes previously specified from its memory area beginning at the LDA address specified.

For an RX instruction, the DL06 CPU reads the number of bytes previously specified from the slave, placing the received data into its memory area beginning at the LDA address specified.

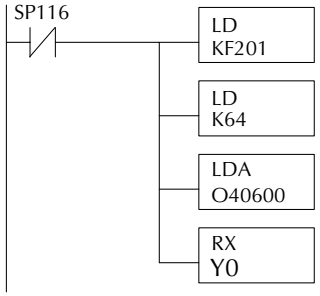


Step 3: Specify Master Memory Area

***NOTE**: Since V-memory words are always 16 bits, you may not always use the whole word. For example, if you only specify 3 bytes and you are reading Y outputs from the slave, you will only get 24 bits of data. In this case, only the 8 least significant bits of the last word location will be modified. The remaining 8 bits are not affected.*

Step 4: Specify Slave Memory Area

The last instruction in our sequence is the WX or RX instruction itself. Use WX to write to the slave, and RX to read from the slave. All four of our instructions are shown to the right. In the last instruction, you must specify the starting address and a valid data type for the slave.

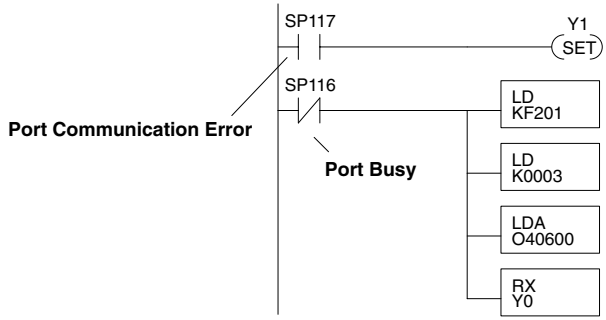


- DirectNET slaves – specify the same address in the WX and RX instruction as the slave's native I/O address
- MODBUS DL405, DL205, or DL06 slaves – specify the same address in the WX and RX instruction as the slave's native I/O address
- MODBUS 305 slaves – use the following table to convert DL305 addresses to MODBUS addresses.

DL305 Series CPU Memory Type–to–MODBUS Cross Reference (excluding 350 CPU)					
PLC Memory Type	**PLC Base Address**	**MODBUS Base Address**	**PLC Memory Type**	**PLC Base Address**	**MODBUS Base Address**
TMR/CNT Current Values	R600	V0	TMR/CNT Status Bits	CT600	GY600
I/O Points	IO 000	GY0	Control Relays	CR160	GY160
Data Registers	R401,R400	V100	Shift Registers	SR400	GY400
Stage Status Bits (D3-330P only)	S0	GY200			

Communications from a Ladder Program

Typically, network communications will last longer than 1 scan. The program must wait for the communications to finish before starting the next transaction.



Port 2, which can be a master, has two Special Relay contacts associated with it (see Appendix D for comm port special relays). One indicates "Port busy"(SP116), and the other indicates "Port Communication Error"(SP117). The example above shows the use of these contacts for a network master that only reads a device (RX). The "Port Busy" bit is on while the PLC communicates with the slave. When the bit is off the program can initiate the next network request.

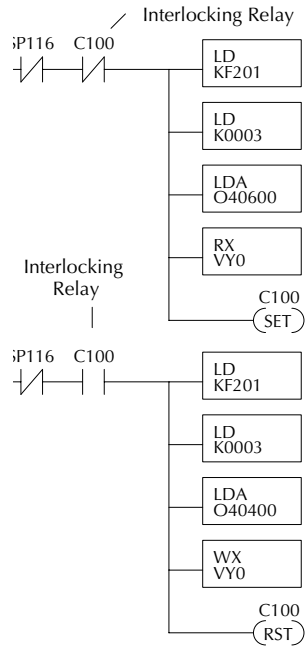
The "Port Communication Error" bit turns on when the PLC has detected an error. Use of this bit is optional. When used, it should be ahead of any network instruction boxes since the error bit is reset when an RX or WX instruction is executed.

Multiple Read and Write Interlocks

If you are using multiple reads and writes in the RLL program, you have to interlock the routines to make sure all the routines are executed. If you don't use the interlocks, then the CPU will only execute the first routine. This is because each port can only handle one transaction at a time.

In the example to the right, after the RX instruction is executed, C100 is set. When the port has finished the communication task, the second routine is executed and C100 is reset.

If you're using RLL*PLUS* Stage Programming, you can put each routine in a separate program stage to ensure proper execution and switch from stage to stage allowing only one of them to be active at a time.



MODBUS RTU Communications

The MODBUS RTU protocol can be used for communication with any device that uses the MODBUS RTU protocol. The protocol is very common and is probably the closest thing to an "industry standard" protocol in existence. The DL06 can be a MODBUS RTU slave on either port 1 or port 2, and it can be a MODBUS RTU master on port 2. The RS 485 wiring standard may be used on port 2 for the MODBUS RTU protocol only.

In order to use port 2 for MODBUS RTU communications you must first set up the port using either ***Direct***SOFT or ladder logic as previously described.

For network slave operation, nothing more needs to be done. Port 2 will function as a slave unless network communications instructions are executed by the ladder logic program.

For network master operation the MODBUS RTU network communication instructions MRX and/or MWX must be added to some ladder rungs. If more than one network communication instruction is used, the rungs need to be interlocked to ensure that only one communication instruction is executed at any given time. If only a few network communications instructions are used in your program, discrete bits can be used to interlock them. If many network communications instructions are used, either a counter or a shift register will be a more convenient way to interlock the instructions.