# Errata Sheet

## This Errata Sheet contains corrections or changes made after the publication of this manual.

| | | | |
|---|---|---|---|
| **Product Family**: | DL405 | **Date**: | December 11, 2018 |
| **Manual Number** | D4-USER-M | | |
| **Revision and Date** | 4th Edition, Rev. A; March 2013 | | |

**Changes to Chapter 2. Installation, Wiring, and Specifications**

Page 2-4. Safety Guidelines; Class 1, Division 2 Approval

*Delete this section, including the two warnings. The DL405 system is NOT Class 1, Division 2 approved.*

Page 2-44. D4-32TD1-1, 5-15VDC Output

*Add the following NOTE to the specifications table for this module:*

**NOTE:The D4-32TD1-1 module operates on reverse logic. Voltage is present when Output is Off; no voltage is present when output is On.**

**Changes to Chapter 4. System Design and Configuration**

Page 4-8. Calculating the Power Budget; Module Power Requirements

*Change the "5V Current Required (mA)" values in the table for these analog modules:*

*F4-04AD: Change from 85 mA to 150 mA*

*F4-04ADS: Change from 270 mA to 370 mA*

Page 4-18. Network Connections to MODBUS and *Direct*NET; Configuring the CPU's Comm Ports

*Change the NOTE near the bottom of the page to say:*

*NOTE: The recommended cable for RS-232 and RS-422 is AutomationDirect L19772-1 (Belden 8102) or equivalent.*

**Changes to Chapter 5. Standard RLL Instructions; Accumulator Logic Instructions**

Page 5-85. Compare with Stack (CMPS)

*The table below has more accurate descriptions for the discrete bit flags than the descriptions in the manual:*

| Discrete Bit Flags | Description |
|---|---|
| SP60 | On when the value in the Accumulator is less than the first level value in the Accumulator Stack. |
| SP61 | On when the value in the Accumulator is equal to the first level value in the Accumulator Stack. |
| SP62 | On when the value in the Accumulator is greater than the first level value in the Accumulator Stack. |

# Errata Sheet

## This Errata Sheet contains corrections or changes made after the publication of this manual.

### Changes to Chapter 8. PID Loop Operation (DL450 only)

Page 8-36. Monitor Limit Alarms. Add the following note at the end of this page:

**NOTE: PID deviation alarm only works in Auto mode.**

### Changes to Chapter 9. Maintenance and Troubleshooting

Page 9-26. Add the following to the end of this chapter (right after BREAK Instruction placement):

**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 1 – 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".
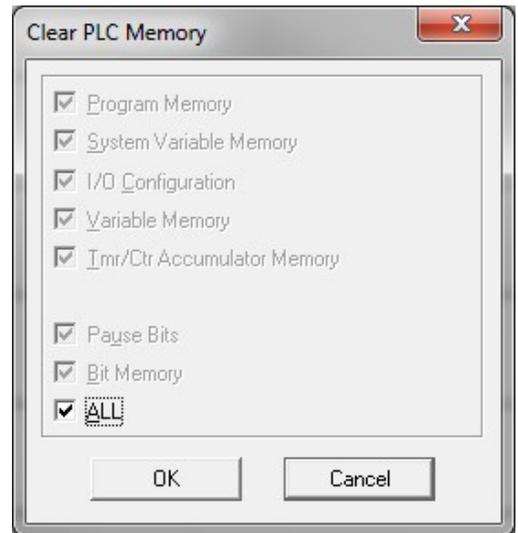
Step 2 – While connected with DirectSoft, go the PLC menu and then to the "Setup" submenu and select "Initialize Scratch Pad". Press "Ok".

**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.

# Errata Sheet

## This Errata Sheet contains corrections or changes made after the publication of this manual.

**Changes to Appendix I. European Union Directives (CE)**

Page I-4. Special Installation Manual

*Delete this section. Manual D-EU-M is no longer available. Instead, refer to the websites listed below:*

Page I-4: Other Sources of Information

*Replace the text that follows the sentence "It may be possible to obtain this information locally; however, the official source of applicable Directives and related standards is:" with the following updated information:*

Publications Office

2, rue Mercier

2985 Luxembourg

LUXEMBOURG

Quickest contact is via the web at:

http://ec.europa.eu/growth/single-market/european-standards/harmonised-standards

Another source is the British Standards Institution at:

British Standards Institution – Sales Department, Linford Wood:

Milton Keynes, MK14 6LE, United Kingdom.

The quickest contact is via the web at http://www.bsigroup.com

Another source is

The 'Blue Guide' on the implementation of EU product rules 2016

https://ec.europa.eu/commission/index_en

Page I-10. Items Specific to the DL405

*Ignore the reference to manual D-EU-M in the first paragraph of this section. That manual is no longer available. See the websites listed above.*

# Standard RLL Instructions

**5**

In This Chapter. . . .

# Introduction

The DL405 instruction set can perform many different types of operations. This chapter shows you how to use these individual instructions. The following table provides a quick reference listing of the instruction mnemonic and the page(s) defining the instruction. Each instruction definition will show in parentheses the HPP keystrokes used to enter the instruction. There are two ways to locate instructions:

- If you know the instruction category (Boolean, Comparative Boolean, etc.), just use the header at the top of the page to find the pages that discuss the instructions in that category.

- If you know the individual instruction mnemonic, use the following table.

The DL450 provides all of the instructions in the table, the DL440 provides a subset, and the DL430 a smaller subset. The instruction definitions indicate which CPUs feature the instruction. In Example 1, only the DL440 and DL450 have the instruction. In Example 2, all CPUs have the instruction.

| Example 1 | Example 2 |
|---|---|
| ✗ ✓ ✓ | ✓ ✓ ✓ |
| **430 440 450** | **430 440 450** |

If you are using a DL450 PLC (with firmware v3.30 or later) with *Direct*SOFT5 programming software, you can also use the IBox instructions covered in the DL405–IBOX–S Supplement.

| Instruction | Page |
|---|---|
| ACON | 5-196 |
| ACOSR | 5-119 |
| ADD | 5-87 |
| ADDB | 5-99 |
| ADDBD | 5-100 |
| ADDBS | 5-113 |
| ADDD | 5-88 |
| ADDF | 5-105 |
| ADDR | 5-89 |
| ADDS | 5-109 |
| AND | 5-14, 5-31, 5-70 |
| ANDD | 5-71 |
| ANDND | 5-23 |
| ANDS | 5-73 |
| ANDSTR | 5-16 |
| ANDB | 5-15 |
| ANDD | 5-71 |
| ANDE | 5-28 |
| ANDF | 5-72 |
| ANDI | 5-34 |
| ANDMOV | 5-171 |
| ANDN | 5-14, 5-31 |
| ANDNB | 5-15 |
| ANDNE | 5-28 |
| ANDNI | 5-34 |
| ANDPD | 5-23 |
| ANDS | 5-73 |

| Instruction | Page |
|---|---|
| ANDSTR | 5-16, 5-118 |
| ASINR | 5-118 |
| ATANR | 5-119 |
| ATH | 5-135 |
| ATT | 5-159 |
| BCALL | 7-27 |
| BCD | 5-129 |
| BCDCPL | 5-131 |
| BEND | 7-27 |
| BIN | 5-128 |
| BLK | 7-27 |
| BREAK | 5-178 |
| BTOR | 5-132 |
| CMP | 5-82 |
| CMPD | 5-83 |
| CMPF | 5-84 |
| CMPR | 5-86 |
| CMPS | 5-85 |
| CNT | 5-47 |
| COSR | 5-118 |
| CV | 7-25 |
| CVJMP | 7-25 |
| DATE | 5-175 |
| DEC | 5-117 |
| DECB | 5-120 |
| DECO | 5-127 |

| Instruction | Page |
|---|---|
| DEGR | 5-134 |
| DISI | 5-186 |
| DIV | 5-96 |
| DIVB | 5-104 |
| DIVBS | 5-116 |
| DIVD | 5-97 |
| DIVF | 5-108 |
| DIVR | 5-98 |
| DIVS | 5-112 |
| DLBL | 5-196 |
| DRUM | 6-15 |
| EDRUM | 6-18 |
| ENCO | 5-126 |
| END | 5-177 |
| ENI | 5-186 |
| FAULT | 5-195 |
| FDGT | 5-144 |
| FILL | 5-142 |
| FIND | 5-143 |
| FINDB | 5-173 |
| FOR | 5-180 |
| GOTO | 5-179 |
| GRAY | 5-139 |
| GTS | 5-181 |
| HISTRY | 5-197 |
| HTA | 5-136 |
| INC | 5-117 |

| Instruction | Page |
|---|---|
| INCB | 5-120 |
| INT | 5-185 |
| INV | 5-130 |
| IRT | 5-186 |
| IRTC | 5-186 |
| ISG | 7-24 |
| JMP | 7-24 |
| LBL | 5-179 |
| LD | 5-59 |
| LDA | 5-61 |
| LDD | 5-59 |
| LDF | 5-60 |
| LDI | 5-37 |
| LDIF | 5-38 |
| LDLBL | 5-162 |
| LDR | 5-64 |
| LDSX | 5-63 |
| LDX | 5-62 |
| MDRMD | 6-21 |
| MDRMW | 6-24 |
| MLR | 5-183 |
| MLS | 5-183 |
| MOV | 5-146 |
| MOVMC | 5-162 |
| MUL | 5-93 |
| MULB | 5-103 |
| MULBS | 5-115 |
| MULD | 5-94 |
| MULF | 5-107 |
| MULR | 5-95 |
| MULS | 5-111 |
| NCON | 5-197 |
| NEXT | 5-180 |
| NJMP | 7-24 |
| NOP | 5-177 |
| NOT | 5-19 |
| OR | 5-12, 5-30, 5-74 |
| ORB | 5-13 |
| ORD | 5-75 |
| ORE | 5-27 |
| ORF | 5-76 |
| ORI | 5-33 |
| ORMOV | 5-171 |
| ORN | 5-12, 5-30 |
| ORNB | 5-13 |

| Instruction | Page |
|---|---|
| ORND | 5-22 |
| ORNE | 5-27 |
| ORNI | 5-33 |
| OROUT | 5-19 |
| OROUTI | 5-35 |
| ORPD | 5-22 |
| ORS | 5-77 |
| ORSTR | 5-16 |
| OUT | 5-17, 5-65 |
| OUTB | 5-18 |
| OUTD | 5-65 |
| OUTF | 5-66 |
| OUTI | 5-35, 5-39 |
| OUTIF | 5-40 |
| OUTL | 5-68 |
| OUTM | 5-68 |
| OUTX | 5-67 |
| PAUSE | 5-20 |
| PD | 5-20 |
| POP | 5-69 |
| PRINT | 5-201 |
| RADR | 5-134 |
| RD | 5-189 |
| RFB | 5-150 |
| RFT | 5-156 |
| ROTL | 5-124 |
| ROTR | 5-125 |
| RST | 5-24 |
| RSTB | 5-25 |
| RSTBIT | 5-167 |
| RSTI | 5-36 |
| RSTWT | 5-178 |
| RT | 5-181 |
| RTC | 5-181 |
| RTOB | 5-133 |
| RX | 5-191 |
| SBR | 5-181 |
| SEG | 5-138 |
| SET | 5-24 |
| SETB | 5-25 |
| SETBIT | 5-167 |
| SETI | 5-36 |
| SFLDGT | 5-140 |
| SG | 7-23 |

| Instruction | Page |
|---|---|
| SGCNT | 5-49 |
| SHFL | 5-122 |
| SHFR | 5-123 |
| SINR | 5-118 |
| SQRTR | 5-119 |
| SR | 5-53 |
| STOP | 5-177 |
| STR | 5-10, 5-29 |
| STRB | 5-11 |
| STRE | 5-26 |
| STRI | 5-32 |
| STRN | 5-10, 5-29 |
| STRNB | 5-11 |
| STRND | 5-21 |
| STRNE | 5-26 |
| STRNI | 5-32 |
| STRPD | 5-21 |
| STT | 5-153 |
| SUB | 5-90 |
| SUBB | 5-101 |
| SUBBD | 5-102 |
| SUBBS | 5-114 |
| SUBD | 5-91 |
| SUBF | 5-106 |
| SUBR | 5-92 |
| SUBS | 5-110 |
| SUM | 5-121 |
| SWAP | 5-174 |
| TANR | 5-118 |
| TIME | 5-176 |
| TMR | 5-42 |
| TMRA | 5-44 |
| TMRAF | 5-44 |
| TMRF | 5-42 |
| TSHFL | 5-169 |
| TSHFR | 5-169 |
| TTD | 5-147 |
| UDC | 5-51 |
| WT | 5-190 |
| WX | 5-192 |
| XOR | 5-78 |
| XORD | 5-79 |
| XORF | 5-80 |
| XORMOV | 5-171 |
| XORS | 5-81 |

Standard
RLL Instructions
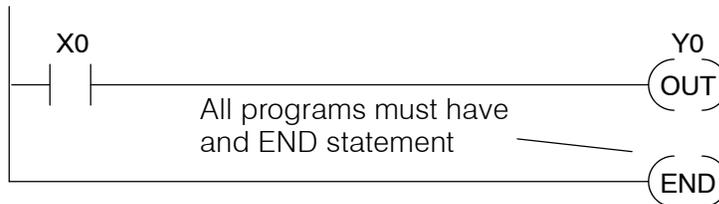
# Using Boolean Instructions

Do you ever wonder why so many PLC manufacturers always quote the scan time for a 1K boolean program? Most all 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; therefore, you don't necessarily have to know the instruction mnemonics in order to develop your program. However, knowledge of mnemonics will be helpful whenever it becomes necessary to troubleshoot the program using a handheld programmer (HPP).

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 box to indicate how the instruction is used with *Direct*SOFT and the HPP. The box will either contain **IMP** for implied, a ✓ for used or an ✗ for not used. The abbreviation, **DS**, or **HPP** will appear beneath the appropriate box as shown below.

| **IMP** | **✓** | **✗** |
|:---:|:---:|:---:|
| **DS** | **HPP** | **DS** |

**END Statement**

DL405 programs require an END statement (coil) as the last instruction. This tells the CPU 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. Chapter 5 discusses the instruction set in detail.



All programs must have
and END statement

**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.

*Direct*SOFT Example                    Handheld Mnemonics



STR X0
OUT Y0
END
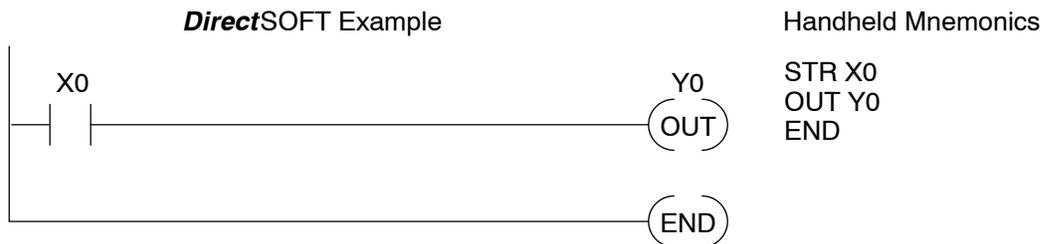
**Normally Closed Contact**

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.

*Direct*SOFT Example                                              Handheld Mnemonics

```
        X0                                    Y0          STRN X0
     ───┤/├────────────────────────────────( OUT )        OUT Y0
                                                           END


                                          ( END )
```

**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.

*Direct*SOFT Example                                              Handheld Mnemonics
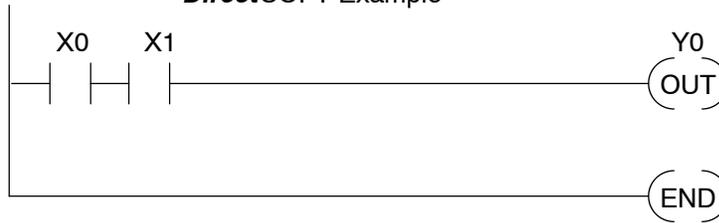
```
      X0    X1                               Y0          STR X0
    ──┤ ├──┤ ├─────────────────────────────( OUT )        AND X1
                                                           OUT Y0
                                                           END


                                          ( END )
```
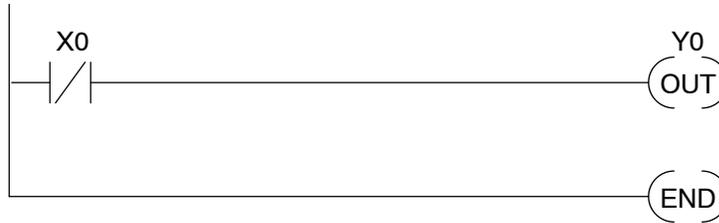
**Midline Outputs**

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.

*Direct*SOFT Example                                              Handheld Mnemonics

```
      X0    X1                               Y0          STR X0
    ──┤ ├──┤ ├──────────────────────────────( OUT )        AND X1
                   │                                        OUT Y0
                   │      X2                  Y1            AND X2
                   └─────┤ ├──────────┐     ( OUT )         OUT Y1
                                      │                     AND X3
                                 X3   │       Y2            OUT Y2
                                 └───┤ ├────( OUT )         END


                                          ( END )
```

**Parallel Elements**   You 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.

*Direct*SOFT Example                             Handheld Mnemonics

| X0 | Y0 (OUT) |
| X1 | |
| | (END) |

STR X0
OR X1
OUT Y0
END

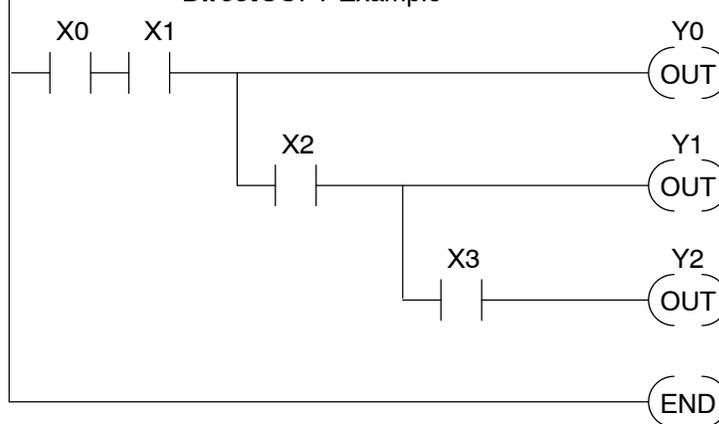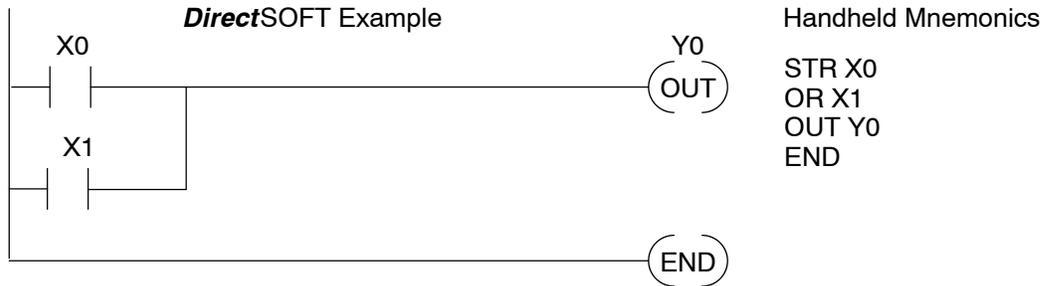**Joining Series Branches in Parallel**   Quite often it is necessary to join several groups of series elements in parallel. The Or Store (ORSTR) instruction allows this operation. The following example shows a simple network consisting of series elements joined in parallel.

*Direct*SOFT Example                             Handheld Mnemonics

| X0  X1 | Y0 (OUT) |
| X2  X3 | |
| | (END) |

STR X0
AND X1
STR X2
AND X3
ORSTR
OUT Y0
END
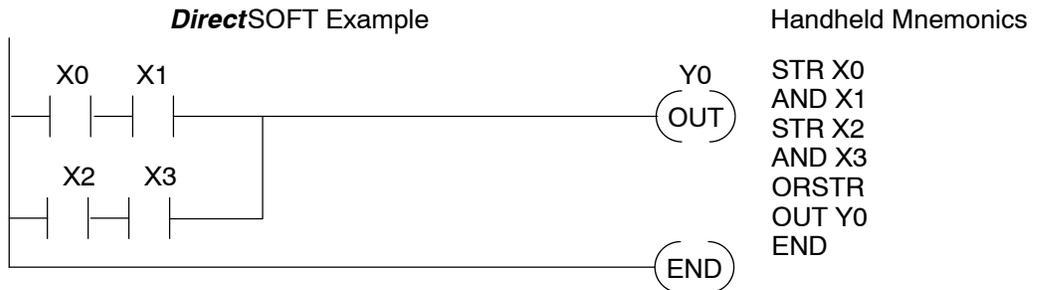
**Joining Parallel Branches in Series**   Quite often it is also necessary to join one or more parallel branches in series. The And Store (ANDSTR) instruction allows this operation. The following example shows a simple network with contact branches in series with parallel contacts.

*Direct*SOFT Example                             Handheld Mnemonics

| X0 | X1 | Y0 (OUT) |
| | X2 | |
| | | (END) |

STR X0
STR X1
OR X2
ANDSTR
OUT Y0
END
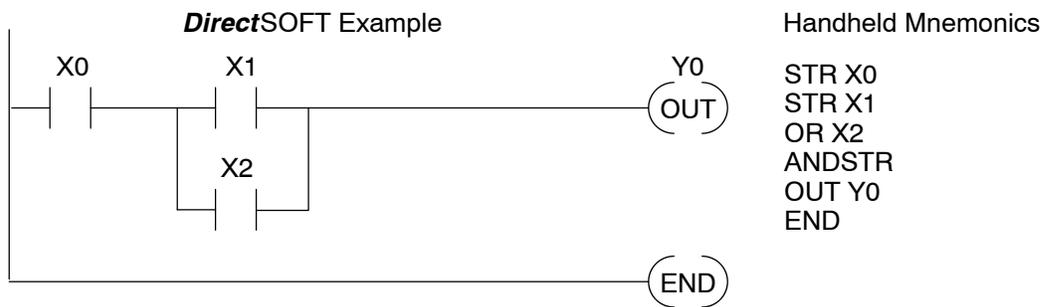
**Combination Networks**

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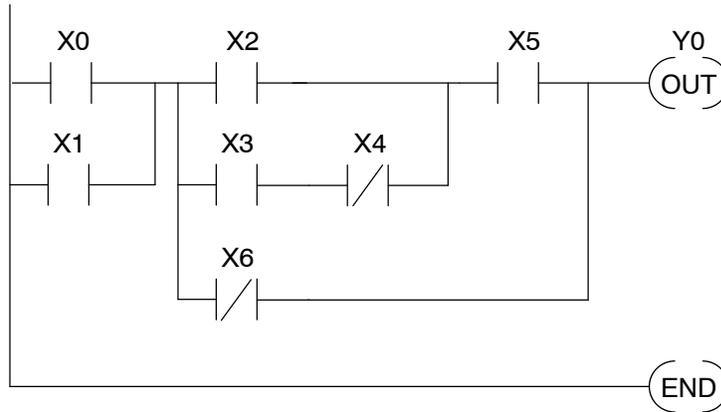


Handheld Mnemonics

STR X0
OR X1
STR X2
STR X3
ANDN X4
ORSTR
AND X5
ORN X6
ANDSTR
OUT Y0

**Comparative Boolean**

The Comparative Boolean evaluates two 4-digit BCD/hex values using boolean contacts. The valid evaluations are: equal to, not equal to, equal to or greater than, and less than.

In the following example when the value in V-memory location V1400 is equal to the constant BCD value 1234, Y3 will energize.
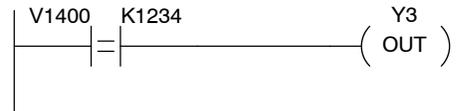
**Boolean Stack**  There are limits to how many elements you can include in a rung. This is because the DL405 CPUs use an 8-level boolean stack to evaluate the various logic elements. The boolean stack is a temporary storage area that solves the logic for the rung. Each time you enter a STR instruction, the instruction is placed on the top of the boolean stack. Any other STR instructions on the boolean stack are pushed down a level. The ANDSTR, and ORSTR instructions combine levels of the boolean stack when they are encountered. Since the boolean stack is only eight levels, an error will occur if the CPU encounters a rung that uses more than the eight levels of the boolean stack.

All of you software programmers may be saying, "I use *Direct*SOFT, so I don't need to know how the stack works." Not quite true. Even though you can build the network with the graphic symbols, the limits of the CPU are still the same. If the stack limit is exceeded when the program is compiled, an error will occur.

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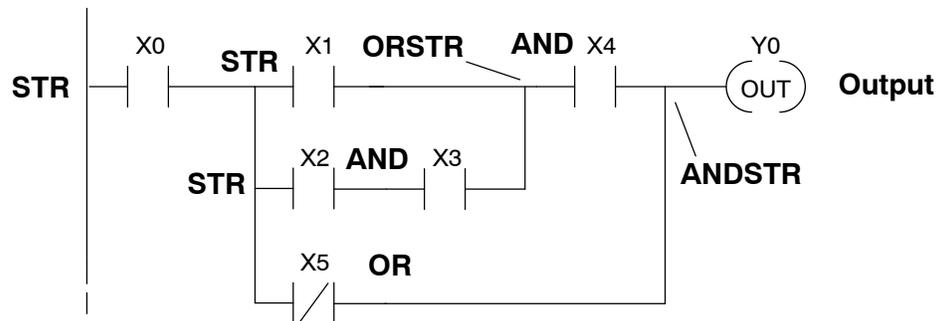


**STR X0**

| 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 | |

**OR X5**

| 1 | NOT X5 OR X4 AND [X1 OR (X2 AND X3)] |
|---|-------------------------------------|
| 2 | STR X0 |
| 3 | |

**ANDSTR**

| 1 | X0 AND (NOT X5 OR X4) AND [X1 OR (X2 AND X3)] |
|---|----------------------------------------------|
| 2 | |
| 3 | |

Standard
RLL Instructions

**Immediate Boolean**   The DL405 CPUs usually can 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 DL405 CPUs 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 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 module. 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 module, 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 module again to update the status. The immediate output instruction will write the status to the module and update the image register.

# Boolean Instructions

**Store (STR)**

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

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.

```
                    Aaaa
                    ─┤ ├─
```

**Store Not (STRN)**

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

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.

```
                    Aaaa
                    ─┤/├─
```

| Operand Data Type | | DL430 Range | DL440 Range | DL450 Range |
|---|---|---|---|---|
| | **A** | **aaa** | **aaa** | **aaa** |
| Inputs | X | 0–477 | 0–477 | 0–1777 |
| Outputs | Y | 0–477 | 0–477 | 0–1777 |
| Control Relays | C | 0–737 | 0–1777 | 0–3777 |
| Stage | S | 0–577 | 0–1777 | 0–1777 |
| Timer | T | 0–177 | 0–377 | 0–377 |
| Counter | CT | 0–177 | 0–177 | 0–377 |
| Special Relay | SP | 0–137, 320–617 | 0–137 320–717 | 0–137, 320–717 |
| Global | GX | 0–777 | 0–1777 | 0–2777 |
| Global | GY | – | – | 0–2777 |

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

*DirectSOFT*

```
        X1                              Y2
    ────┤ ├────────────────────────────( )
                                        OUT
```

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
|---|---|---|---|
| OUT | Y(OUT) | 2 | ↵ |

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

*DirectSOFT*

```
        X1                              Y2
    ────┤/├────────────────────────────( )
                                        OUT
```

Handheld Programmer Keystrokes

| STR | NOT | X(IN) | 1 | ↵ |
|---|---|---|---|---|
| OUT | Y(OUT) | 2 | ↵ | |

Standard
RLL Instructions

## Store Bit-of-Word (STRB)

| × | × | ✓ |
|---|---|---|
| **430** | **440** | **450** |

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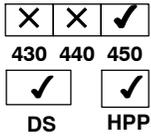
The Store Bit-of-Word instruction begins a new rung or an additional branch in a rung with a normally open contact. Status of the contact will be the same state as the bit referenced in the associated memory location.

Aaaa.bb

──┤ ├──
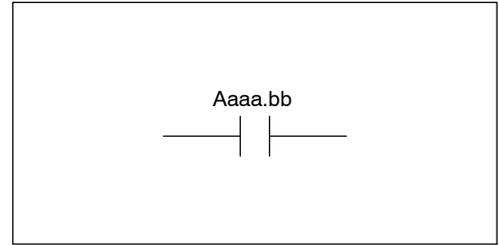
## Store Not Bit-of-Word (STRNB)
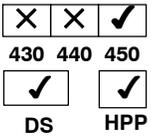
| × | × | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

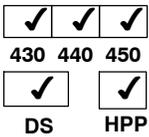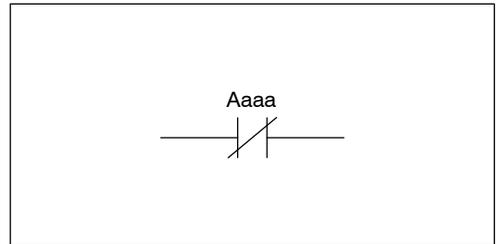The Store Not instruction begins a new rung or an additional branch in a rung with a normally closed contact. Status of the contact will be opposite the state of the bit referenced in the associated memory location.

Aaaa.bb

──┤/├──
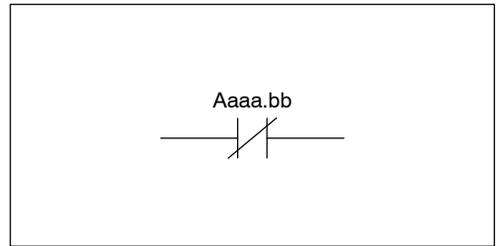
| Operand Data Type | | DL450 Range | |
|---|---|---|---|
| | **A** | **aaa** | **bb** |
| V–memory | B | All (See p. 3–42) | BCD, 0 to 15 |
| Pointer | PB | All (See p. 3–42) | BCD, 0 to 15 |

In the following Store Bit-of-Word example, when bit 12 of V-memory location V1400 is on, output Y2 will energize.

*DirectSOFT*

```
     B1400.12                            Y2
──────┤ ├──────────────────────────────( OUT )
```

Handheld Programmer Keystrokes

| STR | SHFT | B | SHFT | V | 1 | 4 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|

| K(con) | 1 | 2 | ↵ |
|---|---|---|---|

| OUT | Y(OUT) | 2 | ↵ |
|---|---|---|---|
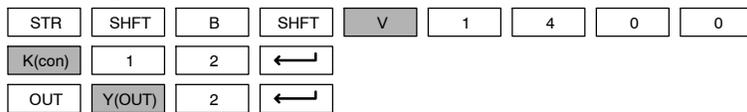
In the following Store Not Bit-of-Word example, when bit 12 of V-memory location V1400 is off, output Y2 will energize.

*DirectSOFT*

```
     B1400.12                            Y2
──────┤/├──────────────────────────────( OUT )
```

Handheld Programmer Keystrokes

| STR | STR | SHFT | B | SHFT | V | 1 | 4 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|

| K(con) | 1 | 2 | ↵ |
|---|---|---|---|

| OUT | Y(OUT) | 2 | ↵ |
|---|---|---|---|

**Or
(OR)**

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| **IMP** | ✓ |
|---------|---|
| **DS** | **HPP** |

The Or instruction logically ors 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.

Aaaa
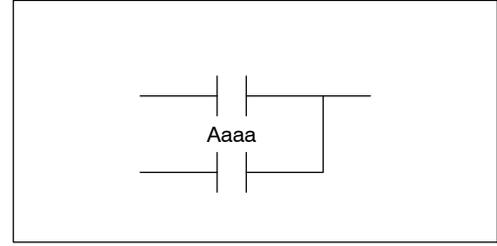
**Or Not
(ORN)**

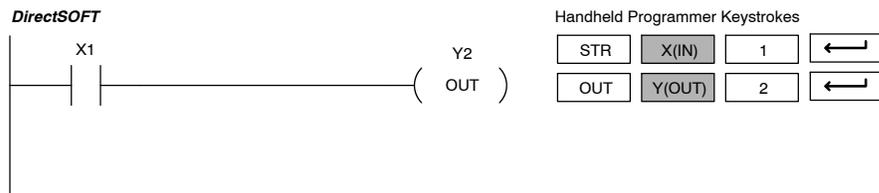| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| **IMP** | ✓ |
|---------|---|
| **DS** | **HPP** |

The Or Not instruction logically ors 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.
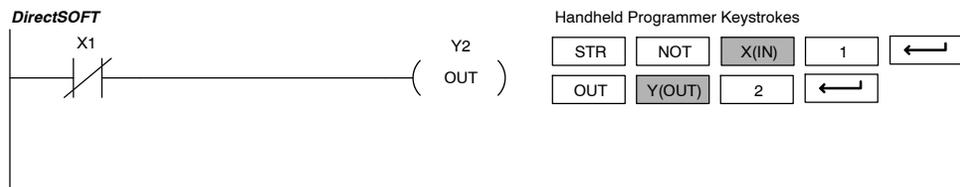
Aaaa

| Operand Data Type | | DL430 Range | DL440 Range | DL450 Range |
|---|---|---|---|---|
| | **A** | **aaa** | **aaa** | **aaa** |
| Inputs | X | 0–477 | 0–477 | 0–1777 |
| Outputs | Y | 0–477 | 0–477 | 0–1777 |
| Control Relays | C | 0–737 | 0–1777 | 0–3777 |
| Stage | S | 0–577 | 0–1777 | 0–1777 |
| Timer | T | 0–177 | 0–377 | 0–377 |
| Counter | CT | 0–177 | 0–177 | 0–377 |
| Special Relay | SP | 0–137, 320–617 | 0–137 320–717 | 0–137, 320–717 |
| Global | GX | 0–777 | 0–1777 | 0–1777 |
| Global | GY | – | – | 0–1777 |

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

**DirectSOFT**

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
| OR | X(IN) | 2 | ↵ |
| OUT | Y(OUT) | 5 | ↵ |

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

**DirectSOFT**

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ | |
| OR | NOT | X(IN) | 2 | ↵ |
| OUT | Y(OUT) | 5 | ↵ | |

## Or Bit-of-Word (ORB)

| × | × | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| IMP | ✓ |
|-----|---|
| **DS** | **HPP** |

The Or Bit-of-Word instruction logically ors 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.



Aaaa.bb
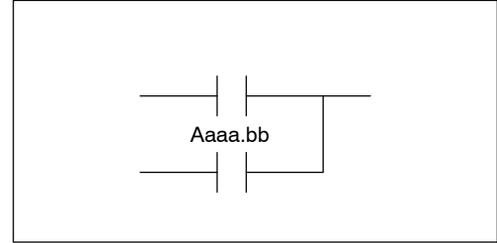
## Or Not Bit-of-Word (ORNB)

| × | × | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| IMP | ✓ |
|-----|---|
| **DS** | **HPP** |

The Or Not Bit-of-Word instruction logically ors 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.



Aaaa.bb

| Operand Data Type | | DL450 Range | |
|---|---|---|---|
| | **A** | **aaa** | **bb** |
| V–memory | B | All (See p. 3–42) | BCD, 0 to 15 |
| Pointer | PB | All (See p. 3–42) | BCD |

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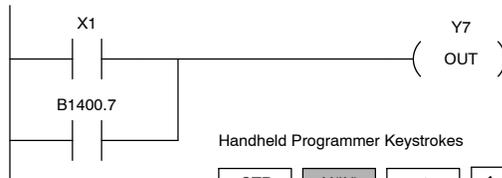
*DirectSOFT*



Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
| OR | SHFT | B | SHFT | V | 1 | 4 | 0 | 0 |
| K(con) | 7 | ↵ |
| OUT | Y(OUT) | 5 | ↵ |

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

*DirectSOFT*
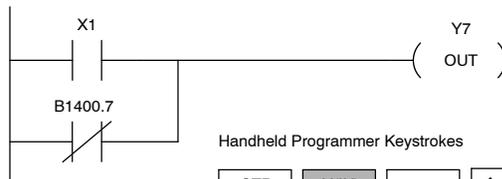


Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
| OR | SHFT | N | B | SHFT | V | 1 | 4 | 0 | 0 |
| K(con) | 7 | ↵ |
| OUT | Y(OUT) | 5 | ↵ |

Standard
RLL Instructions

**And
(AND)**

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| IMP | ✓ |
|---|---|
| **DS** | **HPP** |

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.

Aaaa

**And Not
(ANDN)**

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| IMP | ✓ |
|---|---|
| **DS** | **HPP** |

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 associated image register point or memory location.

Aaaa

| Operand Data Type | | DL430 Range | DL440 Range | DL450 Range |
|---|---|---|---|---|
| | **A** | **aaa** | **aaa** | **aaa** |
| Inputs | X | 0–477 | 0–477 | 0–1777 |
| Outputs | Y | 0–477 | 0–477 | 0–1777 |
| Control Relays | C | 0–737 | 0–1777 | 0–3777 |
| Stage | S | 0–577 | 0–1777 | 0–1777 |
| Timer | T | 0–177 | 0–377 | 0–377 |
| Counter | CT | 0–177 | 0–177 | 0–377 |
| Special Relay | SP | 0–137, 320–617 | 0–137 320–717 | 0–137, 320–717 |
| Global | GX | 0–777 | 0–1777 | 0–1777 |
| Global | GY | – | – | 0–1777 |

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

*DirectSOFT*

X1    X2                    Y5
                          ( OUT )
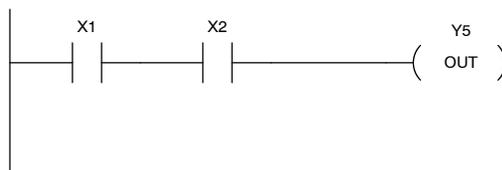
Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
|---|---|---|---|
| AND | X(IN) | 2 | ↵ |
| OUT | Y(OUT) | 5 | ↵ |

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

*DirectSOFT*

X1    X2                    Y5
                          ( OUT )
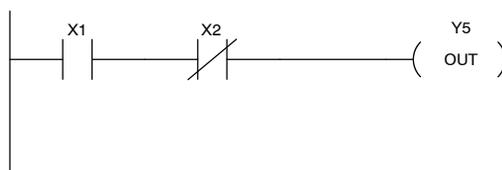
Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ | |
|---|---|---|---|---|
| AND | NOT | X(IN) | 2 | ↵ |
| OUT | Y(OUT) | 5 | ↵ | |

## And Bit-of-Word (ANDB)

| X | X | ✔ |
|---|---|---|
| **430** | **440** | **450** |

| IMP | ✔ |
|---|---|
| **DS** | **HPP** |

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.
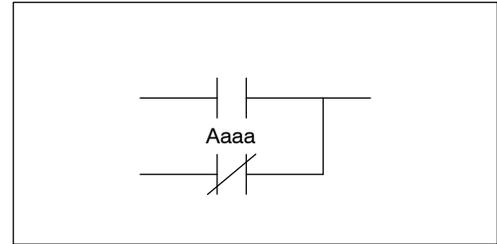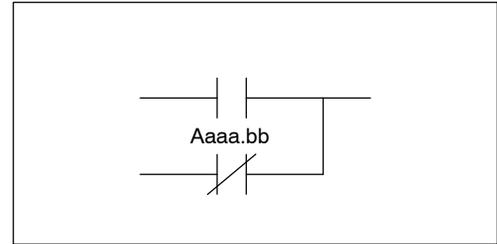
Aaaa.bb
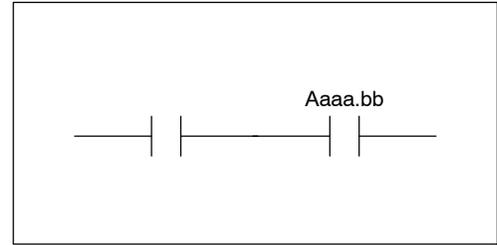
## And Not Bit-of-Word (ANDNB)

| X | X | ✔ |
|---|---|---|
| **430** | **440** | **450** |

| IMP | ✔ |
|---|---|
| **DS** | **HPP** |

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.
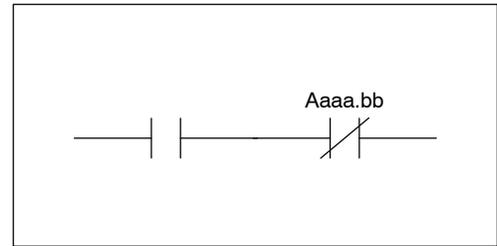
Aaaa.bb

| Operand Data Type | | DL450 Range | |
|---|---|---|---|
| | **A** | **aaa** | **bb** |
| V-memory | B | All (See p. 3-42) | BCD, 0 to 15 |
| Pointer | PB | All (See p. 3-42) | BCD |

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

***DirectSOFT***

```
    X1      B1400.4           Y5
 ---| |------| |-------------( OUT )
```
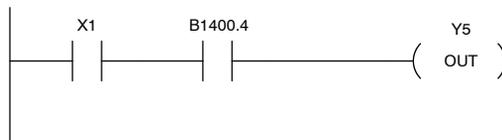
Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ← |
|---|---|---|---|

| AND | SHFT | B | SHFT | V | 1 | 4 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|

| K(con) | 4 | ← |
|---|---|---|

| OUT | Y(OUT) | 5 | ← |
|---|---|---|---|

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.

***DirectSOFT***

```
    X1      B1400.4           Y5
 ---| |------|/|-------------( OUT )
```
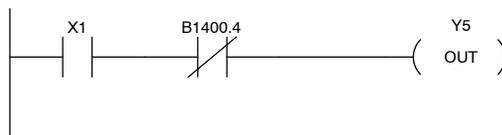
Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ← |
|---|---|---|---|

| AND | SHFT | N | B | SHFT | V | 1 | 4 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|

| K(con) | 4 | ← |
|---|---|---|

| OUT | Y(OUT) | 5 | ← |
|---|---|---|---|

Standard
RLL Instructions

**And Store
(ANDSTR)**

| ✓ | ✓ | ✓ |
|---|---|---|
| 430 | 440 | 450 |

| IMP | ✓ |
|-----|---|
| DS | HPP |

The And Store instruction logically ands two branches of a rung in series. Both branches must begin with the Store instruction.
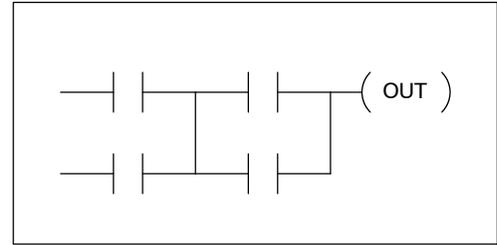
**Or Store
(ORSTR)**

| ✓ | ✓ | ✓ |
|---|---|---|
| 430 | 440 | 450 |

| IMP | ✓ |
|-----|---|
| DS | HPP |

The Or Store instruction logically ors two branches of a rung in parallel. Both branches must begin with the Store instruction.
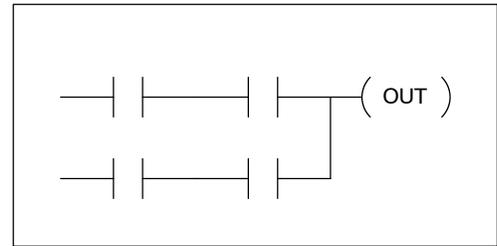
In the following And Store example, the branch consisting of contacts X2, X3, and X4 have been ANDed with the branch consisting of contact X1.

*DirectSOFT*
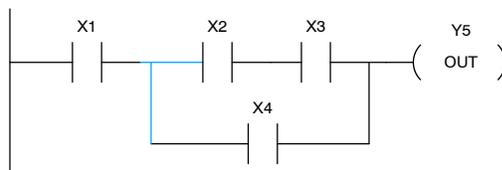
Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
|-----|-------|---|---|
| STR | X(IN) | 2 | ↵ |
| AND | X(IN) | 3 | ↵ |
| OR | X(IN) | 4 | ↵ |
| AND | STR | ↵ | |
| OUT | Y(OUT) | 5 | ↵ |

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

*DirectSOFT*
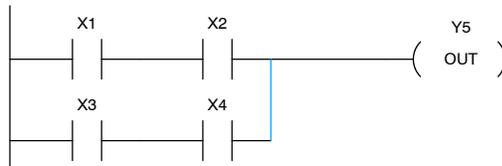
Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
|-----|-------|---|---|
| AND | X(IN) | 2 | ↵ |
| STR | X(IN) | 3 | ↵ |
| AND | X(IN) | 4 | ↵ |
| OR | STR | ↵ | |
| OUT | Y(OUT) | 5 | ↵ |

## Out
## (OUT)

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |
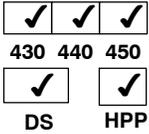
The Out instruction reflects the status of the rung (on/off) and outputs the discrete (on/off) state to the specified image register point or memory location. Multiple Out instructions referencing the same discrete location should not be used since only the last Out instruction in the program will control the physical output point.

```
         Aaaa
    —( OUT )
```
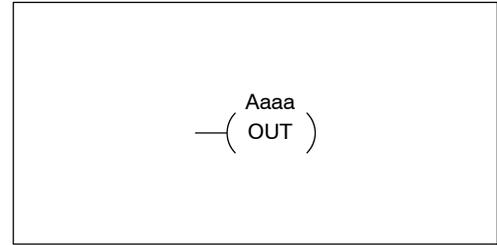
| Operand Data Type | | DL430 Range | DL440 Range | DL450 Range |
|---|---|---|---|---|
| | **A** | **aaa** | **bb** | **bb** |
| Inputs | X | 0–477 | 0–477 | 0–1777 |
| Outputs | Y | 0–477 | 0–477 | 0–1777 |
| Control Relays | C | 0–737 | 0–1777 | 0–3777 |
| Global I/O | GX | 0–777 | 0–1777 | 0–2777 (GX + GY) |

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

*DirectSOFT*

```
   X1                    Y2
  —| |—————————————( OUT )
                       Y5
                    ( OUT )
```

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
|-----|-------|---|---|
| OUT | Y(OUT) | 2 | ↵ |
| OUT | Y(OUT) | 5 | ↵ |

In the following Out example the program contains two Out instructions using the same location (Y10). The physical output of Y10 is ultimately controlled by the last rung of logic referencing Y10. X1 will override the Y10 output being controlled by X0. To avoid this situation, multiple outputs using the same location should not be used in programming.

```
   X0                   Y10
  —| |—————————————( OUT )
                   •
                   •
                   •
                   •
   X1                   Y10
  —| |—————————————( OUT )
```
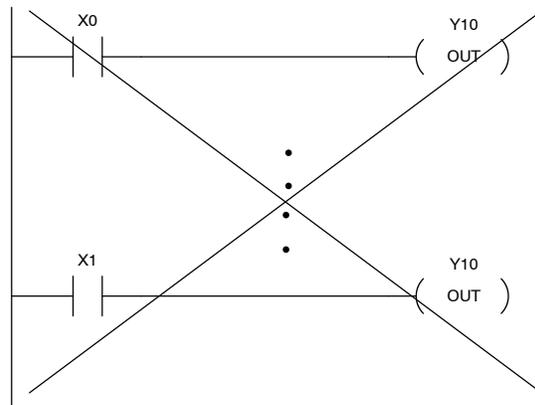
**Out Bit-of-Word (OUTB)**

| 430 | 440 | 450 |
|:---:|:---:|:---:|
| ✕ | ✕ | ✓ |

| DS | HPP |
|:---:|:---:|
| ✓ | ✓ |

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.
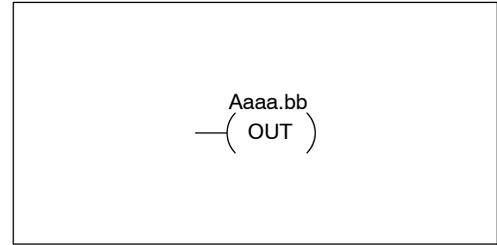
```
          Aaaa.bb
       ─( OUT )
```

| Operand Data Type | | DL450 Range | |
|---|---|---|---|
| | **A** | **aaa** | **bb** |
| V–memory | B | All (See p. 3–42) | BCD, 0 to 15 |
| Pointer | PB | All (See p. 3–42) | BCD |

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.

*DirectSOFT*

```
   X1                                        B1400.3
 ──┤ ├──────────────────────────────────────( OUT )
                                             B1401.6
                                           ──( OUT )
```

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |

| OUT | SHFT | B | SHFT | V | 1 | 4 | 0 | 0 | K(con) | 3 | ↵ |

| OUT | SHFT | B | SHFT | V | 1 | 4 | 0 | 1 | K(con) | 6 | ↵ |
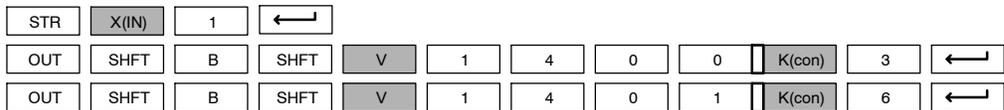
The following Out Bit-of-Word example contains two Out Bit-of-Word instructions using the same bit in the same memory word. The final state bit 3 of V1400 is ultimately controlled by the last rung of logic referencing it. X1 will override the logic state controlled by X0. To avoid this situation, multiple outputs using the same location must not be used in programming.

```
   X0                        V1400    K3
 ──┤ ├──────────────────────( OUT )

                       •
                       •
                       •
                       •

   X1                        V1400    K3
 ──┤ ├──────────────────────( OUT )
```

## Or Out
## (OROUT)

| ✓ | ✓ | ✓ |
|---|---|---|

**430  440  450**

| ✓ | ✓ |
|---|---|

**DS       HPP**
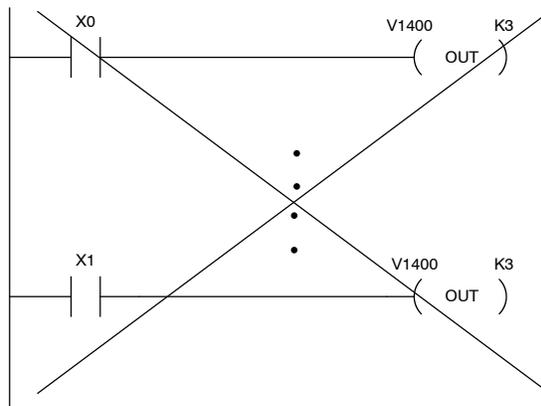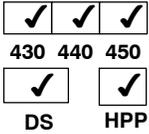
The Or Out instruction has been designed to use more than 1 rung of discrete logic to control a single output. Multiple Or Out instructions referencing the same output coil may be used, since *all* contacts controlling the output are ORed together. If the status of *any* rung is on, the output will also be on.

A aaa
─(OROUT)

| Operand Data Type | | DL430 Range | DL440 Range | DL450 Range |
|---|---|---|---|---|
| | **A** | **aaa** | **aaa** | **aaa** |
| Inputs | X | 0–477 | 0–477 | 0–1777 |
| Outputs | Y | 0–477 | 0–477 | 0–1777 |
| Control Relays | C | 0–737 | 0–1777 | 0–3777 |
| Global I/O | GX | 0–777 | 0–1777 | 0–2777 (GX + GY) |

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

*DirectSOFT*

```
        X1                              Y2
    ──┤ ├──                        ( OR OUT )


        X4                              Y2
    ──┤ ├──                        ( OR OUT )
```

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
|-----|-------|---|---|
| OR | OUT | Y(OUT) | 2 | ↵ |
| STR | X(IN) | 4 | ↵ |
| OR | OUT | Y(OUT) | 2 | ↵ |

## Not
## (NOT)

| ✓ | ✓ | ✓ |
|---|---|---|

**430  440  450**

| ✓ | ✓ |
|---|---|

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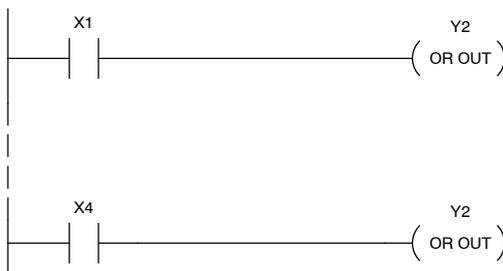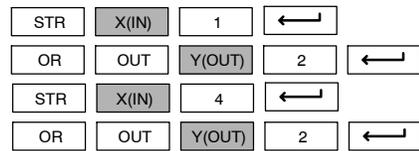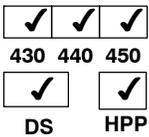
The Not instruction inverts the status of the rung at the point of the instruction.
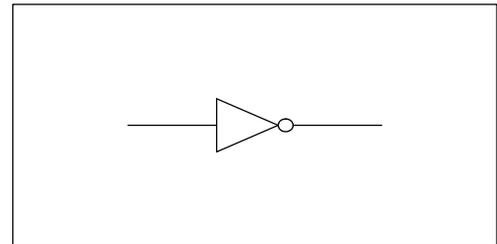
In the following example when X1 is off, Y2 will energize. This is because the Not instruction inverts the status of the rung at the Not instruction.

*DirectSOFT*

```
        X1                              Y2
    ──┤ ├──    ─▷o─              ( OUT )
```

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
|-----|-------|---|---|
| SHFT | N | O | T | ↵ |
| OUT | Y(OUT) | 2 | ↵ |
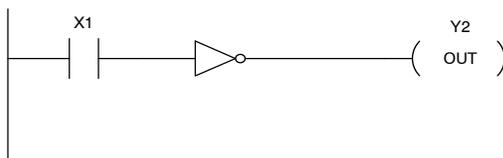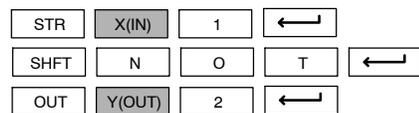
**NOTE:** *Direct*SOFT Release 1.1i and later supports the use of the NOT instruction. The above example rung is merely intended to show the visual representation of the NOT instruction. The rung cannot be created or displayed in *Direct*SOFT versions earlier than 1.1i.

Standard
RLL Instructions

**Pause
(PAUSE)**

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |
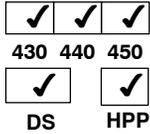
The Pause instruction disables the output update on a range of outputs. The ladder program will continue to run and update the image register however the outputs in the range specified in the Pause instruction will be turned off at the output module.

```
        Y aaa   aaa
      —(PAUSE )
```
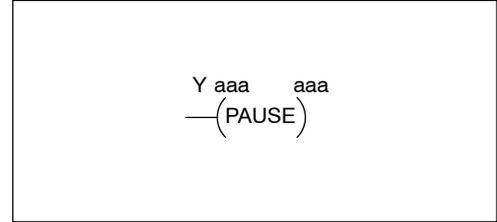
| Operand Data Type | | DL430 Range | DL440 Range | DL450 Range |
|---|---|---|---|---|
| | **A** | **aaa** | **aaa** | **aaa** |
| Outputs | Y | 0–477 | 0–477 | 0–1777 |

In the following example, when X1 is ON, Y10–Y17 will be turned OFF at the output module. The execution of the ladder program will not be affected.

*DirectSOFT*

```
    X1              Y10   Y17
  —| |——| |————( PAUSE )
```

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
| SHFT | P | A | U | S | E |
| SHFT | Y(OUT) | 1 | 0 |
| Y(OUT) | 1 | 7 | ↵ |
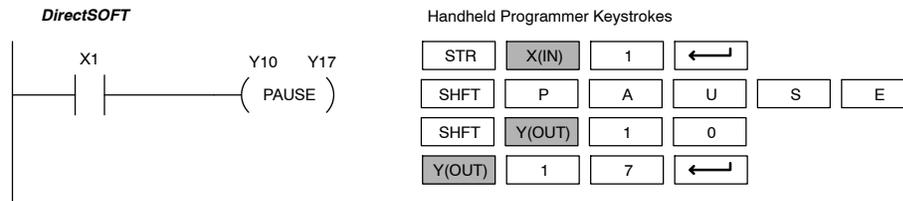
**Positive
Differential
(PD)**

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |
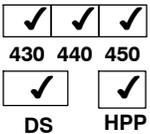
The Positive Differential instruction is typically known as a one-shot. When the input logic produces an Off-to-On transition, the output will energize for one CPU scan. Thereafter, it remains off until its input makes another Off-to-On transition.
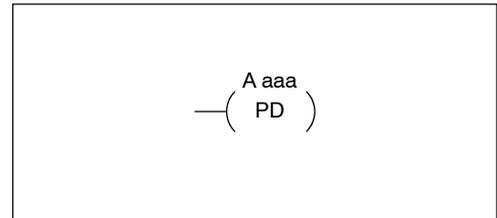
```
        A aaa
      —(  PD  )
```

| Operand Data Type | | DL430 Range | DL440 Range | DL450 Range |
|---|---|---|---|---|
| | **A** | **aaa** | **aaa** | **aaa** |
| Inputs | X | 0–477 | 0–477 | 0–1777 |
| Outputs | Y | 0–477 | 0–477 | 0–1777 |
| Control Relays | C | 0–737 | 0–1777 | 0–3777 |

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

*DirectSOFT*

```
    X1                  C0
  —| |——| |————(  PD  )
```

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
| SHFT | P | D | SHFT | C(CR) | 0 | ↵ |
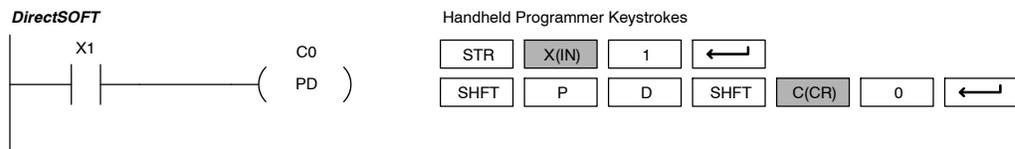
Note that you can place a "NOT" instruction immediately before the PD instruction to generate a "one-shot" pulse on an on-to-off transition.

**Store Positive Differential (STRPD)**

| × | × | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

The Store Positive Differential instruction begins a new rung or an additional branch in a rung with a normally open 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 retentive range.

Aaaa
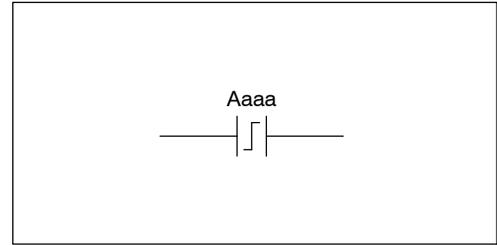
**Store Negative Differential (STRND)**

| × | × | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

The Store Negative Differential instruction begins a new rung or an additional branch in a rung with a normally closed 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).

Aaaa

| Operand Data Type | | DL450 Range |
|---|---|---|
| | **A** | **aaa** |
| Inputs | X | 0–1777 |
| Outputs | Y | 0–1777 |
| Control Relays | C | 0–3777 |
| Stage | S | 0–1777 |
| Timer | T | 0–377 |
| Counter | CT | 0–377 |
| Global | GX | 0–2777 (GX + GY) |

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

*DirectSOFT*

X1        Y4
( OUT )

Handheld Programmer Keystrokes

| STR | SHFT | P | D | SHFT |
|---|---|---|---|---|
| X(IN) | 1 | ↵ | | |
| OUT | Y(OUT) | 4 | ↵ | |
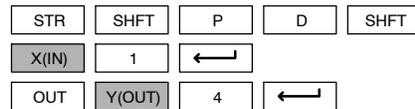
In the following example, each time X1 is makes an On-to-Off transition, Y4 will energize for one scan.

*DirectSOFT*

X1        Y4
( OUT )

Handheld Programmer Keystrokes

| STR | SHFT | N | D | SHFT |
|---|---|---|---|---|
| X(IN) | 1 | ↵ | | |
| OUT | Y(OUT) | 4 | ↵ | |

Standard
RLL Instructions

**Or Positive Differential (ORPD)**

| ✕ | ✕ | ✔ |
|---|---|---|
| **430** | **440** | **450** |

| IMP | ✔ |
|-----|---|
| **DS** | **HPP** |

The Or Positive Differential instruction logically ors a normally open 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.
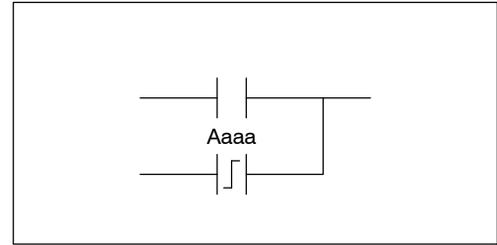


**Or Negative Differential (ORND)**

| ✕ | ✕ | ✔ |
|---|---|---|
| **430** | **440** | **450** |

| IMP | ✔ |
|-----|---|
| **DS** | **HPP** |

The Or Negative Differential instruction logically ors a normally open 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 | | DL450 Range |
|---|---|---|
| | **A** | **aaa** |
| Inputs | X | 0–1777 |
| Outputs | Y | 0–1777 |
| Control Relays | C | 0–3777 |
| Stage | S | 0–1777 |
| Timer | T | 0–377 |
| Counter | CT | 0–377 |
| Global | GX | 0–2777 (GX + GY) |

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

| STR | X(IN) | 1 | ↵ | |
|-----|-------|---|---|---|
| OR | SHFT | P | D | SHFT |
| X(IN) | 2 | ↵ | | |
| OUT | Y(OUT) | 5 | ↵ | |

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

| STR | X(IN) | 1 | ↵ | |
|-----|-------|---|---|---|
| OR | SHFT | N | D | SHFT |
| X(IN) | 2 | ↵ | | |
| OUT | Y(OUT) | 5 | ↵ | |

## And Positive Differential (ANDPD)

| ✕ | ✕ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| IMP | ✓ |
|---|---|
| **DS** | **HPP** |

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.
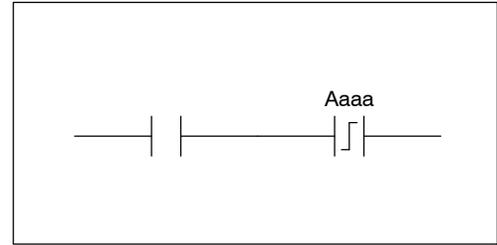
## And Negative Differential (ANDND)

| ✕ | ✕ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| IMP | ✓ |
|---|---|
| **DS** | **HPP** |

The And Negative 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 On-to-Off transition, closing it for one CPU scan. Thereafter, it remains open until another On-to-Off transition.

| Operand Data Type | | DL450 Range |
|---|---|---|
| | **A** | **aaa** |
| Inputs | X | 0–1777 |
| Outputs | Y | 0–1777 |
| Control Relays | C | 0–3777 |
| Stage | S | 0–1777 |
| Timer | T | 0–377 |
| Counter | CT | 0–377 |
| Global | GX | 0–2777 (GX + GY) |

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

*DirectSOFT*

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.

*DirectSOFT*

Handheld Programmer Keystrokes

**Set
(SET)**

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

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.

Optional memory range

A aaa      aaa
—( SET )

**Reset
(RST)**

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

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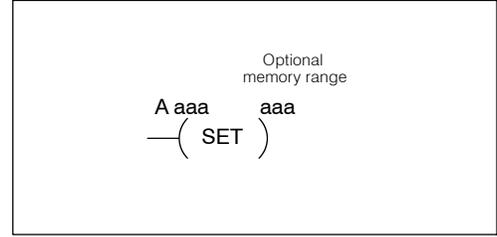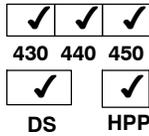
The Reset instruction resets or turns off an image register point/memory location or a range of image registers points/memory locations. Once the point/location is reset it is not necessary for the input to remain on.
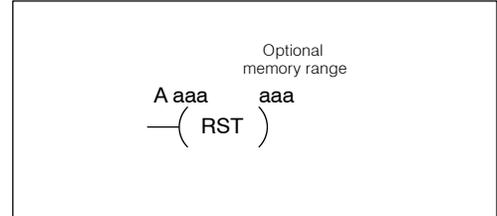
Optional memory range

A aaa      aaa
—( RST )

| Operand Data Type | | DL430 Range | DL440 Range | DL450 Range |
|---|---|---|---|---|
| | **A** | **aaa** | **aaa** | **aaa** |
| Inputs | X | 0–477 | 0–477 | 0–1777 |
| Outputs | Y | 0–477 | 0–477 | 0–1777 |
| Control Relays | C | 0–737 | 0–1777 | 0–3777 |
| Stages | S | 0–577 | 0–1777 | 0–1777 |
| Timers* | T | 0–177 | 0–377 | 0–377 |
| Counters* | CT | 0–177 | 0–177 | 0–377 |
| Global | GX | 0–777 | 0–1777 | 0–2777 (GX + GY) |

* Timer and counter operand data types are not valid using the Set instruction.

In the following example when X1 turns on, Y5 through Y22 will be set to the on state.

*DirectSOFT*

X1                                    Y5      Y22
—| |—————————————————————————( SET )

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
|---|---|---|---|

| SET | Y(OUT) | 5 | Y(OUT) | 2 | 2 | ↵ |
|---|---|---|---|---|---|---|
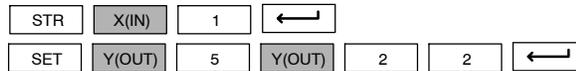
In the following example when X1 turns on, Y5 through Y22 will be reset to the off state.

*DirectSOFT*

X1                                    Y5      Y22
—| |—————————————————————————( RST )

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
|---|---|---|---|

| RST | Y(OUT) | 5 | Y(OUT) | 2 | 2 | ↵ |
|---|---|---|---|---|---|---|
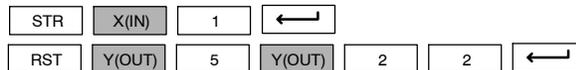
**Set Bit-of-Word (SETB)**

| × | × | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

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.

```
                    Aaaa.bb
        ——(  SET  )
```
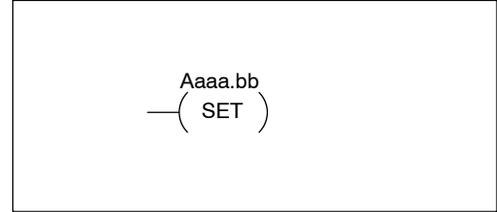
**Reset Bit-of-Word (RSTB)**

| × | × | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

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.

```
                    A aaa.bb
        ——(  RST  )
```
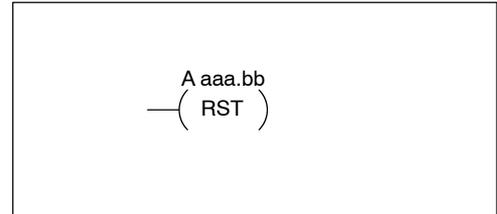
| Operand Data Type | | DL450 Range | |
|---|---|---|---|
| **A** | | **aaa** | **bb** |
| V-memory | B | All (See p. 3-42) | 0 to 15 |
| Pointer | PB | All (See p. 3-42) | 0 to 15 |

In the following example when X1 turns on, bit 0 in V1400 is set to the on state.

*DirectSOFT*

```
        X1                                        B1400.0
        | |                                   ——(  SET  )
```

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ←— |
|-----|-------|---|-----|

| SET | SHFT | B | SHFT | V | 1 | 4 | 0 | 0 |
|-----|------|---|------|---|---|---|---|---|

| K(con) | 0 | ←— |
|--------|---|-----|

In the following example when X1 turns on, bit 15 in V1400 is reset to the off state.

*DirectSOFT*

```
        X1                                        V1400.15
        | |                                   ——(  RST  )
```
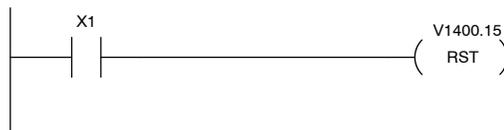
Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ←— |
|-----|-------|---|-----|

| RST | SHFT | B | SHFT | V | 1 | 4 | 0 | 0 |
|-----|------|---|------|---|---|---|---|---|

| K(con) | 1 | 5 | ←— |
|--------|---|---|-----|

# Comparative Boolean Instructions

**Store If Equal (STRE)**

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

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 Aaaa = Bbbb.
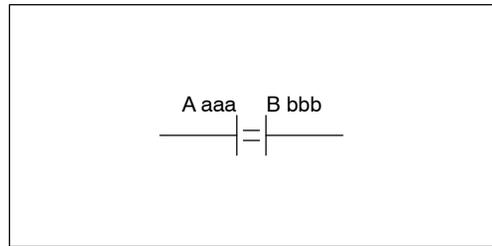
A aaa     B bbb
———| |———

**Store If Not Equal (STRNE)**

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

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 Aaaa ≠ Bbbb.

A aaa     B bbb
———|/|———

| Operand Data Type | | DL430 Range | | DL440 Range | | DL450 Range | |
|---|---|---|---|---|---|---|---|
| | **A/B** | **aaa** | **bbb** | **aaa** | **bbb** | **aaa** | **bbb** |
| V-memory | V | All (See p. 3–40) | All (See p. 3–40) | All (See p. 3–41) | All (See p. 3–41) | All (See p. 3–42) | All (See p. 3–42) |
| Pointer | P | -- | -- | All (See p. 3–41) | All (See p. 3–41) | All (See p. 3–42) | All (See p. 3–42) |
| Constant | K | -- | 0–FFFF | -- | 0–FFFF | -- | 0–FFFF |

In the following example, when the value in V–memory location V1400 = 4933 , Y3 will energize.

*DirectSOFT*

```
V1400   K4933                          Y3
——| = |————————————————————(  OUT  )
```

Handheld Programmer Keystrokes

| STR | SHFT | E | SHFT | V | 1 | 4 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|

| K(CON) | 4 | 9 | 3 | 3 | ↵ |
|---|---|---|---|---|---|

| OUT | Y(OUT) | 3 | ↵ |
|---|---|---|---|

In the following example, when the value in V–memory location V1400 ≠ 5060, Y3 will energize.

*DirectSOFT*

```
V1400   K5060                          Y3
——|/|————————————————————————(  OUT  )
```

Handheld Programmer Keystrokes

| STR | NOT | SHFT | E | SHFT | V | 1 | 4 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|

| K(CON) | 5 | 0 | 6 | 0 | ↵ |
|---|---|---|---|---|---|

| OUT | Y(OUT) | 3 | ↵ |
|---|---|---|---|

## Or If Equal (ORE)

✓ ✓ ✓
**430 440 450**

IMP | ✓
**DS** | **HPP**

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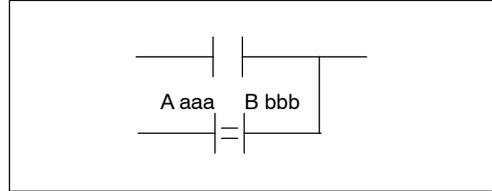


A aaa    B bbb

## Or If Not Equal (ORNE)

✓ ✓ ✓
**430 440 450**

IMP | ✓
**DS** | **HPP**

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



A aaa    B bbb

| Operand Data Type | | DL430 Range | | DL440 Range | | DL450 Range | |
|---|---|---|---|---|---|---|---|
| | A/B | aaa | bbb | aaa | bbb | aaa | bbb |
| V-memory | V | All (See p. 3-40) | All (See p. 3-40) | All (See p. 3-41) | All (See p. 3-41) | All (See p. 3-42) | All (See p. 3-42) |
| Pointer | P | -- | -- | All (See p. 3-41) | All (See p. 3-41) | All (See p. 3-42) | All (See p. 3-42) |
| Constant | K | -- | 0-FFFF | -- | 0-FFFF | -- | 0-FFFF |

In the following example, when the value in V-memory location V1400 = 4500 or V1402 = 2345 , Y3 will energize.
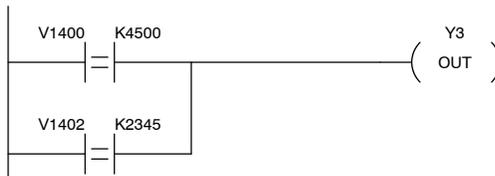
*DirectSOFT*



Handheld Programmer Keystrokes

| STR | SHFT | E | SHFT | V | 1 | 4 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|

| K(CON) | 4 | 5 | 0 | 0 | ← |
|---|---|---|---|---|---|

| OR | SHFT | E | SHFT | V | 1 | 4 | 0 | 2 |
|---|---|---|---|---|---|---|---|---|

| K(CON) | 2 | 3 | 4 | 5 | ← |
|---|---|---|---|---|---|

| OUT | Y(OUT) | 3 | ← |
|---|---|---|---|

In the following example, when the value in V-memory location V1400 = 3916 or V1402 ≠ 2500, Y3 will energize.

*DirectSOFT*
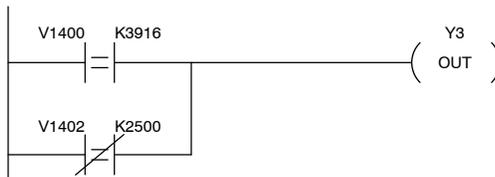


Handheld Programmer Keystrokes

| STR | SHFT | E | SHFT | V | 1 | 4 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|

| K(CON) | 3 | 9 | 1 | 6 | ← |
|---|---|---|---|---|---|

| OR | NOT | SHFT | E | SHFT | V | 1 | 4 | 0 | 2 |
|---|---|---|---|---|---|---|---|---|---|

| K(CON) | 2 | 5 | 0 | 0 | ← |
|---|---|---|---|---|---|

| OUT | Y(OUT) | 3 | ← |
|---|---|---|---|

Standard
RLL Instructions

**And If Equal (ANDE)**

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| IMP | ✓ |
|-----|---|
| **DS** | **HPP** |

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

```
            A aaa     B bbb
    ───────┤ ├──────┤=├──────
```
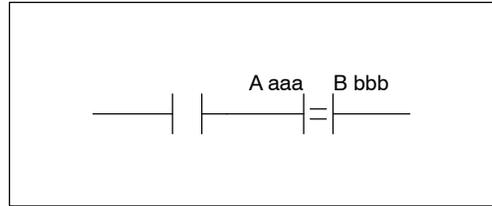
**And If Not Equal (ANDNE)**

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| IMP | ✓ |
|-----|---|
| **DS** | **HPP** |

The And If Not Equal instruction connects a normally closed comparative contact in series with another contact. The contact will be on when Aaaa $\neq$ Bbbb

```
            A aaa     B bbb
    ───────┤ ├──────┤/=├──────
```

| Operand Data Type | | DL430 Range | | DL440 Range | | DL450 Range | |
|---|---|---|---|---|---|---|---|
| | A/B | aaa | bbb | aaa | bbb | aaa | bbb |
| V-memory | V | All (See p. 3–40) | All (See p. 3–40) | All (See p. 3–41) | All (See p. 3–41) | All (See p. 3–42) | All (See p. 3–42) |
| Pointer | P | -- | -- | All (See p. 3–41) | All (See p. 3–41) | All (See p. 3–42) | All (See p. 3–42) |
| Constant | K | -- | 0–FFFF | -- | 0–FFFF | -- | 0–FFFF |

In the following example, when the value in V–memory location V1400 = 5000 and V1402 = 2345, Y3 will energize.

*DirectSOFT*

```
     V1400   K5000      V1402   K2345        Y3
    ───┤=├──────────────┤=├──────────(  OUT  )
```

Handheld Programmer Keystrokes

| STR | SHFT | E | SHFT | V | 1 | 4 | 0 | 0 |
|-----|------|---|------|---|---|---|---|---|

| K(CON) | 5 | 0 | 0 | 0 | ← |
|--------|---|---|---|---|---|

| AND | SHFT | E | SHFT | V | 1 | 4 | 0 | 2 |
|-----|------|---|------|---|---|---|---|---|

| K(CON) | 2 | 3 | 4 | 5 | ← |
|--------|---|---|---|---|---|

| OUT | Y(OUT) | 3 | ← |
|-----|--------|---|---|

In the following example, when the value in V–memory location V1400 = 2550 and V1402 $\neq$ 2500, Y3 will energize.

*DirectSOFT*

```
     V1400   K2550      V1402   K2500        Y3
    ───┤=├──────────────┤/=├──────────(  OUT  )
```
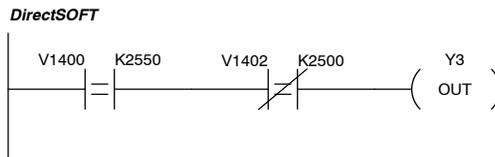
Handheld Programmer Keystrokes

| STR | SHFT | E | SHFT | V | 1 | 4 | 0 | 0 |
|-----|------|---|------|---|---|---|---|---|

| K(CON) | 2 | 5 | 5 | 0 | ← |
|--------|---|---|---|---|---|

| AND | NOT | SHFT | E | SHFT | V | 1 | 4 | 0 | 2 |
|-----|-----|------|---|------|---|---|---|---|---|

| K(CON) | 2 | 5 | 0 | 0 | ← |
|--------|---|---|---|---|---|

| OUT | Y(OUT) | 3 | ← |
|-----|--------|---|---|

Standard RLL Instructions

**Store
(STR)**

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

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 $\geq$ Bbbb.

```
     A aaa    B bbb
  ─────┤≥├─────
```

**Store Not
(STRN)**

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

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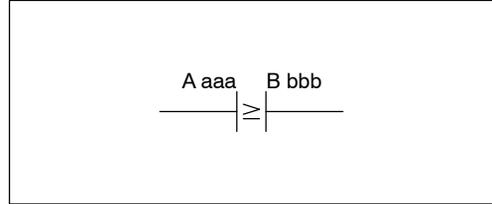
The Comparative Store Not instruction begins a new rung or additional branch in a rung with a normally closed comparative contact. The contact will be on when Aaaa < Bbbb.

```
     A aaa    B bbb
  ─────┤<├─────
```

| Operand Data Type | | DL430 Range | | DL440 Range | | DL450 Range | |
|---|---|---|---|---|---|---|---|
| | **A/B** | **aaa** | **bbb** | **aaa** | **bbb** | **aaa** | **bbb** |
| V-memory | V | All (See p. 3-40) | All (See p. 3-40) | All (See p. 3-41) | All (See p. 3-41) | All (See p. 3-42) | All (See p. 3-42) |
| Pointer | P | -- | -- | All (See p. 3-41) | All (See p. 3-41) | All (See p. 3-42) | All (See p. 3-42) |
| Constant | K | -- | 0-FFFF | -- | 0-FFFF | -- | 0-FFFF |
| Timer | T | 0-177 | | 0-377 | | 0-377 | |
| Counter | CT | 0-177 | | 0-177 | | 0-377 | |

In the following example, when the value in V-memory location V1400 $\geq$ 1000, Y3 will energize.

*DirectSOFT*

```
   V1400   K1000                          Y3
  ───┤≥├──────────────────────────(  OUT  )
```

Handheld Programmer Keystrokes

| STR | V | 1 | 4 | 0 | 0 |
|---|---|---|---|---|---|

| K(CON) | 1 | 0 | 0 | 0 | ← |
|---|---|---|---|---|---|

| OUT | Y(OUT) | 3 | ← |
|---|---|---|---|

In the following example, when the value in V-memory location V1400 < 4050, Y3 will energize.

*DirectSOFT*

```
   V1400   K4050                          Y3
  ───┤<├───────────────────────────(  OUT  )
```
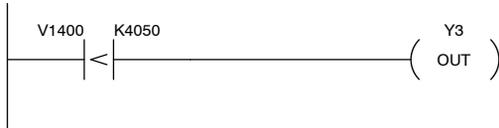
Handheld Programmer Keystrokes

| STR | NOT | V | 1 | 4 | 0 | 0 |
|---|---|---|---|---|---|---|

| K(CON) | 4 | 0 | 5 | 0 | ← |
|---|---|---|---|---|---|

| OUT | Y(OUT) | 3 | ← |
|---|---|---|---|

Standard
RLL Instructions

**Or
(OR)**

| ✔ | ✔ | ✔ |
|---|---|---|
| **430** | **440** | **450** |

| IMP | ✔ |
|---|---|
| **DS** | **HPP** |

The Comparative Or instruction connects a normally open comparative contact in parallel with another contact. The contact will be on when Aaaa ≥ Bbbb.



**Or Not
(ORN)**

| ✔ | ✔ | ✔ |
|---|---|---|
| **430** | **440** | **450** |

| IMP | ✔ |
|---|---|
| **DS** | **HPP** |
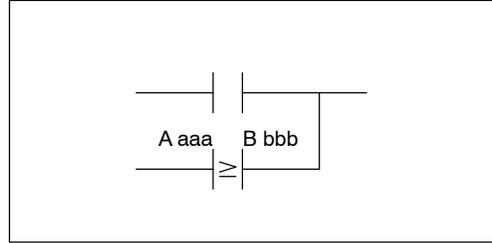
The Comparative Or Not instruction connects a normally open comparative contact in parallel with another contact. The contact will be on when Aaaa < Bbbb.



| Operand Data Type | | DL430 Range | | DL440 Range | | DL450 Range | |
|---|---|---|---|---|---|---|---|
| | A/B | **aaa** | **bbb** | **aaa** | **bbb** | **aaa** | **bbb** |
| V-memory | V | All (See p. 3–40) | All (See p. 3–40) | All (See p. 3–41) | All (See p. 3–41) | All (See p. 3–42) | All (See p. 3–42) |
| Pointer | P | -- | -- | All (See p. 3–41) | All (See p. 3–41) | All (See p. 3–42) | All (See p. 3–42) |
| Constant | K | -- | 0–FFFF | -- | 0–FFFF | -- | 0–FFFF |
| Timer | T | 0–177 | | 0–377 | | 0–377 | |
| Counter | CT | 0–177 | | 0–177 | | 0–377 | |

In the following example, when the value in V–memory location V1400 = 6045 or V1402 ≥ 2345, Y3 will energize.
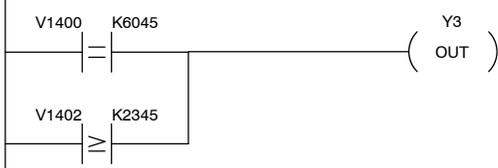
*DirectSOFT*



Handheld Programmer Keystrokes

| STR | SHFT | E | SHFT | V | 1 | 4 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|
| K(CON) | 6 | 0 | 4 | 5 | ↵ | | | |
| OR | V | 1 | 4 | 0 | 2 | | | |
| K(CON) | 2 | 3 | 4 | 5 | ↵ | | | |
| OUT | Y(OUT) | 3 | ↵ | | | | | |

In the following example when the value in V–memory location V1400 = 1000 or V1402 < 2500, Y3 will energize.

*DirectSOFT*
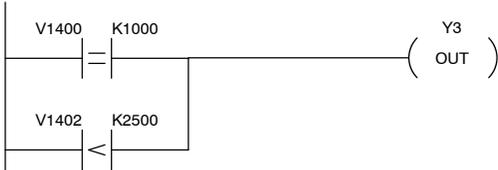


Handheld Programmer Keystrokes

| STR | SHFT | E | SHFT | V | 1 | 4 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|
| K(CON) | 1 | 0 | 0 | 0 | ↵ | | | |
| OR | NOT | V | 1 | 4 | 0 | 2 | | |
| K(CON) | 2 | 5 | 0 | 0 | ↵ | | | |
| OUT | Y(OUT) | 3 | ↵ | | | | | |

Standard
RLL Instructions

**And
(AND)**

✓ ✓ ✓
**430 440 450**

| IMP | ✓ |
|---|---|
| **DS** | **HPP** |

The Comparative And instruction connects a normally open comparative contact in series with another contact. The contact will be on when Aaaa ≥ Bbbb.



**And Not
(ANDN)**

✓ ✓ ✓
**430 440 450**

| IMP | ✓ |
|---|---|
| **DS** | **HPP** |
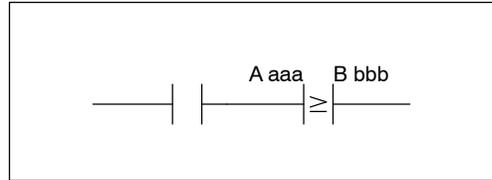
The Comparative And Not instruction connects a normally open comparative contact in series with another contact. The contact will be on when Aaaa < Bbbb.



| Operand Data Type | | DL430 Range | | DL440 Range | | DL450 Range | |
|---|---|---|---|---|---|---|---|
| | A/B | aaa | bbb | aaa | bbb | aaa | bbb |
| V-memory | V | All (See p. 3–40) | All (See p. 3–40) | All (See p. 3–41) | All (See p. 3–41) | All (See p. 3–42) | All (See p. 3–42) |
| Pointer | P | -- | -- | All (See p. 3–41) | All (See p. 3–41) | All (See p. 3–42) | All (See p. 3–42) |
| Constant | K | -- | 0-FFFF | -- | 0-FFFF | -- | 0-FFFF |
| Timer | T | 0-177 | | 0-377 | | 0-377 | |
| Counter | CT | 0-177 | | 0-177 | | 0-377 | |

In the following example, when the value in V-memory location V1400 = 5000, and V1402 ≥ 2345, Y3 will energize.



In the following example, when the value in V-memory location V1400 = 7000 and V1402 < 2500, Y3 will energize.

# Immediate Instructions

**Store Immediate (STRI)**

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

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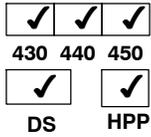
The Store Immediate instruction begins a new rung or additional branch in a rung. The status of the contact will be the same as the status of the associated input point on the module *at the time the instruction is executed*. The image register is not updated.

X aaa
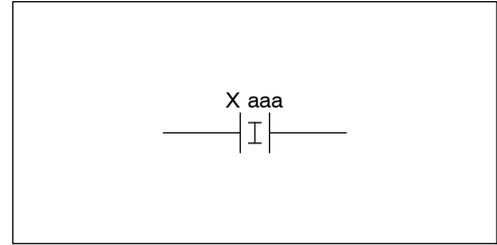
**Store Not Immediate (STRNI)**

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

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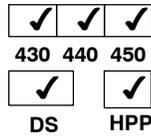
The Store Not Immediate instruction begins a new rung or additional branch in a rung. The status of the contact will be opposite the status of the associated input point on the module *at the time the instruction is executed*. The image register is not updated.

X aaa
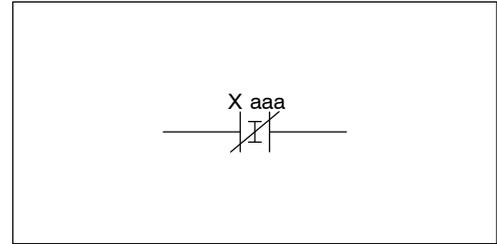
| Operand Data Type | | DL430 Range | DL440 Range | DL450 Range |
|---|---|---|---|---|
| | | aaa | aaa | aaa |
| Inputs | X | 0–477 | 0–477 | 0–1777 |

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

*DirectSOFT*

```
      X1                              Y2
    --| I |--------------------------( OUT )
```

Handheld Programmer Keystrokes

| STR | SHFT | I | SHFT | X(IN) | 1 | ↵ |
| OUT | Y(OUT) | 2 | ↵ |

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

*DirectSOFT*

```
      X1                              Y2
    --|/I|--------------------------( OUT )
```

Handheld Programmer Keystrokes

| STR | NOT | SHFT | I | SHFT | X(IN) | 1 | ↵ |
| OUT | Y(OUT) | 2 | ↵ |

Standard
RLL Instructions

## Or Immediate (ORI)

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| IMP | ✓ |
|---|---|
| **DS** | **HPP** |

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 on the module *at the time the instruction is executed*. The image register is not updated.
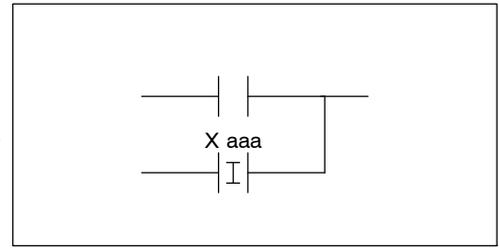
## Or Not Immediate (ORNI)

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| IMP | ✓ |
|---|---|
| **DS** | **HPP** |

The Or Not Immediate connects two contacts in parallel. The status of the contact will be opposite the status of the associated input point on the module *at the time the instruction is executed*. The image register is not updated.
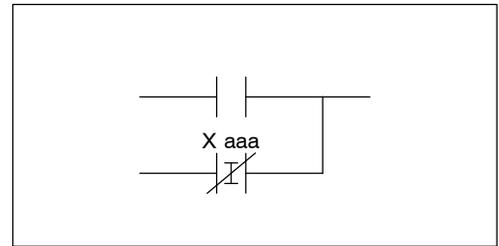
| Operand Data Type | | DL430 Range | DL440 Range | DL450 Range |
|---|---|---|---|---|
| | | **aaa** | **aaa** | **aaa** |
| Inputs | X | 0-477 | 0-477 | 0-1777 |

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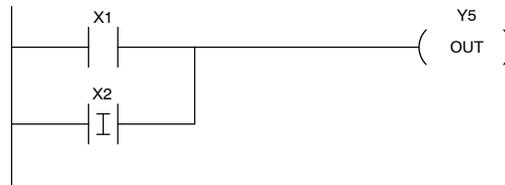
*DirectSOFT*

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ | | | |
|---|---|---|---|---|---|---|
| OR | SHFT | I | SHFT | X(IN) | 2 | ↵ |
| OUT | Y(OUT) | 5 | ↵ | | | |

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

*DirectSOFT*
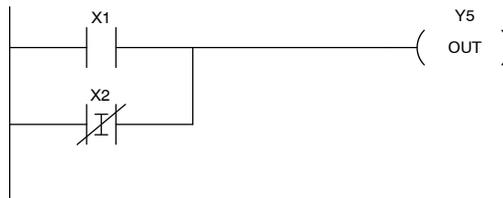
Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ | | | | |
|---|---|---|---|---|---|---|---|
| OR | NOT | SHFT | I | SHFT | X(IN) | 2 | ↵ |
| OUT | Y(OUT) | 5 | ↵ | | | | |

Standard
RLL Instructions

**And Immediate (ANDI)**

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |
| **IMP** | ✓ | |
| **DS** | **HPP** | |

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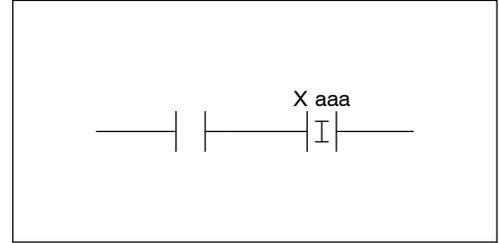
**And Not Immediate (ANDNI)**

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |
| **IMP** | ✓ | |
| **DS** | **HPP** | |

The And Not Immediate connects two contacts in series. The status of the contact will be opposite the status of the associated input point on the module *at the time the instruction is executed*. The image register is not updated.
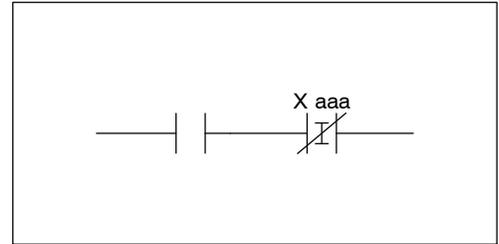
| Operand Data Type | | DL430 Range | DL440 Range | DL450 Range |
|---|---|---|---|---|
| | | aaa | aaa | aaa |
| Inputs | X | 0–477 | 0–477 | 0–1777 |

In the following example, when X1 and X2 is on, Y5 will energize.
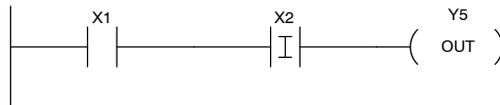
*DirectSOFT*

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
| AND | SHFT | I | SHFT | X(IN) | 2 | ↵ |
| OUT | Y(OUT) | 5 | ↵ |

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

*DirectSOFT*
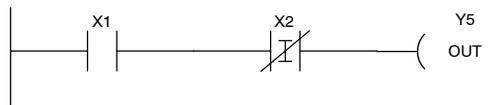
Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
| AND | NOT | SHFT | I | SHFT | X(IN) | 2 | ↵ |
| OUT | Y(OUT) | 5 | ↵ |

## Out Immediate (OUTI)

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |
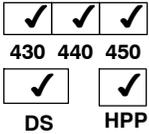
The Out Immediate instruction reflects the status of the rung (on/off) and outputs the discrete (on/off) status to the specified module output point and the image register *at the time the instruction is executed*. If multiple Out Immediate instructions referencing the same discrete point are used it is possible for the module output status to change multiple times in a CPU scan. See Or Out Immediate.

Y aaa
—( OUTI )
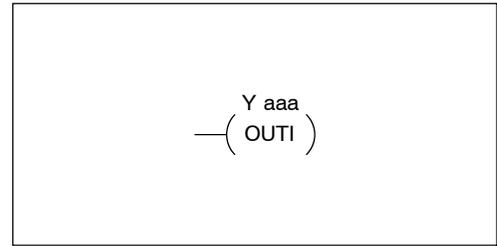
## Or Out Immediate (OROUTI)

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

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 ored together. If the status of *any* rung is on *at the time the instruction is executed*, the output will also be on.
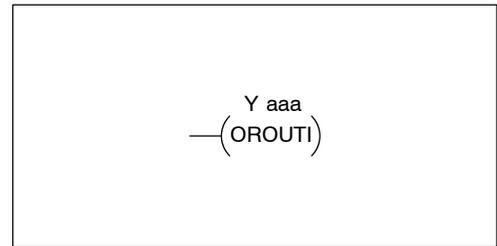
Y aaa
—(OROUTI)

| Operand Data Type | | DL430 Range | DL440 Range | DL450 Range |
|---|---|---|---|---|
| | | aaa | aaa | aaa |
| Outputs | Y | 0-477 | 0-477 | 0-1777 |

In the following example, when X1 is on, output point Y2 on the output module will turn on.

**DirectSOFT**

```
       X1                                    Y2
    ——| |——————————————————————————————————( OUTI )
```

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
|-----|-------|---|---|

| OUT | SHFT | I | SHFT | Y(OUT) | 2 | ↵ |
|-----|------|---|------|--------|---|---|

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

**DirectSOFT**

```
       X1                                    Y2
    ——| |——————————————————————————————————( OR OUTI )
    •
    •
    •
       X4                                    Y2
    ——| |——————————————————————————————————( OR OUTI )
```

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
|-----|-------|---|---|

| OR | OUT | SHFT | I | SHFT | Y(OUT) | 2 | ↵ |
|----|-----|------|---|------|--------|---|---|

| STR | X(IN) | 4 | ↵ |
|-----|-------|---|---|

| OR | OUT | SHFT | I | SHFT | Y(OUT) | 2 | ↵ |
|----|-----|------|---|------|--------|---|---|

Standard
RLL Instructions
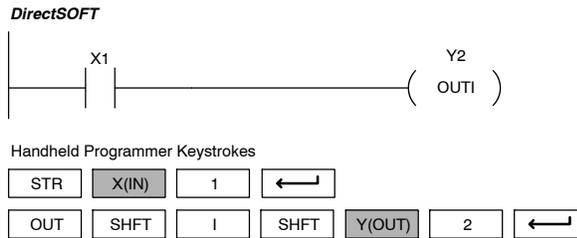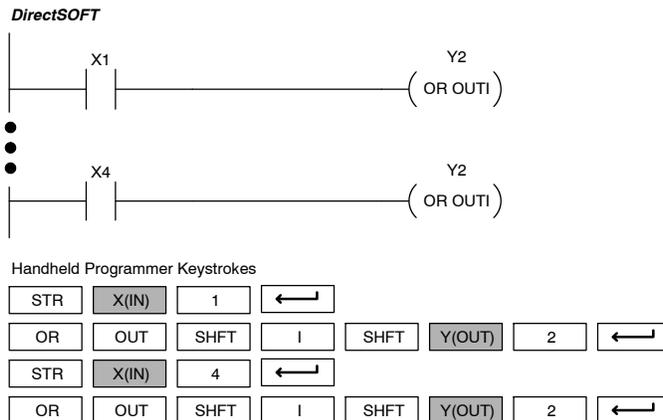
**Set Immediate (SETI)**

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |
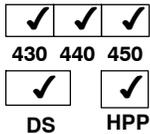
The Set Immediate instruction immediately sets or turns on an output or a range of outputs and the corresponding output module(s) *at the time the instruction is executed*. The image register is not updated. Once the outputs are set it is not necessary for the input to remain on. The Reset Immediate instruction can be used to reset the outputs.

```
          Y aaa      aaa
         —( SETI )
```
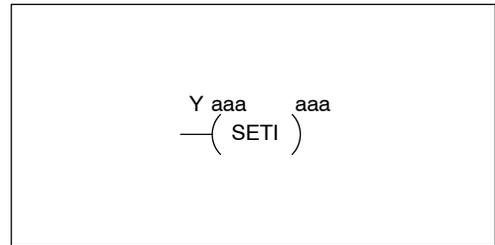
**Reset Immediate (RSTI)**

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |
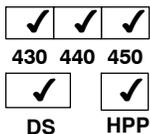
The Reset Immediate instruction immediately resets, or turns off an output or a range of outputs and the output module(s) *at the time the instruction is executed*. The image register is not updated. Once the outputs are reset it is not necessary for the input to remain on.
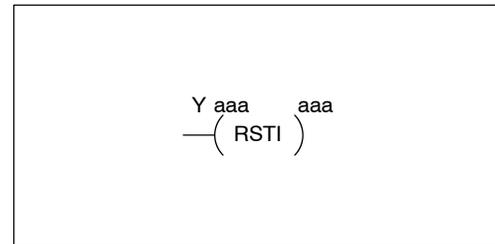
```
          Y aaa      aaa
         —( RSTI )
```

| Operand Data Type | | DL430 Range | DL440 Range | DL450 Range |
|---|---|---|---|---|
| | | aaa | aaa | aaa |
| Outputs | Y | 0–477 | 0–477 | 0–1777 |

In the following example, when X1 is on, Y5 through Y22 will be set (on) for the corresponding output module(s).

*DirectSOFT*

```
        X1                              Y5   Y22
    ——| |——————————————————————————————( SETI )
```

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ← |
|---|---|---|---|

| SET | SHFT | I | SHFT | Y(OUT) | 5 | Y(OUT) | 2 | 2 | ← |
|---|---|---|---|---|---|---|---|---|---|
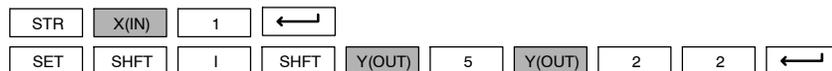
In the following example, when X1 is on, Y5 through Y22 will be reset (off) for the corresponding output module(s).

*DirectSOFT*

```
        X1                              Y5   Y22
    ——| |——————————————————————————————( RSTI )
```

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ← |
|---|---|---|---|

| RST | SHFT | I | SHFT | Y(OUT) | 5 | Y(OUT) | 2 | 2 | ← |
|---|---|---|---|---|---|---|---|---|---|

**Load Immediate (LDI)**

| × | × | ✓ |
|---|---|---|
| 430 | 440 | 450 |

| ✓ | ✓ |
|---|---|
| DS | HPP |

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.
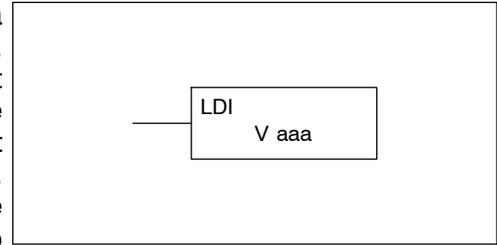
```
        ┌─────────────┐
────────┤ LDI         │
        │     V aaa   │
        └─────────────┘
```

| Operand Data Type | | DL450 Range |
|---|---|---|
| | | **aaaaa** |
| Inputs | V | 40400 – 40477 |

In the following example, when C0 is on, the binary pattern of X10–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 on a full CPU scan to occur).

**DirectSOFT**

```
     C0          ┌──────────┐
─────┤ ├─────┬───┤ LDI      │
               │   │  V40400  │
               │   └──────────┘
               │   Load the inputs from X0 to
               │   X17 into the accumulator,
               │   immediately.
               │
               │   ┌──────────┐
               └───┤ OUTI     │
                   │  V40502  │
                   └──────────┘
                   Output the value in the accumu-
                   lator to output points Y40 to Y57.
```

Location
V40400

| X17 | X16 | X15 | X14 | X13 | X12 | X11 | X10 | X7 | X6 | X5 | X4 | X3 | X2 | X1 | X0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ON | OFF | ON | ON | OFF | ON | OFF | OFF | ON | OFF | ON | ON | OFF | ON | OFF | ON |

Unused accumulator bits are set to zero

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Acc. 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |

Location
V40502

| Y57 | Y56 | Y55 | Y54 | Y53 | Y52 | Y51 | Y50 | Y47 | Y46 | Y45 | Y44 | Y43 | Y42 | Y41 | Y40 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ON | OFF | ON | ON | OFF | ON | OFF | OFF | ON | OFF | ON | ON | OFF | ON | OFF | ON |

Handheld Programmer Keystrokes

| STR | C(CR) | 0 | ↵ | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| LD | SHFT | I | SHFT | V | 4 | 0 | 4 | 0 | 0 | ↵ |
| OUT | SHFT | I | SHFT | V | 4 | 0 | 5 | 4 | 0 | ↵ |

**Load Immediate Formatted (LDIF)**

| × | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

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.

```
          ┌─────────────────────┐
──────────┤ LDIF          X aaa  │
          │               K bbb  │
          └─────────────────────┘
```
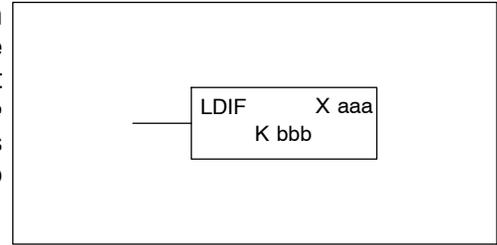
| Operand Data Type | | DL440 Range | | DL450 Range | |
|---|---|---|---|---|---|
| | | **aaa** | **bbb** | **aaa** | **bbb** |
| Inputs | X | 0–477 | –– | 0–1777 | –– |
| Constant | K | –– | 1–32 | –– | 1–32 |

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).

*DirectSOFT*



Load the value of 8 consecutive location into the accumulator starting with X10

Copy the value of the lower 8 bits of the accumulator to Y30 –Y37

Unused accumulator bits are set to zero

Handheld Programmer Keystrokes

Standard
RLL Instructions

## Out Immediate (OUTI)

| × | × | ✓ |
|---|---|---|
| 430 | 440 | 450 |

| ✓ | | ✓ |
|---|---|---|
| DS | | HPP |
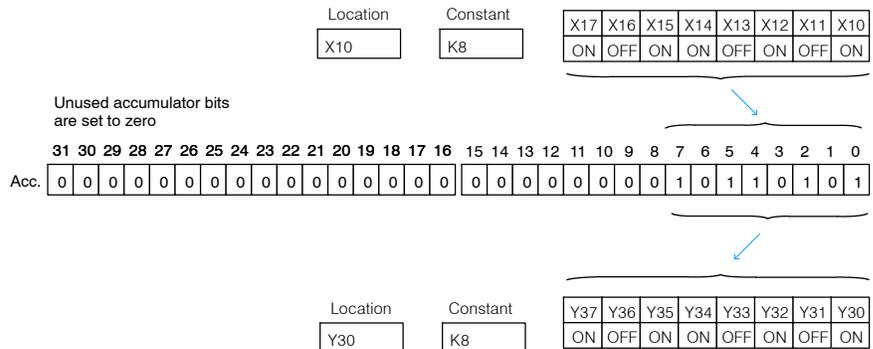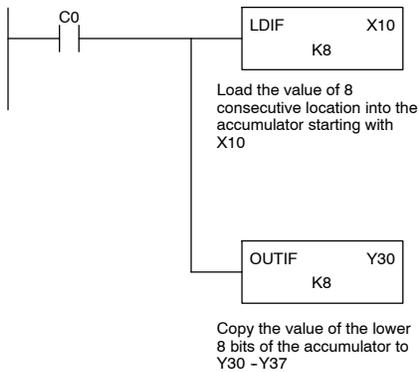
The Out Immediate instruction outputs a 16-bit binary value from the accumulator to a V-memory address *at the time the instruction is executed*. The valid address range includes all output point addresses on the local base. This instruction may be used instead of the OUTIF instruction which requires you to specify the number of input points.

```
        ┌──────────┐
────────┤ OUTI     │
        │   V aaaaa│
        └──────────┘
```

| Operand Data Type | | DL450 Range |
|---|---|---|
| | | **aaaaa** |
| Inputs | V | 40400 – 40477 |

In the following example, when C0 is on, the binary pattern of X10–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 outputs (without waiting for the CPU scan).

*DirectSOFT*



```
   C0        ┌──────────┐
───┤├────┬───┤ LDI      │
        │   │   V40400  │
        │   └──────────┘
        │   Load the inputs from X0 to
        │   X17 into the accumulator,
        │   immediately.
        │
        │   ┌──────────┐
        └───┤ OUTI     │
            │   V40542  │
            └──────────┘
            Output the value in the accumu-
            lator to output points Y40 to Y57.
```

Location V40400

| X17 | X16 | X15 | X14 | X13 | X12 | X11 | X10 | X7 | X6 | X5 | X4 | X3 | X2 | X1 | X0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ON | OFF | ON | ON | OFF | ON | OFF | OFF | ON | OFF | ON | ON | OFF | ON | OFF | ON |

Unused accumulator bits are set to zero

Acc.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |

Location V40542

| Y57 | Y56 | Y55 | Y54 | Y53 | Y52 | Y51 | Y50 | Y47 | Y46 | Y45 | Y44 | Y43 | Y42 | Y41 | Y40 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ON | OFF | ON | ON | OFF | ON | OFF | OFF | ON | OFF | ON | ON | OFF | ON | OFF | ON |

Handheld Programmer Keystrokes

| STR | C(CR) | 0 | ↵ |
|---|---|---|---|

| LD | SHFT | I | SHFT | V | 4 | 0 | 4 | 0 | 0 | ↵ |
|---|---|---|---|---|---|---|---|---|---|---|

| OUT | SHFT | I | SHFT | V | 4 | 0 | 5 | 4 | 0 | ↵ |
|---|---|---|---|---|---|---|---|---|---|---|

Standard RLL Instructions

**Out Immediate Formatted (OUTIF)**

| × | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | | ✓ |
|---|---|---|
| **DS** | | **HPP** |

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.

```
        OUTIF      Y aaa
                   K bbb
```
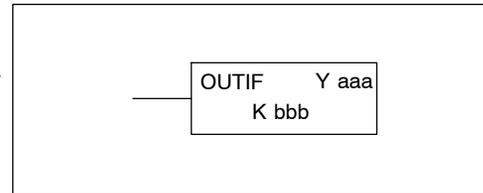
| Operand Data Type | | DL440 Range | | DL450 Range | |
|---|---|---|---|---|---|
| | | **aaa** | **bbb** | **aaa** | **bbb** |
| Output | Y | 0–477 | -- | 0–1777 | -- |
| Constant | K | -- | 1–32 | -- | 1–32 |

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**

```
   C0
   ┤ ├───────┬──────────────
             │   LDIF      X10
             │             K8
```

Load the value of 8 consecutive location into the accumulator starting with X10

```
             │   OUTIF     Y30
             │             K8
```

Copy the value in the lower 8 bits of the accumulator to Y30–Y37

| Location | Constant |
|---|---|
| X10 | K8 |

| X17 | X16 | X15 | X14 | X13 | X12 | X11 | X10 |
|---|---|---|---|---|---|---|---|
| ON | OFF | ON | ON | OFF | ON | OFF | ON |

Unused accumulator bits are set to zero

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |

Acc.

| Location | Constant |
|---|---|
| Y30 | K8 |

| Y37 | Y36 | Y35 | Y34 | Y33 | Y32 | Y31 | Y30 |
|---|---|---|---|---|---|---|---|
| ON | OFF | ON | ON | OFF | ON | OFF | ON |

Handheld Programmer Keystrokes

| STR | C(CR) | 0 | ↵ |
|---|---|---|---|

| LD | SHFT | I | F | SHFT | X(IN) | 1 | 0 | K(CON) | 8 | ↵ |
|---|---|---|---|---|---|---|---|---|---|---|

| OUT | SHFT | I | F | SHFT | Y(OUT) | 3 | 0 | K(CON) | 8 | ↵ |
|---|---|---|---|---|---|---|---|---|---|---|

# Timer, Counter, and Shift Register Instructions

**Using Timers**

Timers are used to time an event for a desired length of time. 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 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 discrete bit associated with each timer to indicate 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.
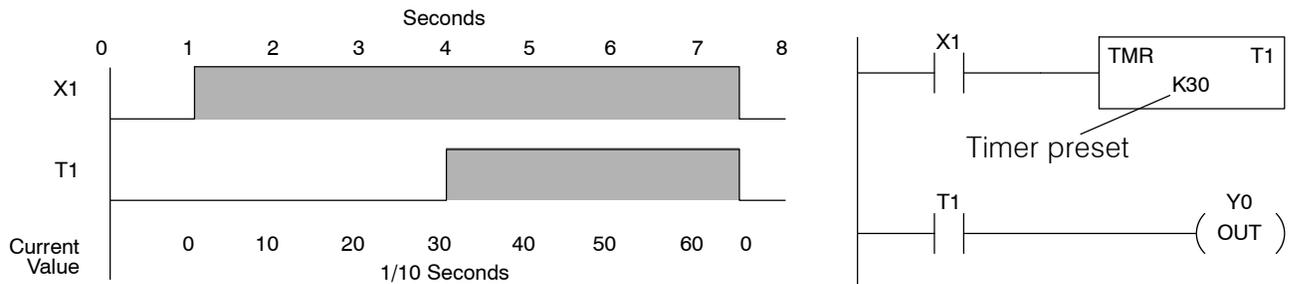
The accumulating timer works similarly to the regular timer, but two inputs are required. The start/stop input starts and stops the timer. When the timer stops, the elapsed time is maintained. When the timer starts again, the timing continues from the elapsed time. When the reset input is turned on, 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.

Standard
RLL Instructions
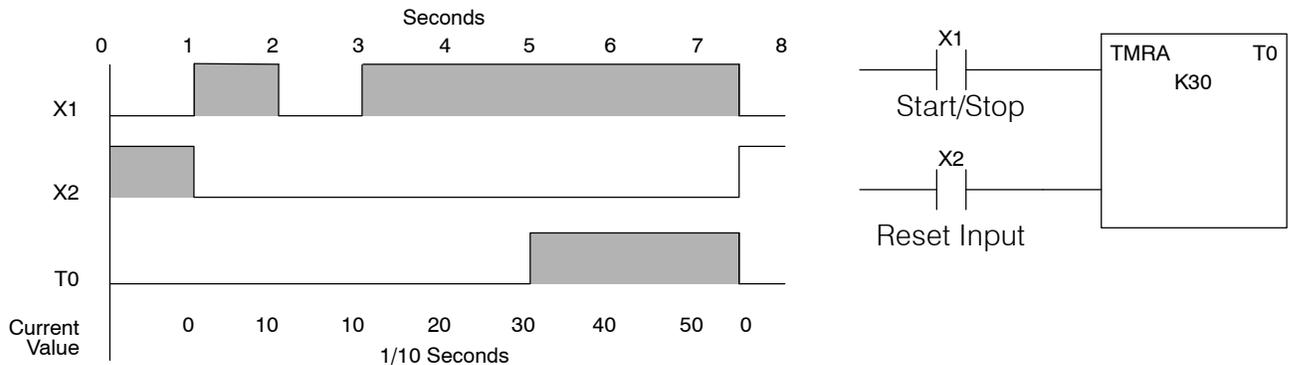
**Timer (TMR) and Timer Fast (TMRF)**

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

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).

**Instruction Specifications**

**Timer Reference** (Taaa): Specifies the timer number.

**Preset Value** (Bbbb): Constant value (K) or a V–memory location. (V locations are 16-bit words.)

**Current Value**: Timer current values 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. (V locations are 16-bit words.)



The timer discrete status bit and the current value is not specified in the timer instruction.

**Discrete Status Bit**: The discrete status bit is accessed by referencing the associated T memory location. 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.
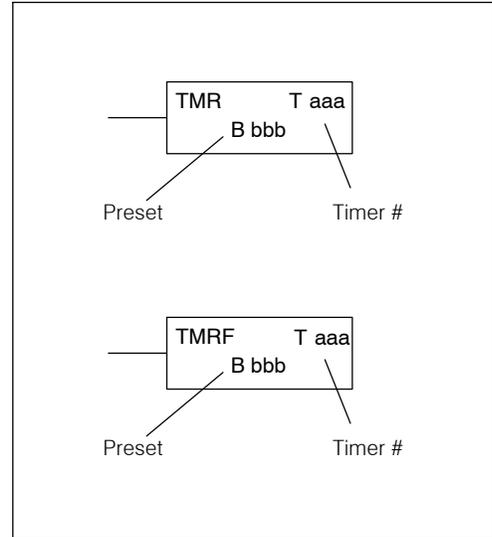
| Operand Data Type | | DL430 Range | | DL440 Range | | DL450 Range | |
|---|---|---|---|---|---|---|---|
| | **B** | **aaa** | **bbb** | **aaa** | **bbb** | **aaa** | **bbb** |
| Timers | T | 0–177 | -- | 0–377 | -- | 0–377 | -- |
| V–memory for preset values | V | -- | 1400–7377 | -- | 1400–7377 10000–17777 | -- | 1400–7377 10000–17777 |
| Pointers (preset only) | P | -- | -- | -- | 1400–7377 10000–17777 | -- | 1400–7377 10000–37777 |
| Constants (preset only) | K | -- | 0–9999 | -- | 0–9999 | -- | 0–9999 |
| Timer discrete status bits | T | 0–177 | | 0–377 | | 0–377 | |
| Timer current values | V /T* | 0–177 | | 0–377 | | 0–377 | |

There are two methods of programming timers. You can perform functions when the timer reaches the specified preset using the discrete status bit, or use the comparative contacts to perform functions at different time intervals based on one timer. The following examples show each method of using timers.

**NOTE: *** For the Handheld Programmer, both the Timer discrete status bits and current value can be accessed with the same data type (example T2). The way the data type is used determines if it is a status bit or a current value. Any comparative instruction using T2 will access the current value, all other instructions using T2 will access the status bit. Current values may also be accessed by the V-memory location. For *Direct*SOFT, the use of T2 will refer to the timer's discrete status bit. You should use V2 (or the alias TA2) to refer to the current value.

## 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 after 7.5 seconds turning the discrete status bit off and resetting the timer current value to 0.

**DirectSOFT**

*Timing Diagram*

```
       X1                    TMR    T2
    ---| |---              ---------K30

       T2                         Y0
    ---| |---                 ---( OUT )---
```

**Handheld Programmer Keystrokes**

| STR | X(IN) | 1 | ⏎ |
|-----|-------|---|---|
| TMR | TMR | 2 | K(CON) | 3 | 0 | ⏎ |

| STR | TMR | 2 | ⏎ |
|-----|-----|---|---|

| OUT | Y(OUT) | 0 | ⏎ |
|-----|--------|---|---|

## Timer Example Using Comparative Contacts
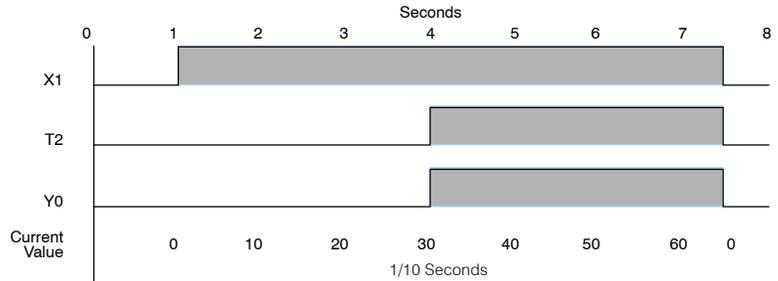
In the following example, a single input timer is used with a preset of 234.5 seconds. Comparative contacts are used to energize Y3, Y4, and Y5 at one second intervals respectively. When X1 is turned off the timer will be reset to 0 and the comparative contacts will turn off Y3, Y4, and Y5.

**DirectSOFT** (see Note)

*Timing Diagram*

```
       X1                    TMR    T20
    ---| |---              ---------K2345

      TA20   K10                   Y3
    ---|≥|---                  ---( OUT )---

      TA20   K20                   Y4
    ---|≥|---                  ---( OUT )---

      TA20   K30                   Y5
    ---|≥|---                  ---( OUT )---
```
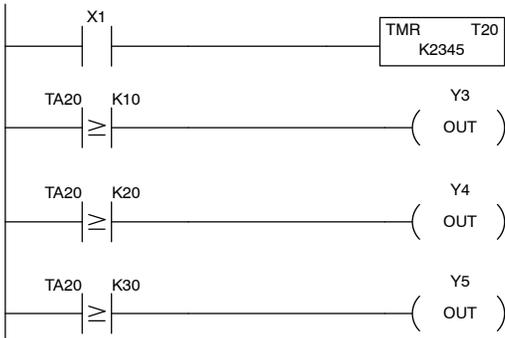
**Handheld Programmer Keystrokes**

| STR | X(IN) | 1 | ⏎ |
|-----|-------|---|---|
| TMR | TMR | 2 | 0 | K(CON) | 2 | 3 | 4 | 5 | ⏎ |
| STR | TMR | 2 | 0 | K(CON) | 1 | 0 | ⏎ |
| OUT | Y(OUT) | 3 | ⏎ |
| STR | TMR | 2 | 0 | K(CON) | 2 | 0 | ⏎ |
| OUT | Y(OUT) | 4 | ⏎ |
| STR | TMR | 2 | 0 | K(CON) | 3 | 0 | ⏎ |
| OUT | Y(OUT) | 5 | ⏎ |
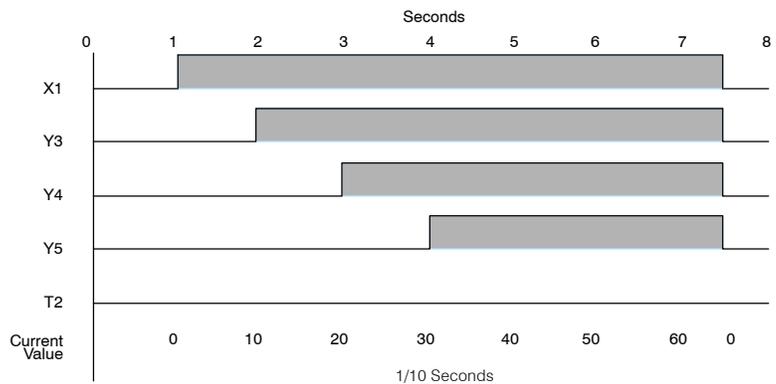
**NOTE:** Since this representation is showing a *Direct*SOFT example, you would use the alias TA20 (or V20) instead of T20, which would be necessary for the equivalent rung entered with the Handheld Programmer.

**Accumulating Timer (TMRA) and Accumulating Fast Timer (TMRAF)**

| ✓ | ✓ | ✓ |
|---|---|---|

**430 440 450**

| ✓ | | ✓ |
|---|---|---|

**DS    HPP**

The Accumulating Timer is a 0.1 second two input timer that times to a maximum of 9999999.9. The Accumulating Fast Timer is a 0.01 second two input timer that times to a maximum of 999999.99. These timers have two inputs, an enable and a reset. The timer will start timing when the enable is on and stop timing when the enable is off without resetting the current value to 0. The reset will reset the timer when on and allow the timer to time when off.

**Instruction Specifications**

**Timer Reference** (Taaa): Specifies the timer number.

**Preset Value** (Bbbb): Constant value (K) or a V–memory location. (V locations are 32-bit double words.)

**Current Value**: Timer current value is a double word accessed by referencing the associated V or T memory location*. For example, the timer current value for T0 resides in V-memory location V0 and V1. (V locations are 16-bit words.)

**Discrete Status Bit**: The discrete status bit is accessed by referencing the associated T memory location. 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.

Enable — TMRA    T aaa
B bbb
Reset —

Preset                    Timer #

Enable — TMRAF    T aaa
B bbb
Reset —

Preset                    Timer #

The timer discrete status bit and the current value is not specified in the timer instruction.

**Caution: The TMRA uses two consecutive timer locations, since the preset can now be 8 digits, which requires two V–memory locations. For example, if TMRA T0 is used in the program, the next available timer is T2. Or if T0 was a normal timer, and T1 was an accumulating timer, then the next available timer would be T3.**
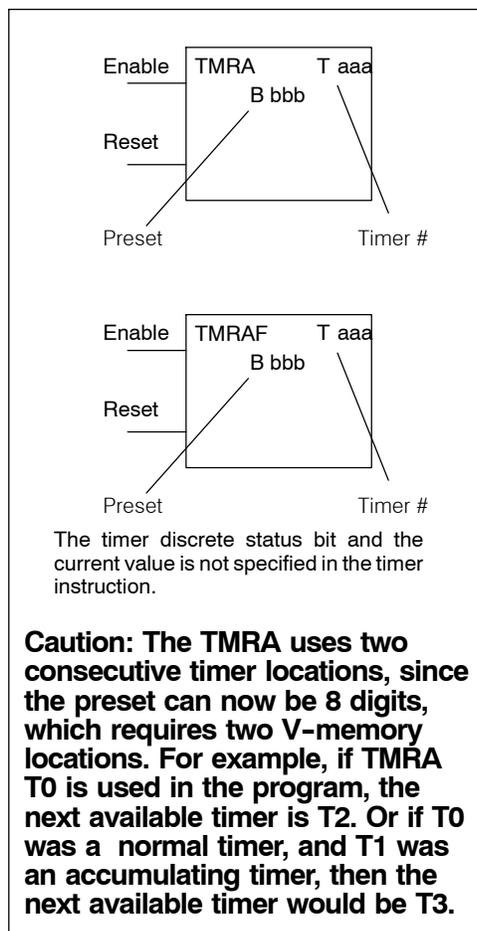
| Operand Data Type | | DL430 Range | | DL440 Range | | DL450 Range | |
|---|---|---|---|---|---|---|---|
| | **B** | **aaa** | **bbb** | **aaa** | **bbb** | **aaa** | **bbb** |
| Timers | T | 0–176 | -- | 0–376 | -- | 0–376 | -- |
| V–memory for preset values | V | -- | 1400–7377 | -- | 1400–7377 10000–17777 | -- | 1400–7377 10000–17777 |
| Pointers (preset only) | P | -- | -- | -- | 1400–7377 10000–17777 | -- | 1400–7377 10000–37777 |
| Constants (preset only) | K | -- | 0–99999999 | -- | 0–99999999 | -- | 0–99999999 |
| Timer discrete status bits | T | 0–177 | | 0–377 | | 0–377 | |
| Timer current values | V/T* | 0–177 | | 0–377 | | 0–377 | |

There are two methods of programming timers. You can perform functions when the timer reaches the specified preset using the the discrete status bit, or use the comparative contacts to perform functions at different time intervals based on one timer. The following examples show each method of using timers.

**NOTE:*** For the Handheld Programmer, both the Timer discrete status bits and current value can be accessed with the same data type (example T2). The way the data type is used determines if it is a status bit or a current value. Any comparative instruction using T2 will access the current value, all other instructions using T2 will access the status bit. Current values may also be accessed by the V-memory location. For *Direct*SOFT, the use of T2 will refer to the timer's discrete status bit. You should use V2 (or the alias TA2) to refer to the current value.

**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 the timer times for 1 second , stops for one second, then resumes timing. The timer will reset when C10 turn on after 5.5 seconds turning the discrete status bit off and resetting the timer current value to 0.

*DirectSOFT*

Timing Diagram



Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |

| STR | C(CR) | 1 | 0 | ↵ |

| TMR | SHFT | A | SHFT | TMR | 6 |

Handheld Programmer Keystrokes (cont)

| K(CON) | 3 | 0 | ↵ |

| STR | TMR | 6 | ↵ |

| OUT | Y(OUT) | 1 | 0 | ↵ |
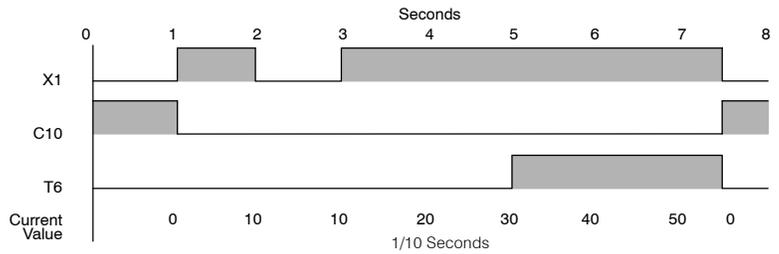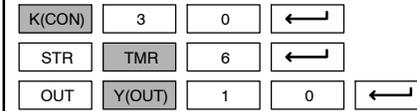
**Accumulator Timer Example Using Comparative Contacts**

In the following example, a single input timer is used with a preset of 23458.9 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.

Contacts
Direct SOFT

Timing Diagram



Handheld Programmer Keystrokes



Handheld Programmer Keystrokes (cont d)



**NOTE:** Since this representation is showing a *Direct*SOFT example, you would use the alias TA20 (or V20) instead of T20, which would be necessary for the equivalent rung entered with the Handheld Programmer.

Standard
RLL Instructions

**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 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 up down counter 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 input, counter reset, associated discrete bit, current value, and counter preset.



The stage counter has a count input and is reset by the RST instruction. This instruction is useful when programming RLL$^{PLUS}$, by allowing you to reference the same counter from multiple stages. 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)**

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | | ✓ |
|---|---|---|
| **DS** | | **HPP** |

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. (V locations are 16-bit words.)

**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. (V locations are 16-bit words.)

The counter discrete status bit and the current value are not specified in the counter instruction.

**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.

| Operand Data Type | | DL430 Range | | DL440 Range | | DL450 Range | |
|---|---|---|---|---|---|---|---|
| | **B** | **aaa** | **bbb** | **aaa** | **bbb** | **aaa** | **bbb** |
| Counters | CT | 0–177 | -- | 0–177 | -- | 0–377 | -- |
| V-memory (preset only) | V | -- | 1400–7377 | -- | 1400–7377 10000–17777 | -- | 1400–7377 10000–37777 |
| Pointers (preset only) | P | -- | -- | -- | 1400–7377 10000–17777 | -- | 1400–7377 10000–37777 |
| Constants (preset only) | K | -- | 0–9999 | -- | 0–9999 | -- | 0–9999 |
| Counter discrete status bits | CT | 0–177 | | 0–177 | | 0–377 | |
| Counter current values | V/CT* | 1000–1177 | | 1000–1177 | | 1000–1377 | |

**NOTE:*** For the Handheld Programmer, both the Counter discrete status bits and current value can be accessed with the same data type (example CT2). The way the data type is used determines if it is a status bit or a current value. Any comparative instruction using CT2 will access the current value, all other instructions using CT2 will access the status bit. Current values may also be accessed by the V-memory location. For *Direct*SOFT, the use of CT2 will refer to the timer's discrete status bit. You should use V1002 (or the alias CTA2) to refer to the current value.

Standard
RLL Instructions

**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 Y10. 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.

*DirectSOFT*
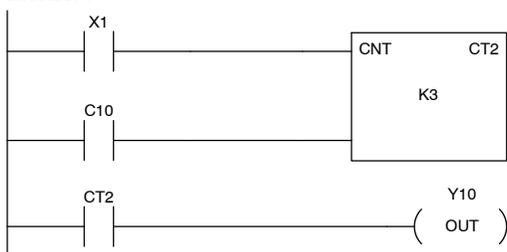
Counting diagram
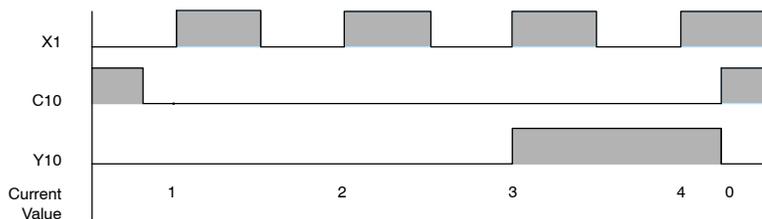


Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ← |
| STR | C(CR) | 1 | 0 | ← |
| CNT | CNT | 2 | K(CON) | 3 | ← |
| STR | CNT | 2 | ← |
| OUT | Y(OUT) | 1 | 0 | ← |

**Counter Example Using Comparative Contacts**

In the following example, when X1 is 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. The comparative contacts will turn off when the counter is reset. When the reset C10 turns on, the counter status bit will turn off and the counter current value will be 0.

*DirectSOFT* (see Note)
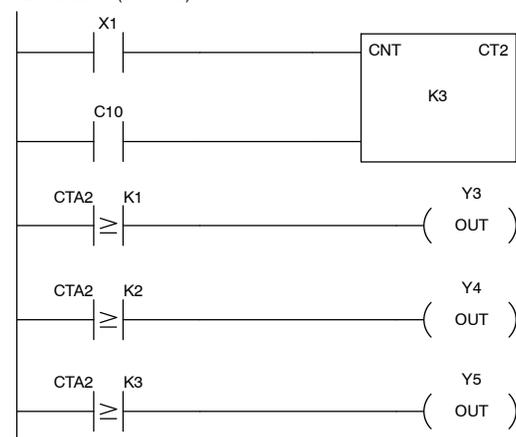
Counting diagram
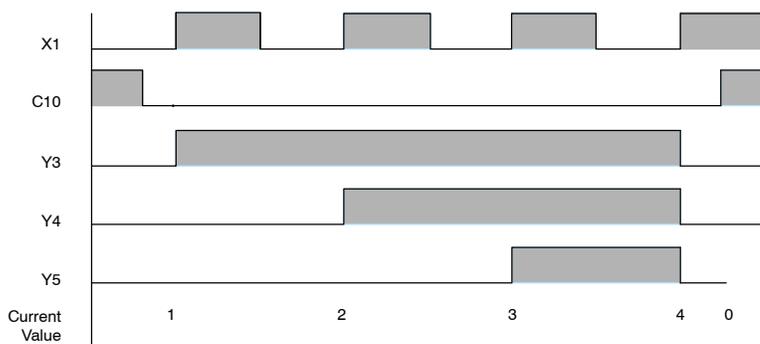


Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ← |
| STR | C(CR) | 1 | 0 | ← |
| CNT | CNT | 2 | K(CON) | 3 | ← |
| STR | CNT | 2 | K(CON) | 1 | ← |
| OUT | Y(OUT) | 3 | ← |

Handheld Programmer Keystrokes (cont)

| STR | CNT | 2 | K(CON) | 2 | ← |
| OUT | Y(OUT) | 4 | ← |
| STR | CNT | 2 | K(CON) | 3 | ← |
| OUT | Y(OUT) | 5 | ← |

**NOTE:** Since this representation is showing a *Direct*SOFT example, you would use the alias CTA2 (or V1002) instead of CT2, which would be necessary for the equivalent rung entered with the Handheld.

## Stage Counter (SGCNT)

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

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.
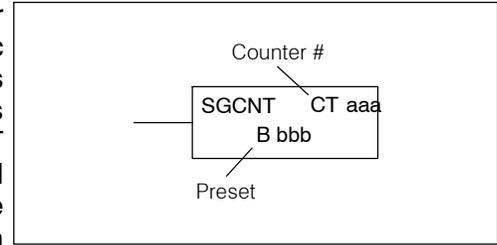


The counter discrete status bit and the current value are not specified in the counter instruction.

### Instruction Specifications

**Counter Reference** (CTaaa): Specifies the counter number.

**Preset Value** (Bbbb): Constant value (K) or a V–memory location. (V locations are 16-bit words.)

**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. (V locations are 16-bit words.)

**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.

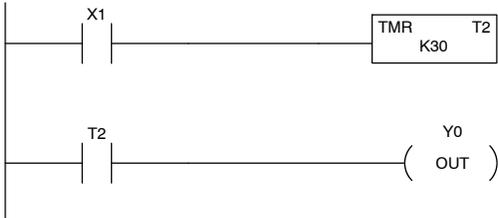| Operand Data Type | | DL430 Range | | DL440 Range | | DL450 Range | |
|---|---|---|---|---|---|---|---|
| | **B** | **aaa** | **bbb** | **aaa** | **bbb** | **aaa** | **bbb** |
| Counters | CT | 0–177 | -- | 0–177 | -- | 0–377 | -- |
| V-memory | V | -- | 1400–7377 | -- | 1400–7377<br>10000–17777 | -- | 1400–7377<br>10000–17777 |
| Pointers (preset only) | P | -- | -- | -- | 1400–7377<br>10000–17777 | -- | 1400–7377<br>10000–37777 |
| Constants | K | -- | 0–9999 | -- | 0–9999 | -- | 0–9999 |
| Counter discrete status bits | CT | 0–177 | | 0–177 | | 0–377 | |
| Counter current values | V/CT* | 1000–1177 | | 1000–1177 | | 1000–1377 | |

**NOTE:*** For the Handheld Programmer, both the Stage Counter discrete status bits and current value can be accessed with the same data type (example CT2). The way the data type is used determines if it is a status bit or a current value. Any comparative instruction using CT2 will access the current value, all other instructions using CT2 will access the status bit. Current values may also be accessed by the V-memory location. For *Direct*SOFT, the use of CT2 will refer to the timer's discrete status bit. You should use V1002 (or the alias CTA2) to refer to the current value.

**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 Y10. 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.
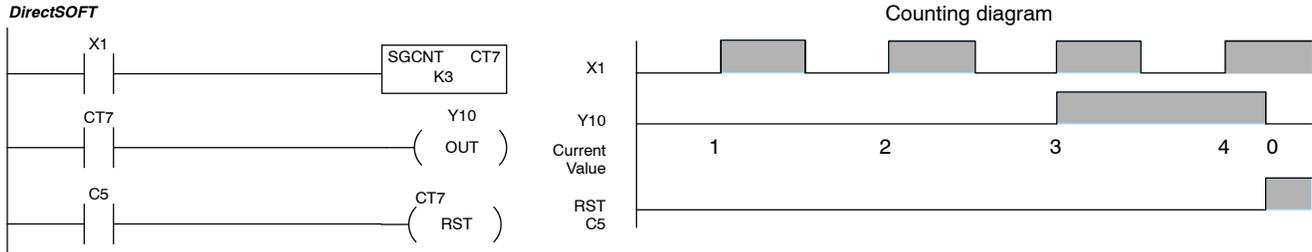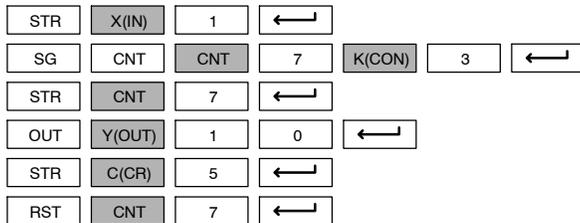
*DirectSOFT*

| X1 | | SGCNT    CT7 |
|----|--|--------------|
| | | K3 |

| CT7 | | Y10 |
|-----|--|-----|
| | | ( OUT ) |

| C5 | | CT7 |
|----|--|-----|
| | | ( RST ) |

Counting diagram



Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ← |
|-----|-------|---|---|
| SG | CNT | CNT | 7 | K(CON) | 3 | ← |
| STR | CNT | 7 | ← |
| OUT | Y(OUT) | 1 | 0 | ← |
| STR | C(CR) | 5 | ← |
| RST | CNT | 7 | ← |

**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.

*tSOFT* (see Note)

| X1 | | SGCNT    CT2 |
|----|--|--------------|
| | | K10 |

| CTA2 | K1 | Y3 |
|------|----|----|
| ≥ | | ( OUT ) |

| CTA2 | K2 | Y4 |
|------|----|----|
| ≥ | | ( OUT ) |

| CTA2 | K3 | Y5 |
|------|----|----|
| ≥ | | ( OUT ) |

Counting diagram



Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ← |
|-----|-------|---|---|
| SG | CNT | CNT | 2 | K(CON) | 1 | 0 | ← |
| STR | CNT | 2 | K(CON) | 1 | ← |
| OUT | Y(OUT) | 3 | ← |

Handheld Programmer Keystrokes (cont)

| STR | CNT | 2 | K(CON) | 2 | ← |
|-----|-----|---|--------|---|---|
| OUT | Y(OUT) | 4 | ← |
| STR | CNT | 2 | K(CON) | 3 | ← |
| OUT | Y(OUT) | 5 | ← |

**NOTE:** Since this representation is showing a *Direct*SOFT example, you would use the alias CTA2 (or V1002) instead of CT2, which would be necessary for the equivalent rung entered with the Handheld.

Standard
RLL Instructions
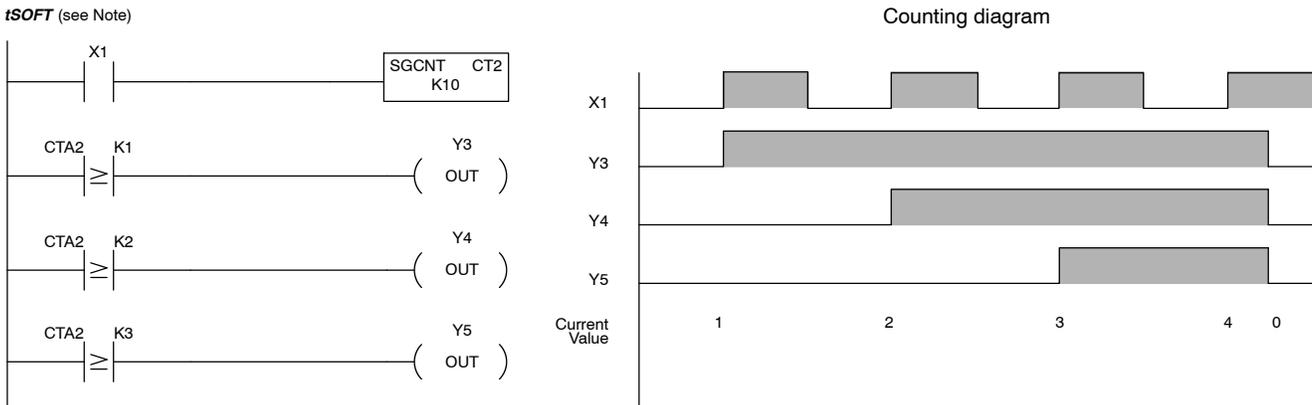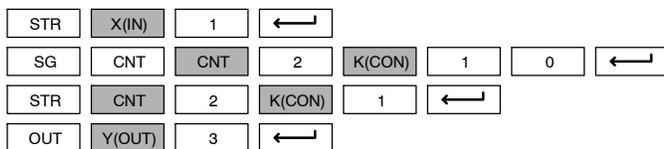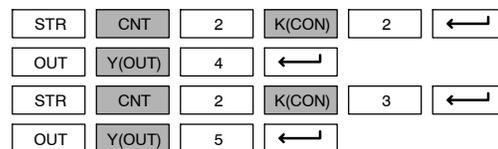
## Up/Down Counter (UDC)

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | | ✓ |
|---|---|---|
| **DS** | | **HPP** |
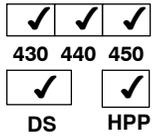
This Up/Down Counter counts up on each off to on transition of the Up input and counts down on each off to on transition of the Down input. The counter is reset to 0 when the Reset input is on. The count range is 0–99999999. The count input not being used must be off in order for the active count input to function.
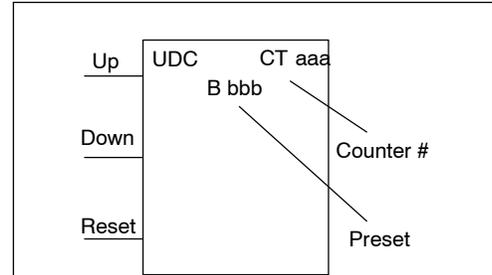
### Instruction Specification

**Counter Reference** (CTaaa): Specifies the counter number.

**Preset Value** (Bbbb): Constant value (K) or two consecutive V–memory locations. (V locations are 16-bit words.)

**Current Values**: Current count is a double word value 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 CT5 resides in V–memory location V1005 and V1006. (V locations are 16-bit words.)

**Discrete Status Bit**: The discrete status bit is accessed by referencing the associated CT memory location. It will be on if value is equal to or greater than the preset value. For example the discrete status bit for counter 2 would be CT2.



The counter discrete status bit and the current value is not specified in the counter instruction.

**Caution: The UDC uses two consecutive counter locations, since the preset can now be 8 digits, which requires two V–memory locations. For example, if UDC CT0 is used in the program, the next available counter is CT2. Or if CT0 was a normal counter, and CT1 was an up/down counter, then the next available counter would be CT3.**

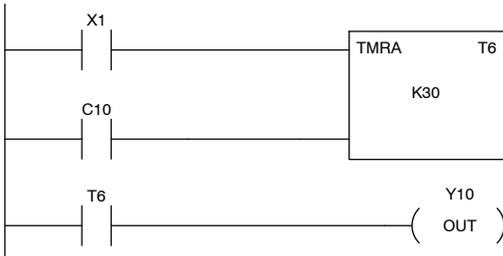| Operand Data Type | | DL430 Range | | DL440 Range | | DL450 Range | |
|---|---|---|---|---|---|---|---|
| | **B** | **aaa** | **bbb** | **aaa** | **bbb** | **aaa** | **bbb** |
| Counters | CT | 0–176 | – – | 0–176 | – – | 0–376 | – – |
| V–memory | V | – – | 1400–7377 | – – | 1400–7377 10000–17777 | – – | 1400–7377 10000–37777 |
| Pointers (preset only) | P | – – | – – | – – | 1400–7377 10000–17777 | – – | 1400–7377 10000–37777 |
| Constants | K | – – | 0–99999999 | – – | 0–99999999 | – – | 0–99999999 |
| Counter discrete status bits | CT | 0–177 | | 0–177 | | 0–377 | |
| Counter current values | V/ CT* | 1000–1177 | | 1000–1177 | | 1000–1377 | |

**NOTE:*** For the Handheld Programmer, both the Stage Counter discrete status bits and current value can be accessed with the same data type (example CT2). The way the data type is used determines if it is a status bit or a current value. Any comparative instruction using CT2 will access the current value, all other instructions using CT2 will access the status bit. Current values may also be accessed by the V-memory location. For *Direct*SOFT, the use of CT2 will refer to the timer's discrete status bit. You should use V1002 (or the alias CTA2) to refer to the current value.
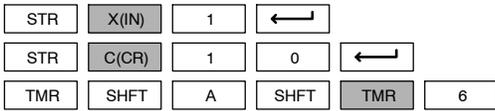
### Up/Down Counter Example Using Discrete Status Bits

In the following example if X2 and X3 are off when X1 toggles from off to on the counter will increment by one. 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 | X(IN) | 1 | ← |
| STR | X(IN) | 2 | ← |
| STR | X(IN) | 3 | ← |
| SHFT | U | D | C | SHFT | CNT | 2 |

Handheld Programmer Keystrokes (cont)

| K(CON) | 3 | ← |
| STR | CNT | 2 | ← |
| OUT | Y(OUT) | 1 | 0 | ← |
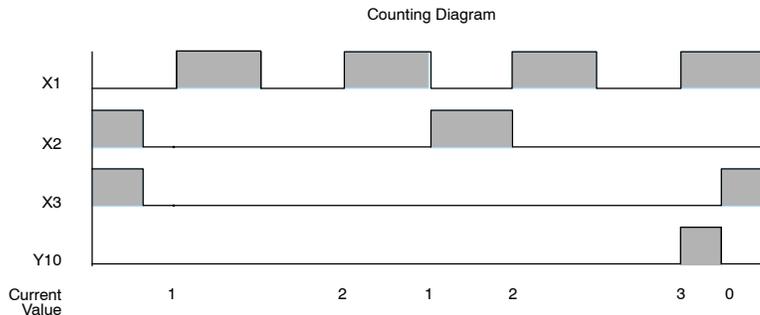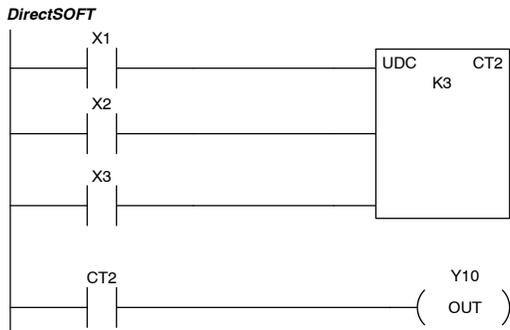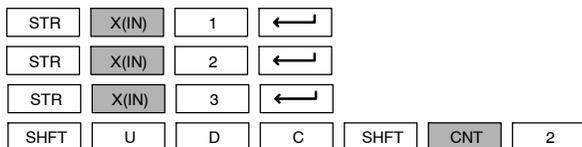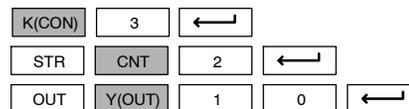
### Up/Down Counter Example Using Comparative Contacts

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. The comparative contacts will turn off when the counter is reset. When the reset X3 turns on, the counter status bit will turn off and the current value will be 0.

*DirectSOFT* (see Note)



Counting Diagram

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ← |
| STR | X(IN) | 2 | ← |
| STR | X(IN) | 3 | ← |
| SHFT | U | D | C | SHFT | CNT | 2 |
| V | 1 | 4 | 0 | 0 | ← |
| STR | CNT | 2 | K(CON) | 1 | ← |

Handheld Programmer Keystrokes (cont)

| OUT | Y(OUT) | 3 | ← |
| STR | CNT | 2 | K(CON) | 2 | ← |
| OUT | Y(OUT) | 4 | ← |
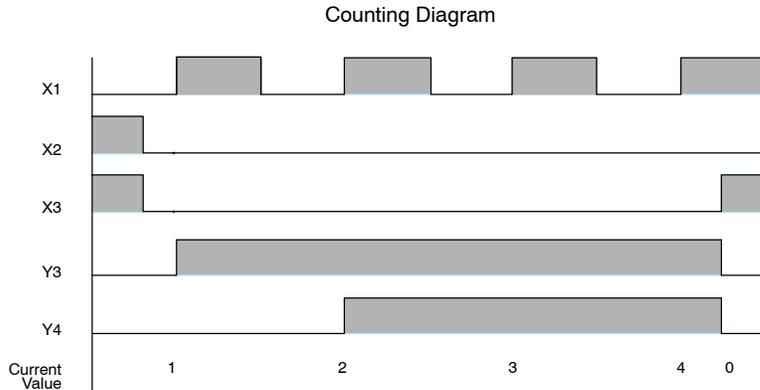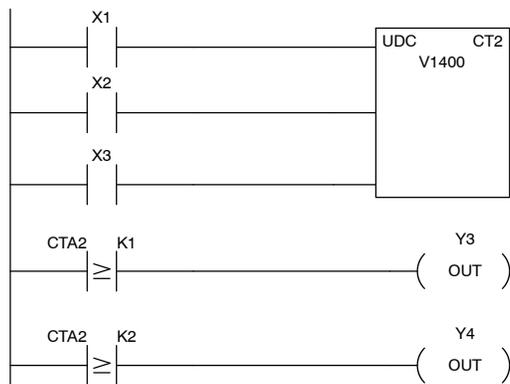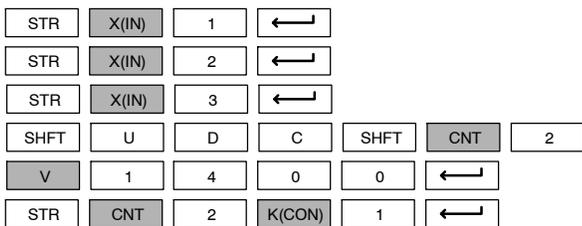
**NOTE:** Since this representation is showing a *Direct*SOFT example, you would use the alias CTA2 (or V1002) instead of CT2, which would be necessary for the equivalent rung entered with the Handheld.

## Shift Register (SR)

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

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 end at the end of an 8 bit boundary.

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.

```
DATA ────── SR

CLOCK ─────  From  C aaa

RESET ─────  To    C bbb
```
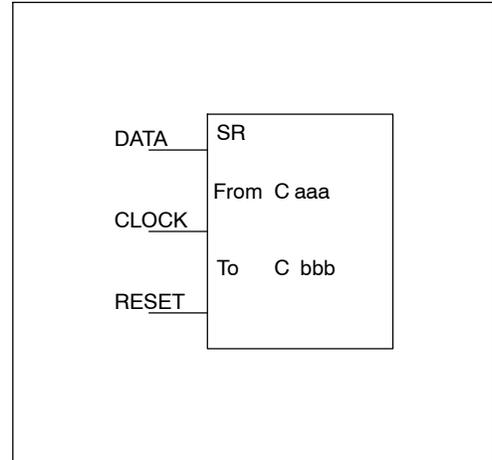
With each off to on transition of the clock input, the bits which make up the shift register block are shifted by one bit position and the status of the data input is placed into the starting bit position in the shift register. The direction of the shift depends on the entry in the From and To fields. From C0 to C17 would define a block of sixteen bits to be shifted from lower address to higher address. From C17 to C0 would define a block of sixteen bits, to be shifted from higher address to lower address. 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 | | DL430 Range | | DL440 Range | | DL440 Range | |
|---|---|---|---|---|---|---|---|
| | **B** | **aaa** | **bbb** | **aaa** | **bbb** | **aaa** | **bbb** |
| Control Relay | C | 0–737 | 0–737 | 0–1777 | 0–1777 | 0–3777 | 0–3777 |

*DirectSOFT*

```
     X1      Data Input      SR
    ─┤ ├─

     X2      Clock Input    From   C0
    ─┤ ├─

     X3      Reset Input     To    C17
    ─┤ ├─
```
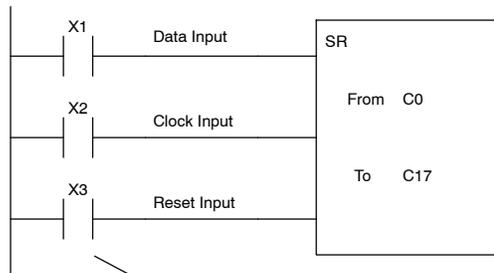
Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ← |
|---|---|---|---|
| STR | X(IN) | 2 | ← |
| STR | X(IN) | 3 | ← |
| SR | C(CR) | 0 | |
| C(CR) | 1 | 7 | ← |

Inputs on Successive Scans      Shift Register Bits

| Data | Clock | Reset |
|---|---|---|
| 1 | ⌐ | 0 |
| 0 | ⌐ | 0 |
| 0 | ⌐ | 0 |
| 1 | ⌐ | 0 |
| 0 | ⌐ | 0 |
| – | – | 1 |

C0 ———————————— C17

# Accumulator/Data Stack Load and Output Instructions

**Using the Accumulator**

The accumulator in the DL405 series 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, or a 32-bit 2's complement 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 V1400 to V-memory location V1410.

X1 — LD V1400

Copy data from V1400 to the lower 16 bits of the accumulator

OUT V1410

Copy data from the lower 16 bits of the accumulator to V1410

V1400
| 8 | 9 | 3 | 5 |

Unused accumulator bits are set to zero

Acc. | 0 | 0 | 0 | 0 | 8 | 9 | 3 | 5 |

| 8 | 9 | 3 | 5 |
V1410
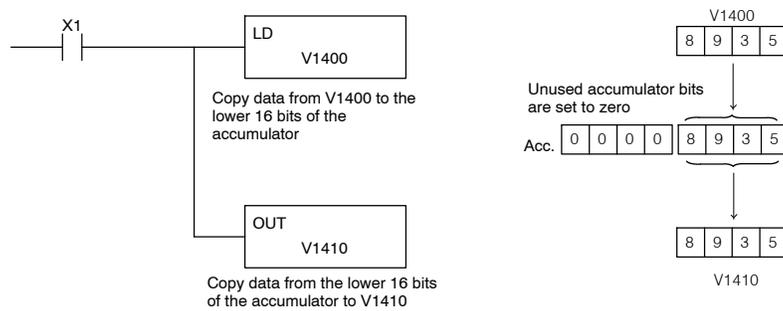
Since the accumulator is 32 bits and V-memory locations are 16 bits, the Load Double and Out Double (or variations thereof) use two consecutive V-memory locations or 8 digit BCD constants to copy data either to the accumulator from a V-memory address or from a V-memory address to the accumulator. For example, if you wanted to copy data from V-memory location V1400 and V1401 to V-memory location V1410 and V1411, the most efficient way to perform this function would be as follows:

X1 — LDD V1400

Copy data from V1400 and V1401 to the accumulator

OUTD V1410

Copy data from the accumulator to V1410 and V1411

V1401        V1400
| 6 | 7 | 3 | 9 | 5 | 0 | 2 | 6 |

Acc. | 6 | 7 | 3 | 9 | 5 | 0 | 2 | 6 |

| 6 | 7 | 3 | 9 | 5 | 0 | 2 | 6 |
V1411        V1410
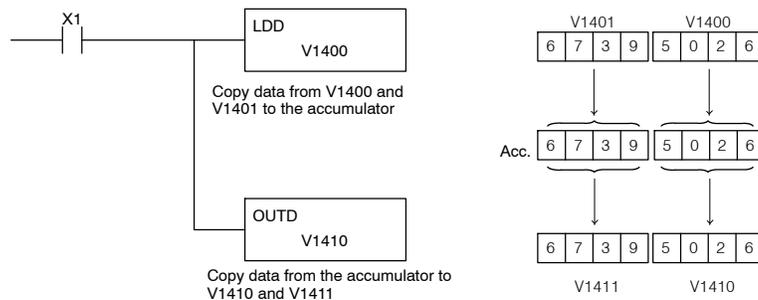
**Changing the Accumulator Data**

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 BCD value 4935 into the accumulator, shifts the data right 4 bits, and outputs the result to V1410.
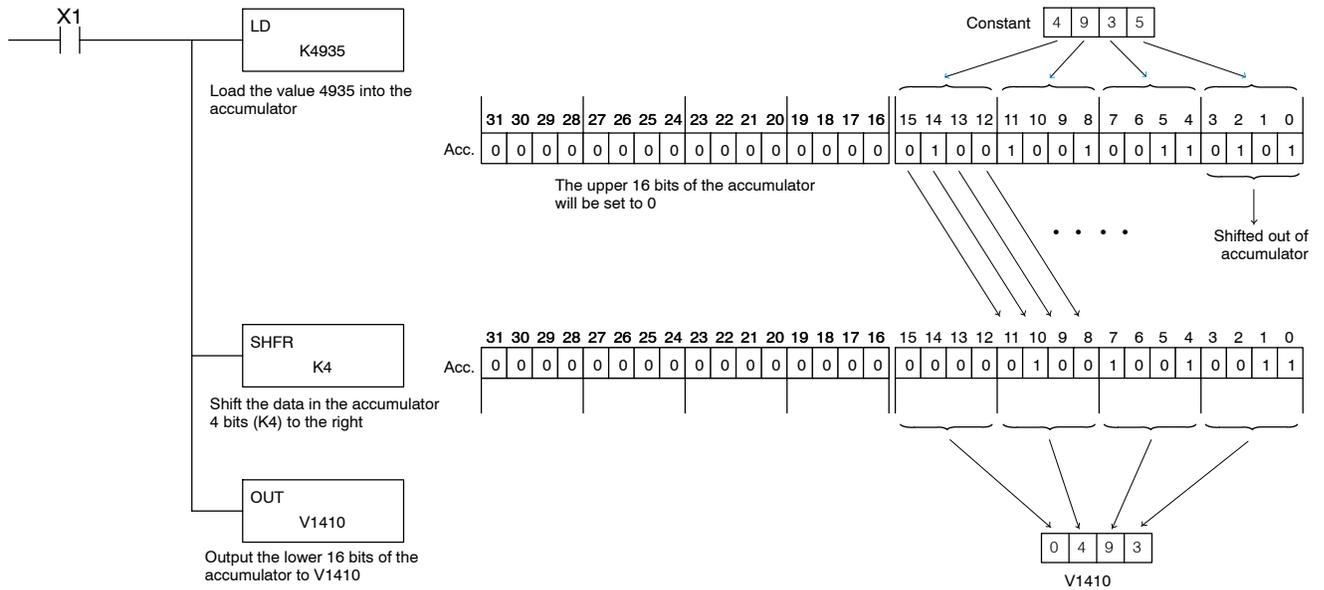


Some of the data manipulation instructions use 32 bits. They use two consecutive V-memory locations or 8 digit BCD constants to manipulate data in the accumulator. The following example rotates the value 67053101 two bits to the right and outputs the value to V1410 and V1411.
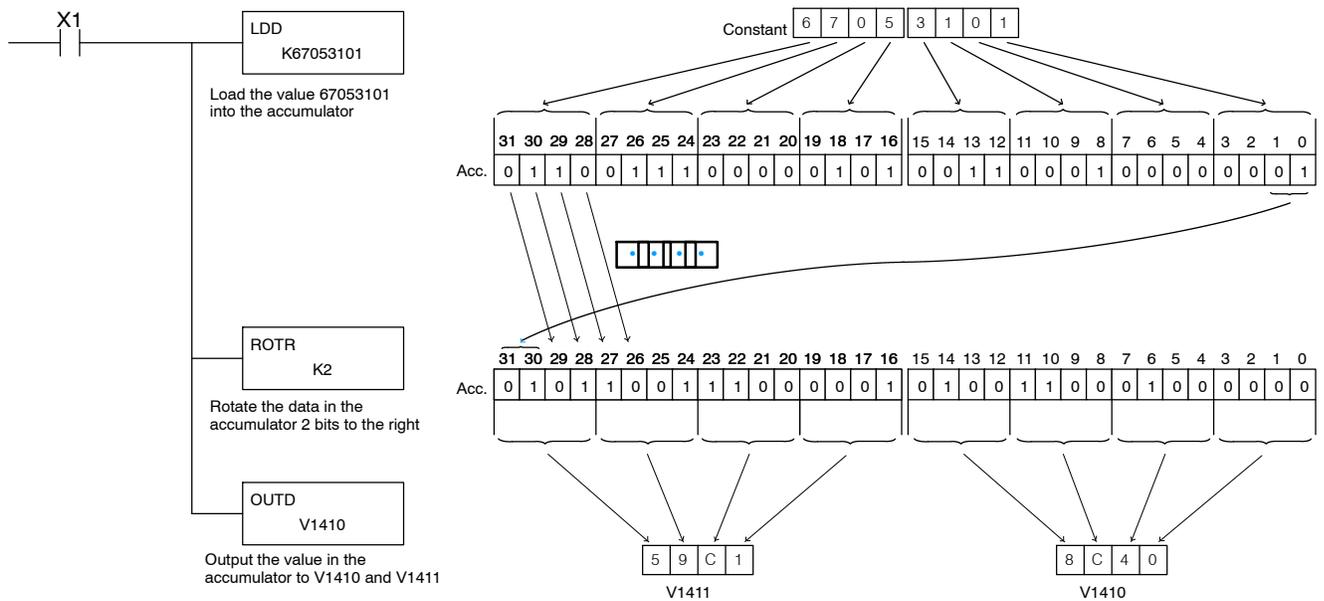
**Using the Accumulator Stack**

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 type instruction is executed without the use of the Out type 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.
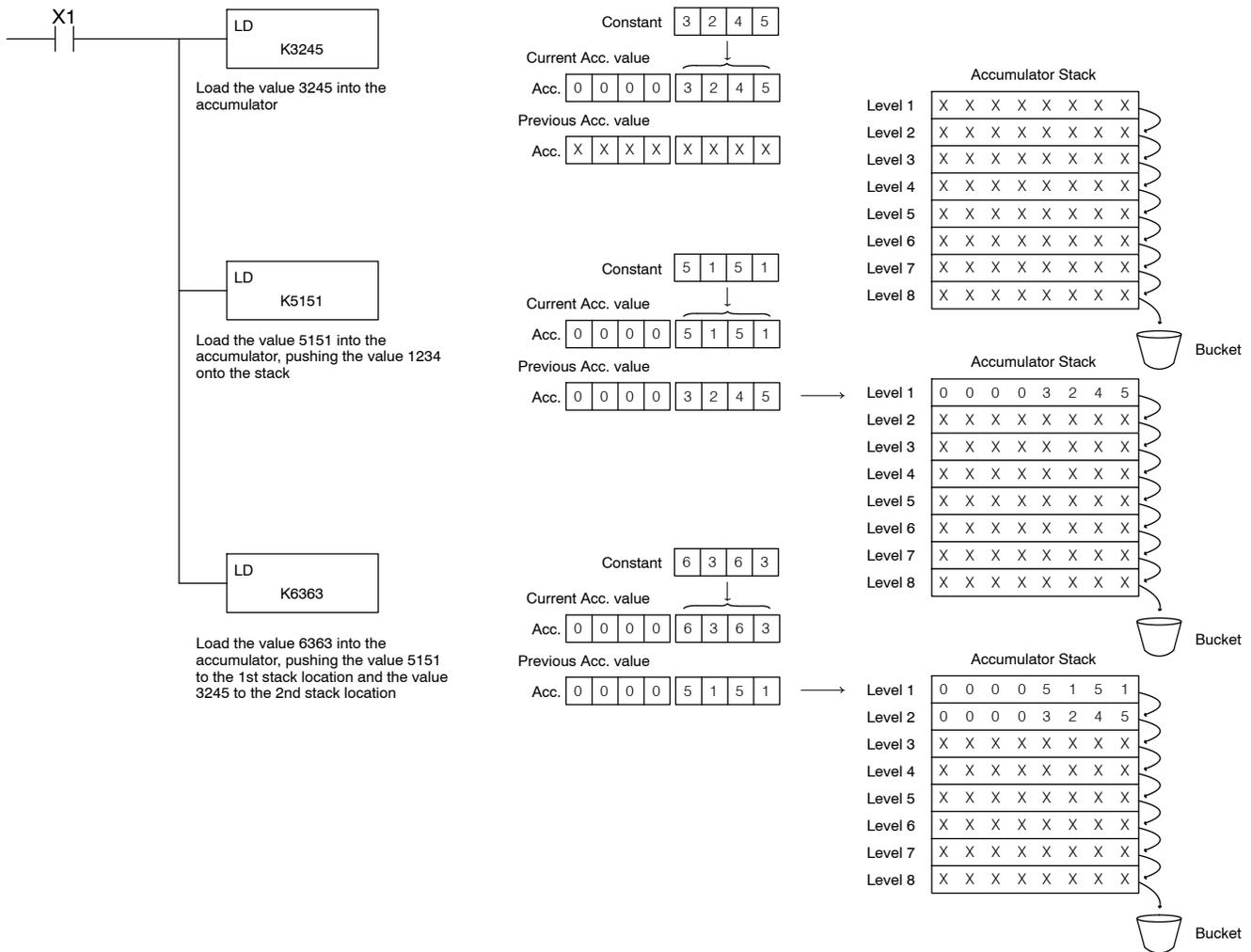


The POP instruction rotates values upward through the stack into the accumulator. When a POP is executed the value which was in the accumulator is cleared and the value that was on top of the stack is in the accumulator. The values in the stack are shifted up one position in the stack.

X1

POP

POP the 1st value on the stack into the accumulator and move stack values up one location

OUT
V1400

Copy data from the accumulator to V1400
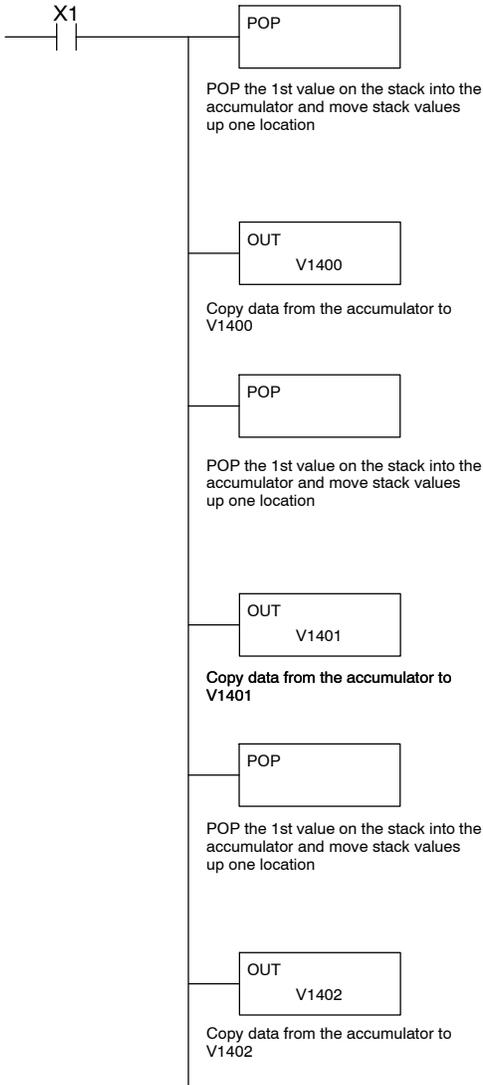
Previous Acc. value

Acc. | X | X | X | X | X | X | X | X |

Current Acc. value

Acc. | 0 | 0 | 0 | 0 | 4 | 5 | 4 | 5 |  ← Level 1

V1400 | 4 | 5 | 4 | 5 |

Accumulator Stack

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Level 1 | 0 | 0 | 0 | 0 | 3 | 7 | 9 | 2 |
| Level 2 | 0 | 0 | 0 | 0 | 7 | 9 | 3 | 0 |
| Level 3 | X | X | X | X | X | X | X | X |
| Level 4 | X | X | X | X | X | X | X | X |
| Level 5 | X | X | X | X | X | X | X | X |
| Level 6 | X | X | X | X | X | X | X | X |
| Level 7 | X | X | X | X | X | X | X | X |
| Level 8 | X | X | X | X | X | X | X | X |

POP

POP the 1st value on the stack into the accumulator and move stack values up one location

OUT
V1401

**Copy data from the accumulator to V1401**

Previous Acc. value

Acc. | 0 | 0 | 0 | 0 | 4 | 5 | 4 | 5 |

Current Acc. value

Acc. | 0 | 0 | 0 | 0 | 3 | 7 | 9 | 2 |  ← Level 1

V1400 | 3 | 7 | 9 | 2 |

Accumulator Stack

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Level 1 | 0 | 0 | 0 | 0 | 7 | 9 | 3 | 0 |
| Level 2 | X | X | X | X | X | X | X | X |
| Level 3 | X | X | X | X | X | X | X | X |
| Level 4 | X | X | X | X | X | X | X | X |
| Level 5 | X | X | X | X | X | X | X | X |
| Level 6 | X | X | X | X | X | X | X | X |
| Level 7 | X | X | X | X | X | X | X | X |
| Level 8 | X | X | X | X | X | X | X | X |

POP

POP the 1st value on the stack into the accumulator and move stack values up one location

OUT
V1402

Copy data from the accumulator to V1402

Previous Acc. value

Acc. | 0 | 0 | 0 | 0 | 3 | 7 | 9 | 2 |

Current Acc. value

Acc. | X | X | X | X | 7 | 9 | 3 | 0 |  ← Level 1

V1400 | 7 | 9 | 3 | 0 |

Accumulator Stack

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Level 1 | X | X | X | X | X | X | X | X |
| Level 2 | X | X | X | X | X | X | X | X |
| Level 3 | X | X | X | X | X | X | X | X |
| Level 4 | X | X | X | X | X | X | X | X |
| Level 5 | X | X | X | X | X | X | X | X |
| Level 6 | X | X | X | X | X | X | X | X |
| Level 7 | X | X | X | X | X | X | X | X |
| Level 8 | X | X | X | X | X | X | X | X |

**Accumulator and Accumulator Stack Memory Locations**

| X | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | X |
|---|---|
| **DS** | **HPP** |

There may be times when you want to read a value that has been placed onto the accumulator stack without having to pop the stack first. Both the accumulator and the accumulator stack have corresponding V-memory locations that can be accessed by the program.

You cannot write to these locations, but you can read them or use them in comparative boolean instructions, etc.

Accumulator

| 0 | 0 | 0 | 0 | 3 | 7 | 9 | 2 |
|---|---|---|---|---|---|---|---|
| V701 | | | | V700 | | | |

Accumulator Stack

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Level 1 | 0 | 0 | 0 | 0 | 3 | 7 | 9 | 2 | V703 – V702 |
| Level 2 | 0 | 0 | 0 | 0 | 7 | 9 | 3 | 0 | V705 – V704 |
| Level 3 | X | X | X | X | X | X | X | X | V707 – V706 |
| Level 4 | X | X | X | X | X | X | X | X | V711 – V710 |
| Level 5 | X | X | X | X | X | X | X | X | V713 – V712 |
| Level 6 | X | X | X | X | X | X | X | X | V715 – V714 |
| Level 7 | X | X | X | X | X | X | X | X | V717 – V716 |
| Level 8 | X | X | X | X | X | X | X | X | V721 – V720 |

Standard
RLL Instructions

**Using Pointers**

Many of the DL405 series instructions will allow V–memory pointers as a operand. Pointers can be useful in ladder logic programming, but can be difficult to understand or implement in your application if you do not have prior experience with pointers (commonly known as indirect addressing). Pointers allow instructions to obtain data from V–memory locations referenced by the pointer value.

**NOTE:** In the DL405 V-memory addressing is in octal. However the value in the pointer location which will reference a V-memory location is viewed as HEX. Use the Load Address instruction to move a address into the pointer location. This instruction performs the Octal to Hexadecimal conversion for you.

The following example uses a pointer operand in a Load instruction. V-memory location 1400 is the pointer location. V1400 contains the value 340 which is the HEX equivalent of the Octal address V-memory location V1500. The CPU copies the data from V1500 (contains 2635) into the lower word of the accumulator.

X1
LD
  P1400

V1400 (P1400) contains the value 340
Hex. 340 Hex. = 1500 Octal which
contains the value 2635.

OUT
  V1600

Copy the data from the lower 16 bits of
the accumulator to V1600.

V1400 | 0 | 3 | 4 | 0

| | | | | |
|---|---|---|---|---|
| V1476 | X | X | X | X |
| V1477 | X | X | X | X |
| V1500 | 2 | 6 | 3 | 5 |
| V1501 | X | X | X | X |
| V1502 | X | X | X | X |
| V1503 | X | X | X | X |
| V1504 | X | X | X | X |
| V1505 | X | X | X | X |
| V1600 | 2 | 6 | 3 | 5 |
| V1601 | X | X | X | X |

Accumulator
2 | 6 | 3 | 5
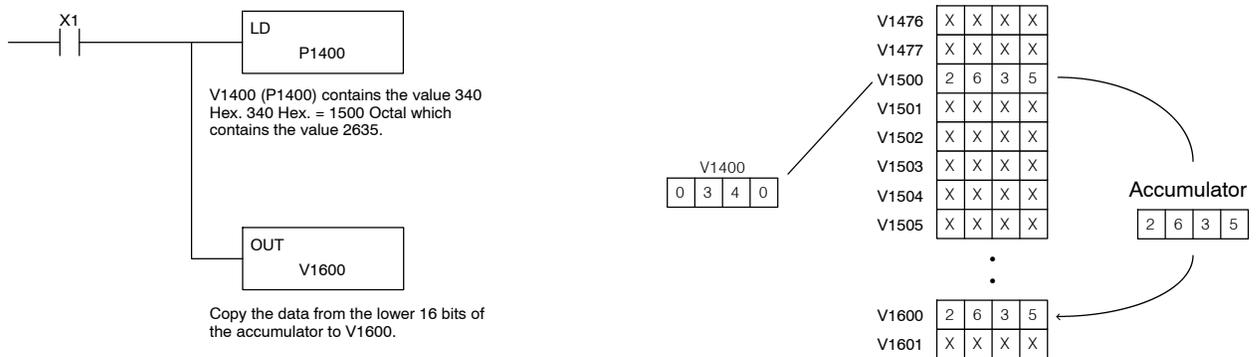
The following example is similar to the one above, except for the LDA (load address) instruction which automatically converts the Octal address to the Hex equivalent.

X1
LDA
  O 1500

Load the lower 16 bits of the accumulator with Hexadecimal equivalent to Octal 1500 (340)

OUT
  V 1400

Copy the data from the lower 16 bits of the accumulator to V1400

LD
  P 1400

V1400 (P1400) contains the value 340 HEX 340 HEX. = 1500 Octal which contains the value 2635

OUT
  V 1600

Copy the data from the lower 16 bits of the accumulator to V1600

1 | 5 | 0 | 0

Unused accumulator bits are set to zero

1500 Octal is converted to Hexadecimal 340 and loaded into the accumulator

Acc. | 0 | 0 | 0 | 0 | 0 | 3 | 4 | 0

0 | 3 | 4 | 0
V1400

V1400 | 0 | 3 | 4 | 0

| | | | | |
|---|---|---|---|---|
| V1476 | X | X | X | X |
| V1477 | X | X | X | X |
| V1500 | 2 | 6 | 3 | 5 |
| V1501 | X | X | X | X |
| V1502 | X | X | X | X |
| V1503 | X | X | X | X |
| V1504 | X | X | X | X |
| V1505 | X | X | X | X |
| V1600 | 2 | 6 | 3 | 5 |
| V1601 | X | X | X | X |

Accumulator
0 | 0 | 0 | 0 | 2 | 6 | 3 | 5

**Load
(LD)**

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | | ✓ |
|---|---|---|
| **DS** | | **HPP** |

**Load Double
(LDD)**

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | | ✓ |
|---|---|---|
| **DS** | | **HPP** |

The Load instruction is a 16 bit instruction that loads the value (Aaaa) (either a V-memory location or a 4 digit constant) into the lower 16 accumulator bits. The upper 16 accumulator bits are set to 0.

```
      ┌──────────┐
──────┤ LD       │
      │   A aaa  │
      └──────────┘
```
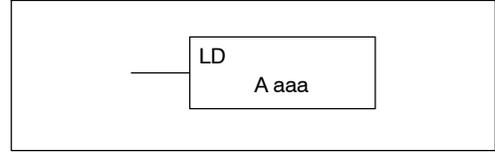
The Load Double instruction is a 32 bit instruction that loads the value (Aaaa), which is either two consecutive V–memory locations or an 8 digit constant value, into the accumulator.

```
      ┌──────────┐
──────┤ LDD      │
      │   A aaa  │
      └──────────┘
```

| Operand Data Type | | DL430 Range | DL440 Range | DL440 Range |
|---|---|---|---|---|
| | **A** | **aaa** | **aaa** | **aaa** |
| V-memory | V | All (See p. 3-40) | All (See p. 3-41) | All (See p. 3-42) |
| Pointer | P | All V mem (See p. 3-40) | All V mem (See p. 3-41) | All V mem (See p. 3-42) |
| Constant | K | 0–FFFF | 0–FFFF | 0–FFFF |

| Discrete Bit Flags | Description |
|---|---|
| SP76 | on when the value loaded into the accumulator by any instruction is zero. |

**NOTE:** Two consecutive Load or Load Double instructions will place the value of the first load instruction onto the accumulator stack.

In the following Load example, when X1 is on, the value in V1400 will be loaded into the accumulator and output to V1500.

*DirectSOFT*

```
  X1        ┌──────────┐
──┤ ├───────┤ LD       │
            │   V1400  │
            └──────────┘
```
Load the value in V1400 into the lower 16 bits of the accumulator

```
            ┌──────────┐
         ───┤ OUT      │
            │   V1500  │
            └──────────┘
```
Copy the value in the lower 16 bits of the accumulator to V1500

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
| LD | V | 1 | 4 | 0 | 0 | ↵ |
| OUT | V | 1 | 5 | 0 | 0 | ↵ |

V1400

| 8 | 9 | 3 | 5 |

The unused accumulator bits are set to zero

Acc. | 0 | 0 | 0 | 0 | 8 | 9 | 3 | 5 |

| 8 | 9 | 3 | 5 |

V1500

In the following example, when X1 is on, the 32 bit value in V1400 and V1401 will be loaded into the accumulator and output to V1500 and V1501.

*DirectSOFT*

```
  X1        ┌──────────┐
──┤ ├───────┤ LDD      │
            │   V1400  │
            └──────────┘
```
Load the value in V1400 and V1401 into the 32 bit accumulator

```
            ┌──────────┐
         ───┤ OUTD     │
            │   V1500  │
            └──────────┘
```
Copy the value in the 32 bit accumulator to V1500 and V1501

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
| LD | SHFT | D | SHFT | V | 1 | 4 | 0 | 0 | ↵ |
| OUT | SHFT | D | SHFT | V | 1 | 5 | 0 | 0 | ↵ |

| V1401 | | | | V1400 | | | |
| 6 | 7 | 3 | 9 | 5 | 0 | 2 | 6 |

Acc. | 6 | 7 | 3 | 9 | 5 | 0 | 2 | 6 |

| 6 | 7 | 3 | 9 | 5 | 0 | 2 | 6 |
| V1501 | | | | V1500 | | | |

Standard
RLL Instructions

## Load Formatted (LDF)

| × | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

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.

```
         LDF      A aaa
                  K bbb
```

| Operand Data Type | | DL440 Range | | DL450 Range | |
|---|---|---|---|---|---|
| | **A** | **aaa** | **bbb** | **aaa** | **bbb** |
| Inputs | X | 0–477 | -- | 0–1777 | -- |
| Outputs | Y | 0–477 | -- | 0–1777 | -- |
| Control Relays | C | 0–1777 | -- | 0–3777 | -- |
| Stage Bits | S | 0–1777 | -- | 0–1777 | -- |
| Timer Bits | T | 0–377 | -- | 0–377 | -- |
| Counter Bits | CT | 0–177 | -- | 0–377 | -- |
| Special Relays | SP | 0–137 320–717 | -- | 0–137 320–717 | -- |
| Global I/O | GX | 0–1777 | -- | 0–2777 | -- |
| Constant | K | -- | 1–32 | -- | 1–32 |

| Discrete Bit Flags | Description |
|---|---|
| SP76 | on when the value loaded into the accumulator by any instruction is zero. |

**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 6 bits of the accumulator are output to Y20–Y26 using the Out Formatted instruction.

*DirectSOFT*

```
  C0
--| |----+-----[ LDF      C10 ]
         |            K7
```
Load the status of 7 consecutive bits (C10–C16) into the accumulator

| Location | Constant |
|---|---|
| C10 | K7 |

| C16 | C15 | C14 | C13 | C12 | C11 | C10 |
|---|---|---|---|---|---|---|
| OFF | OFF | OFF | ON | ON | ON | OFF |

The unused accumulator bits are set to zero

| 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 | 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 |
|---|---|
| Acc. 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 |

```
         |
         +-----[ OUTF     Y20 ]
                     K7
```
Copy the value from the specified number of bits in the accumulator to Y20–Y26

| Location | Constant |
|---|---|
| Y20 | K7 |

| Y26 | Y25 | Y24 | Y23 | Y22 | Y21 | Y20 |
|---|---|---|---|---|---|---|
| OFF | OFF | OFF | ON | ON | ON | OFF |

Handheld Programmer Keystrokes

| STR | C(CR) | 0 | ↵ | | | | | |
|---|---|---|---|---|---|---|---|---|
| LD | SHFT | F | SHFT | C(CR) | 1 | 0 | K(CON) | 7 | ↵ |
| OUT | SHFT | F | SHFT | Y(OUT) | 2 | 0 | K(CON) | 7 | ↵ |

Standard RLL Instructions

## Load Address (LDA)

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |
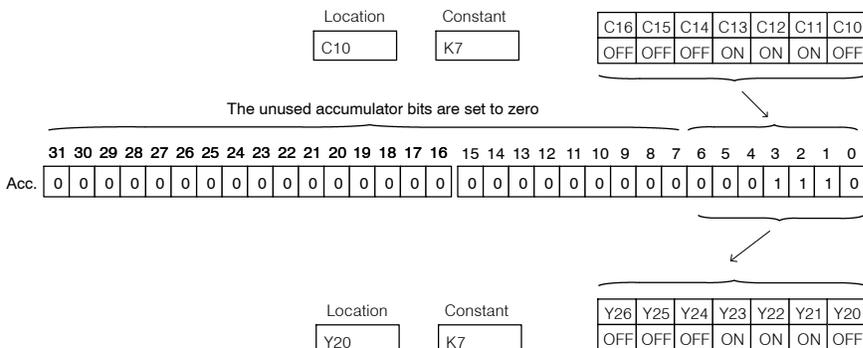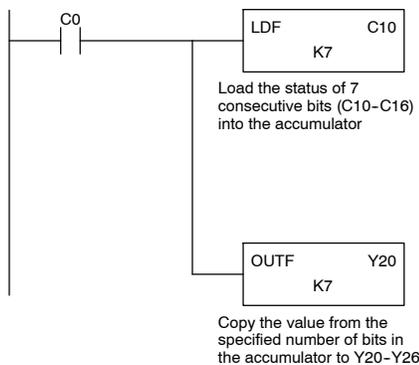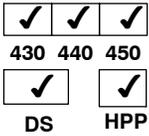
The Load Address instruction is a 16 bit instruction. It converts any octal value or address to the HEX equivalent value and loads the HEX value into the accumulator. This instruction is useful when an address parameter is required since all addresses for the DL405 system are in octal.

```
        LDA
           O aaa
```
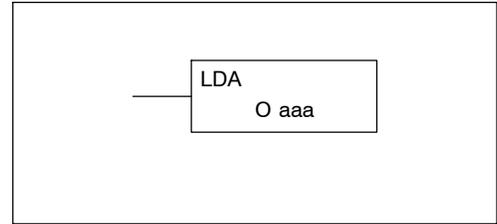
| Operand Data Type | | DL430 Range | DL440 Range | DL450 Range |
|---|---|---|---|---|
| | | **aaa** | **aaa** | **aaa** |
| Octal Address | O | 0 – 77777 | 0 – 77777 | 0 – 177777 |

| Discrete Bit Flags | Description |
|---|---|
| SP76 | on when the value loaded into the accumulator by any instruction is zero. |

**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 V1440 using the Out instruction.



*DirectSOFT*

Load The HEX equivalent to the octal number into the lower 16 bits of the accumulator

Copy the value in lower 16 bits of the accumulator to V1440

The unused accumulator bits are set to zero

Handheld Programmer Keystrokes

Standard
RLL Instructions
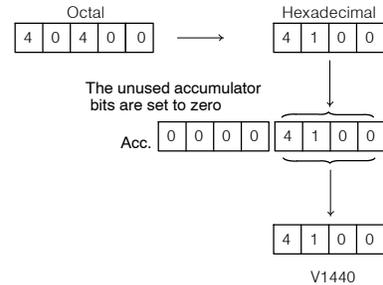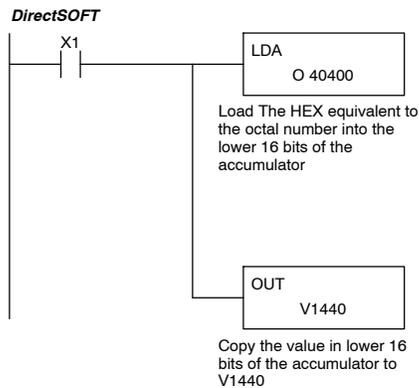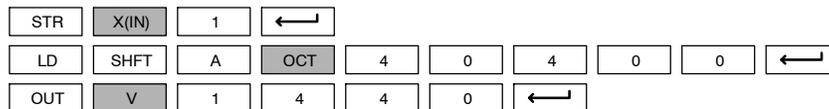
## Load Accumulator Indexed (LDX)

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

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.

```
        ┌──────────┐
        │ LDX      │
────────┤   A aaa  │
        │          │
        └──────────┘
```
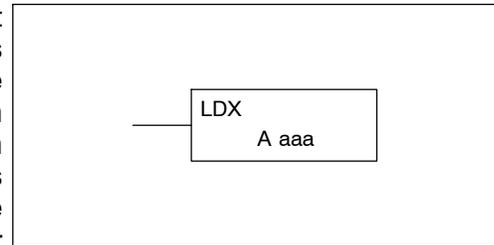
Helpful Hint: — The Load Address instruction can be used to convert an octal address to a HEX address and load the value into the accumulator.

| Operand Data Type | | DL430 Range | DL440 Range | DL440 Range |
|---|---|---|---|---|
| | **A** | **aaa** | **aaa** | **aaa** |
| V–memory | V | All (See p. 3–40) | All (See p. 3–41) | All (See p. 3–42) |
| Pointer | P | -- | All (See p. 3–41) | All (See p. 3–42) |

| Discrete Bit Flags | Description |
|---|---|
| SP76 | on when the value loaded into the accumulator by any instruction is zero. |

**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 1st. 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.



```
   X1            ┌───────────┐
───┤ ├──────────┤ LDA       │
                │    O 25   │
                └───────────┘
    Load The HEX equivalent to
    octal 25 into the lower 16
    bits of the accumulator

                ┌───────────┐
        ────────┤ LDX       │
                │   V1410   │
                └───────────┘
    Move the offset to the stack.
    Load the accumulator with
    the address to be offset

                ┌───────────┐
        ────────┤ OUT       │
                │   V1500   │
                └───────────┘
    Copy the value in the lower
    16 bits of the accumulator
    to V1500
```

Octal → Hexadecimal

| 2 | 5 | | 0 | 0 | 1 | 5 |

The unused accumulator bits are set to zero

Acc. | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 5 |

HEX Value in 1st stack location

Octal
V | 1 | 4 | 1 | 0 | + | 1 | 5 | = V | 1 | 4 | 3 | 5 |

The unused accumulator bits are set to zero

Acc. | 0 | 0 | 0 | 0 | 2 | 3 | 4 | 5 |

The value in V1435 is 2345

| 2 | 3 | 4 | 5 |

V1500

Accumulator Stack

| Level 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 5 |
| Level 2 | X | X | X | X | X | X | X | X |
| Level 3 | X | X | X | X | X | X | X | X |
| Level 4 | X | X | X | X | X | X | X | X |
| Level 5 | X | X | X | X | X | X | X | X |
| Level 6 | X | X | X | X | X | X | X | X |
| Level 7 | X | X | X | X | X | X | X | X |
| Level 8 | X | X | X | X | X | X | X | X |

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
| LD | SHFT | A | OCT | 2 | 5 | ↵ |
| LD | SHFT | X | SHFT | V | 1 | 4 | 1 | 0 | ↵ |
| OUT | V | 1 | 5 | 0 | 0 | ↵ |

Standard
RLL Instructions

## Load Accumulator Indexed from Data Constants (LDSX)

| × | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | | ✓ |
|---|---|---|
| **DS** | | **HPP** |
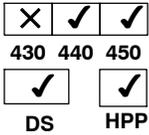
The Load Accumulator Indexed from Data Constants is a 16 bit instruction. The instruction specifies a Data Label Area (DLBL) where numerical or ASCII constants are stored. This value will be loaded into accumulator's lower 16 bits.

```
        ┌─────────────┐
────────┤ LDSX        │
        │     K aaa   │
        └─────────────┘
```
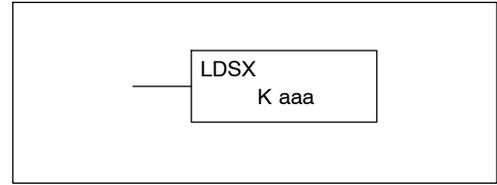
The LDSX instruction uses the value in the first level of the accumulator stack as an offset to determine which numerical or ASCII constant within the Data Label Area will be loaded into the accumulator. The LDSX instruction interprets the value in the first level of the accumulator stack as a HEX value.

Helpful Hint: — The Load Address instruction can be used to convert octal to HEX and load the value into the accumulator.

| Operand Data Type | | DL440 Range | DL450 Range |
|---|---|---|---|
| | | **aaa** | **aaa** |
| Constant | K | 1–FFFF | 1–FFFF |

**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 Load instruction loads the offset of 1 into the accumulator. When the LDSX instruction executes, this value is placed into the first level of the accumulator stack. The LDSX instruction specifies the Data Label (DLBL K2) where the numerical constant(s) are located in the program. It loads the constant value according to the offset value into the accumulator's lower 16 bits.



X1

LD
K1

Load the offset value of 1 (K1) into the lower 16 bits of the accumulator.

LDSX
K2

Move the offset to the stack. Load the accumulator with the data label number

OUT
V1500

Copy the value in the lower 16 bits of the accumulator to V1500

( END )

DLBL       K2

NCON
K3333        Offset 0

NCON
K2323        Offset 1

NCON
K4549        Offset 2

Hexadecimal
| 0 | 0 | 0 | 1 |

The unused accumulator bits are set to zero

Acc. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

Value in 1st. level of stack is used as offset. The value is 1

Accumulator Stack

| Level 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| Level 2 | X | X | X | X | X | X | X | X |
| Level 3 | X | X | X | X | X | X | X | X |
| Level 4 | X | X | X | X | X | X | X | X |
| Level 5 | X | X | X | X | X | X | X | X |
| Level 6 | X | X | X | X | X | X | X | X |
| Level 7 | X | X | X | X | X | X | X | X |
| Level 8 | X | X | X | X | X | X | X | X |

Constant
K | 0 | 0 | 0 | 2 |

The unused accumulator bits are set to zero

Acc. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 |

The unused accumulator bits are set to zero

Acc. | 0 | 0 | 0 | 0 | 2 | 3 | 2 | 3 |

| 2 | 3 | 2 | 3 |

V1500

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
| LD | K(CON) | 1 | ↵ |
| LD | SHFT | S | X | K(CON) | 2 | ↵ |
| OUT | V | 1 | 5 | 0 | 0 | ↵ |
| END | ↵ |

| D | L | B | L | K(CON) | 2 | ↵ |
| SHFT | N | C | O | N | SHFT | K(CON) | 2 | 2 | 2 | 2 | ↵ |
| SHFT | N | C | O | N | SHFT | K(CON) | 2 | 3 | 2 | 3 | ↵ |
| SHFT | N | C | O | N | SHFT | K(CON) | 4 | 4 | 4 | 4 | ↵ |

## Load Real Number (LDR)

| ✕ | ✕ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✕ |
|---|---|
| **DS** | **HPP** |

The Load Real Number instruction loads a real number contained in two consecutive V-memory locations, or an 8-digit constant into the accumulator.

```
        LDR
        A aaa
```
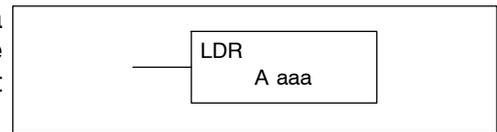
| Operand Data Type | | DL450 Range |
|---|---|---|
| | **A** | **aaa** |
| V–memory | V | All V mem (See p. 3–42) |
| Pointer | P | All V mem (See p. 3–42) |
| Real Constant | R | Full IEEE 32-bit range |

*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
```
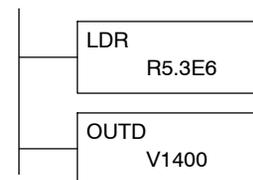
These real numbers are in the IEEE 32-bit floating point format, so they occupy two V-memory locations, *regardless of how big or small the number may be!* If you view a stored real number in hex, binary, or even BCD, the number shown will be very difficult to decipher. Just like all other number types, 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)

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |
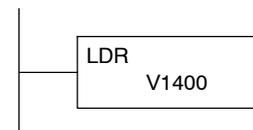
The Out instruction is a 16 bit instruction that copies the value in the lower 16 bits of the accumulator to a specified V-memory location (Aaaa).

```
        ┌──────────┐
────────┤ OUT      │
        │    A aaa │
        └──────────┘
```

| Operand Data Type | | DL430 Range | DL440 Range | DL450 Range |
|---|---|---|---|---|
| | **A** | **aaa** | **aaa** | **aaa** |
| V-memory | V | All (See p. 3-40) | All (See p. 3-41) | All (See p. 3-42) |
| Pointer | P | All (See p. 3-40) | All (See p. 3-41) | All (See p. 3-42) |

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 Load instruction. The value in the lower 16 bits of the accumulator is copied to V1500 using the Out instruction.

*DirectSOFT*

```
    X1            ┌──────────┐     Load the value in V1400 into
────┤ ├───────────┤ LD       │     the lower 16 bits of the
    │             │    V1400 │     accumulator
    │             └──────────┘
    │             ┌──────────┐     Copy the value in the lower
    └─────────────┤ OUT      │     16 bits of the accumulator to
                  │    V1500 │     V1500
                  └──────────┘
```

V1400

| 8 | 9 | 3 | 5 |
|---|---|---|---|

The unused accumulator bits are set to zero

Acc.

| 0 | 0 | 0 | 0 | 8 | 9 | 3 | 5 |
|---|---|---|---|---|---|---|---|

| 8 | 9 | 3 | 5 |
|---|---|---|---|

V1500

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
|---|---|---|---|
| LD | V | 1 | 4 | 0 | 0 | ↵ |
| OUT | V | 1 | 5 | 0 | 0 | ↵ |

## Out DOUBLE
## (OUTD)

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

The Out Double instruction is a 32 bit instruction that copies the value in the accumulator to two consecutive V-memory locations at a Specified starting location (Aaaa).
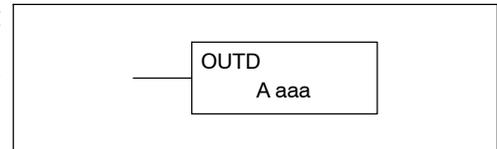
```
        ┌──────────┐
────────┤ OUTD     │
        │    A aaa │
        └──────────┘
```

| Operand Data Type | | DL430 Range | DL440 Range | DL450 Range |
|---|---|---|---|---|
| | **A** | **aaa** | **aaa** | **aaa** |
| V-memory | V | All (See p. 3-40) | All (See p. 3-41) | All (See p. 3-42) |
| Pointer | P | All V mem (See p. 3-40) | All V mem (See p. 3-41) | All V mem (See p. 3-42) |

In the following example, when X1 is on, the 32 bit value in V1400 and V1401 will be loaded into the accumulator using the Load Double instruction. The value in the accumulator is output to V1500 and V1501 using the Out Double instruction.

*DirectSOFT*

```
    X1            ┌──────────┐     Load the value in V1400 and
────┤ ├───────────┤ LDD      │     V1401 into the accumulator
    │             │    V1400 │
    │             └──────────┘
    │             ┌──────────┐     Copy the value in the
    └─────────────┤ OUTD     │     accumulator to V1500 and
                  │    V1500 │     V1501
                  └──────────┘
```

V1401      V1400

| 6 | 7 | 3 | 9 | 5 | 0 | 2 | 6 |
|---|---|---|---|---|---|---|---|

Acc.

| 6 | 7 | 3 | 9 | 5 | 0 | 2 | 6 |
|---|---|---|---|---|---|---|---|

| 6 | 7 | 3 | 9 | 5 | 0 | 2 | 6 |
|---|---|---|---|---|---|---|---|

V1501      V1500

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
|---|---|---|---|
| LD | SHFT | D | SHFT | V | 1 | 4 | 0 | 0 | ↵ |
| OUT | SHFT | D | SHFT | V | 1 | 5 | 0 | 0 | ↵ |

Standard
RLL Instructions
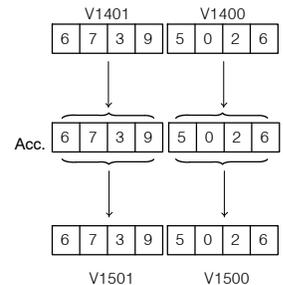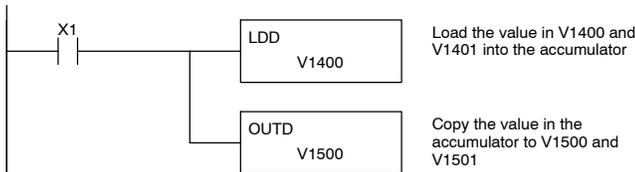
**Out
Formatted
(OUTF)**

| × | ✓ | ✓ |
|---|---|---|

**430 440 450**

| ✓ | | ✓ |
|---|---|---|
| **DS** | | **HPP** |

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.
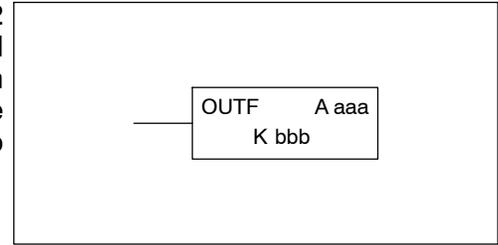
```
        OUTF      A aaa
                  K bbb
```

| Operand Data Type | | DL440 Range | | DL450 Range | |
|---|---|---|---|---|---|
| | **A** | **aaa** | **bbb** | **aaa** | **bbb** |
| Inputs | X | 0–477 | –– | 0–1777 | –– |
| Outputs | Y | 0–477 | –– | 0–1777 | –– |
| Control Relays | C | 0–1777 | –– | 0–3777 | –– |
| Global I/O | GX | 0–1777 | –– | 0–2777 | –– |
| Constant | K | –– | 1–32 | –– | 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 Y20–Y26 using the Out Formatted instruction.

*DirectSOFT*

```
   C0              LDF        C10
───┤ ├──────┬────────        K7
            │
            │     Load the status of 7
            │     consecutive bits (C10–C16)
            │     into the accumulator
            │
            │
            │     OUTF        Y20
            └────────        K7

                  Copy the value of the
                  specified number of bits
                  from the accumulator to
                  Y20–Y26
```

Location: C10
Constant: K7

| C16 | C15 | C14 | C13 | C12 | C11 | C10 |
|---|---|---|---|---|---|---|
| OFF | OFF | OFF | ON | ON | ON | OFF |

The unused accumulator bits are set to zero

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |

Accumulator

Location: Y20
Constant: K7

| Y26 | Y25 | Y24 | Y23 | Y22 | Y21 | Y20 |
|---|---|---|---|---|---|---|
| OFF | OFF | OFF | ON | ON | ON | OFF |
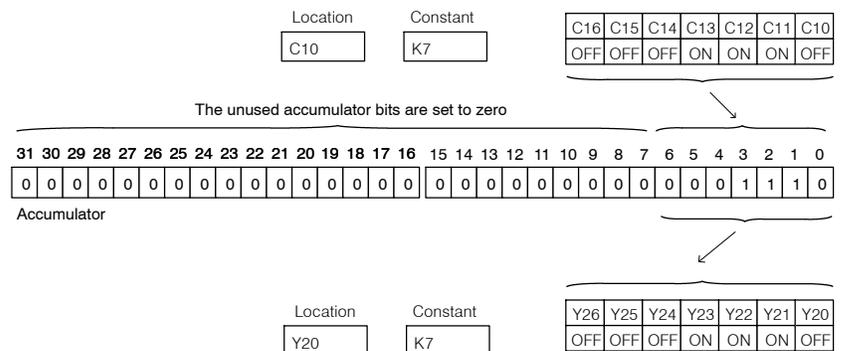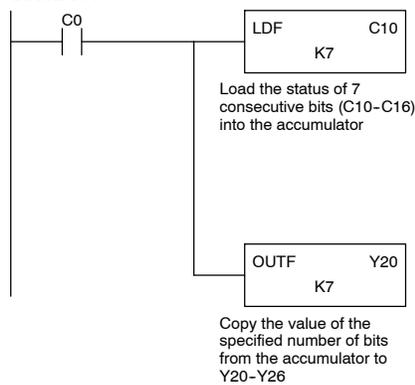
Handheld Programmer Keystrokes

| STR | C(CR) | 0 | ↵ |
|---|---|---|---|

| LD | SHFT | F | SHFT | C(CR) | 1 | 0 | K(CON) | 7 | ↵ |
|---|---|---|---|---|---|---|---|---|---|

| OUT | SHFT | F | SHFT | Y(OUT) | 2 | 0 | K(CON) | 7 | ↵ |
|---|---|---|---|---|---|---|---|---|---|

**Out Indexed
(OUTX)**

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | | ✓ |
|---|---|---|
| **DS** | | **HPP** |

The Out Indexed 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.
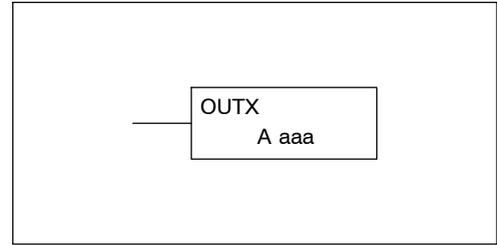
```
      OUTX
       A aaa
```

| Operand Data Type | | DL430 Range | DL440 Range | DL450 Range |
|---|---|---|---|---|
| | **A** | **aaa** | **aaa** | **aaa** |
| V-memory | V | All (See p. 3-40) | All (See p. 3-41) | All (See p. 3-42) |
| Pointer | P | All (See p. 3-40) | All (See p. 3-41) | All (See p. 3-42) |

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 Load Address instruction is executed. Remember, two consecutive Load instructions places the value of the first load instruction onto the stack. The Load Address instruction converts octal 25 to HEX 15 and places the value in the accumulator. The Out Indexed instruction outputs the value 3544 which resides in the first level of the accumulator stack to V1525.

*DirectSOFT*



X1

LD
K3544

Load the accumulator with the value 3544

LDA
O 25

Load The HEX equivalent to octal 25 into the lower 16 bits of the accumulator. This is the offset for the Out Indexed instruction, which determines the final destination address

OUTX
V1500

Copy the value in the first level of the stack to the offset address 1525 (V1500 + 25)

Constant
| 3 | 5 | 4 | 4 |

The unused accumulator bits are set to zero

Acc. | 0 | 0 | 0 | 0 | 3 | 5 | 4 | 4 |

Octal
| 2 | 5 |  →  HEX | 0 | 0 | 1 | 5 |

The unused accumulator bits are set to zero

Acc. | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 5 |

Octal                Octal              Octal
V | 1 | 5 | 0 | 0 | + | 2 | 5 | = V | 1 | 5 | 2 | 5 |

The hex 15 converts to 25 octal, which is added to the base address of V1500 to yield the final destination.

| 3 | 5 | 4 | 4 |

V1525

Accumulator Stack

| Level 1 | 0 | 0 | 0 | 0 | 3 | 5 | 4 | 4 |
|---|---|---|---|---|---|---|---|---|
| Level 2 | X | X | X | X | X | X | X | X |
| Level 3 | X | X | X | X | X | X | X | X |
| Level 4 | X | X | X | X | X | X | X | X |
| Level 5 | X | X | X | X | X | X | X | X |
| Level 6 | X | X | X | X | X | X | X | X |
| Level 7 | X | X | X | X | X | X | X | X |
| Level 8 | X | X | X | X | X | X | X | X |

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ← |
|---|---|---|---|

| LD | K | 3 | 5 | 4 | 4 | ← |
|---|---|---|---|---|---|---|

| LD | SHFT | A | OCT | 2 | 5 | ← |
|---|---|---|---|---|---|---|

| OUT | SHFT | X | SHFT | V | 1 | 5 | 0 | 0 | ← |
|---|---|---|---|---|---|---|---|---|---|

Standard
RLL Instructions
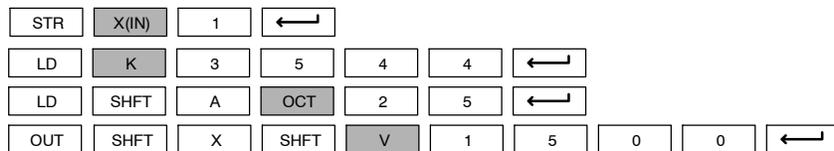
**Out Least
(OUTL)**

| × | × | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

The Out Least 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).
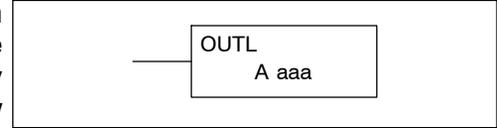
```
        OUTL
        A aaa
```

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 Load instruction. The value in the lower 8 bits of the accumulator are copied to V1500 using the Out Least instruction.

*DirectSOFT*



Load the value in V1400 into the lower 16 bits of the accumulator

Copy the value in the lower 8 bits of the accumulator to V1500.

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
| LD | V | 1 | 4 | 0 | 0 | ↵ |
| OUT | SHFT | L | SHFT | V | 1 | 5 | 0 | 0 | ↵ |

V1400: | 8 | 9 | 3 | 5 |

The unused accumulator bits are set to zero

Acc. | 0 | 0 | 0 | 0 | 8 | 9 | 3 | 5 |

| 0 | 0 | 3 | 5 |
V1500
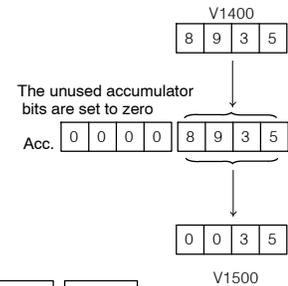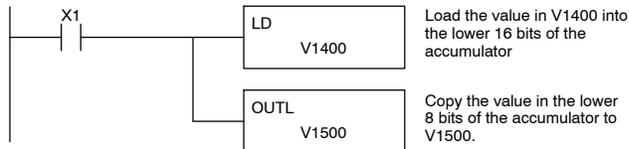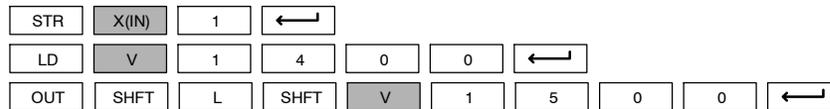
**Out Most
(OUTM)**

| × | × | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

The Out Most 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).

```
        OUTM
        A aaa
```
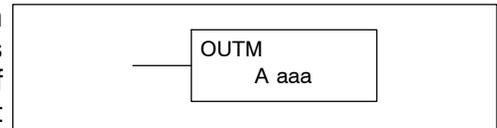
| Operand Data Type | | DL450 Range |
|---|---|---|
| | **A** | **aaa** |
| V-memory | V | All (See p. 3–42) |
| Pointer | P | All (See p. 3–42) |

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 Load instruction. The value in the upper 8 bits of the lower 16 bits of the accumulator is copied to V1500 using the Out Most instruction.

*DirectSOFT*



Load the value in V1400 into the lower 16 bits of the accumulator

Copy the value in the upper 8 bits of the lower 16 bits of the accumulator to V1500.

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
| LD | V | 1 | 4 | 0 | 0 | ↵ |
| OUT | SHFT | M | SHFT | V | 1 | 5 | 0 | 0 | ↵ |

V1400: | 8 | 9 | 3 | 5 |

The unused accumulator bits are set to zero

Acc. | 0 | 0 | 0 | 0 | 8 | 9 | 3 | 5 |

| 8 | 9 | 0 | 0 |
V1500
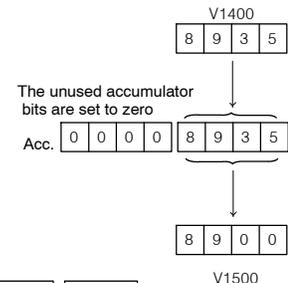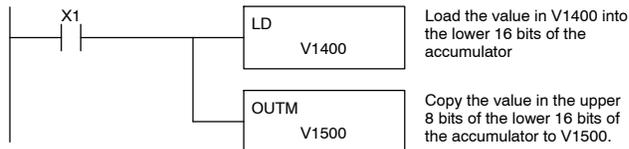
## Pop
(POP)

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

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.

```
        ┌─────────┐
 ───────┤  POP    │
        │         │
        └─────────┘
```
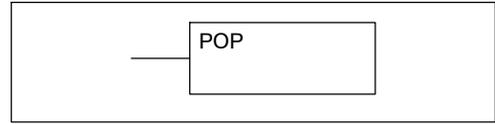
In the example below, when C0 is on the Pop instruction moves the value 4545 currently on top of the stack into the accumulator. The value is output to V1400 using the Out instruction. The next Pop moves the value 3792 into the accumulator and outputs the value to V1401. The last Pop moves the value 7930 into the accumulator and outputs it to V1402. Remember to use Out Double instructions if the value in the stack uses more than 16 bits (4 digits). Each value will occupy two V-memory locations.

| Discrete Bit Flags | Description |
|---|---|
| SP63 | on when the result of the instruction causes the value in the accumulator to be zero. |



**DirectSOFT**

Pop the 1st. value on the stack into the accumulator and move stack values up one location

Copy the value in the lower 16 bits of the accumulator to V1400

Pop the 1st. value on the stack into the accumulator and move stack values up one location

Copy the value in the lower 16 bits of the accumulator to V1401

Pop the 1st. value on the stack into the accumulator and move stack values up one location

Copy the value in the lower 16 bits of the accumulator to V1402

Handheld Programmer Keystrokes

**Standard RLL Instructions**

# Accumulator Logic Instructions

## And
## (AND)

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

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 in the accumulator. The discrete status flag indicates if the result of the And is zero.

```
        ┌─────────────┐
        │ AND         │
────────┤   A aaa     │
        │             │
        └─────────────┘
```

| Operand Data Type | | DL430 Range | DL440 Range | DL450 Range |
|---|---|---|---|---|
| | **A** | **aaa** | **aaa** | **aaa** |
| V–memory | V | All (See p. 3–40) | All (See p. 3–41) | All (See p. 3–42) |
| Pointer | P | – – | All (See p. 3–41) | All (See p. 3–42) |

| Discrete Bit Flags | Description |
|---|---|
| SP63 | Will be on if the result in 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 V1400 will be loaded into the accumulator using the Load instruction. The value in the accumulator is Anted with the value in V1420 using the And instruction. The value in the lower 16 bits of the accumulator is output to V1500 using the Out instruction.

*DirectSOFT*

```
   X1        ┌──────────┐
───┤ ├───────┤ LD       │
             │   V1400  │
             └──────────┘
```
Load the value in V1400 into the lower 16 bits of the accumulator

```
             ┌──────────┐
             │ AND      │
             │   V1420  │
             └──────────┘
```
AND the value in the accumulator with the value in V1420

```
             ┌──────────┐
             │ OUT      │
             │   V1500  │
             └──────────┘
```
Copy the lower 16 bits of the accumulator to V1500

V1400

| 2 | 8 | 7 | A |
|---|---|---|---|

The upper 16 bits of the accumulator will be set to 0

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

Acc. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 |

Acc. 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 1 1 1 0 1 0

6A38
AND (V1420) 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 1 0 0 0 1 1 1 0 0 0

Acc. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |

| 2 | 8 | 3 | 8 |
|---|---|---|---|

V1500

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
|---|---|---|---|

| LD | V | 1 | 4 | 0 | 0 | ↵ |
|---|---|---|---|---|---|---|

| AND | V | 1 | 4 | 2 | 0 | ↵ |
|---|---|---|---|---|---|---|

| OUT | V | 1 | 5 | 0 | 0 | ↵ |
|---|---|---|---|---|---|---|

Standard
RLL Instructions

## And Double (ANDD)

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

The And Double 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 And Double is zero or a negative number (the most significant bit is on).

```
          ANDD
            A aaa
```

| Operand Data Type | | DL430 Range | DL440 Range | DL440 Range |
|---|---|---|---|---|
| | **A** | **aaa** | **aaa** | **aaa** |
| V–memory | V | -- | All (See p. 3–41) | All (See p. 3–42) |
| Pointer | P | -- | All (See p. 3–41) | All (See p. 3–42) |
| Constant | K | 0–FFFF | 0–FFFF | 0–FFFF |

| Discrete Bit Flags | Description |
|---|---|
| SP63 | Will be on if the result in the accumulator is zero |
| SP70 | Will be on is 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 Load Double instruction. The value in the accumulator is Anted with V1420 and V1421 using the And double instruction. The value in the accumulator is output to V1500 and V1501 using the Out Double instruction.

*DirectSOFT*



Load the value in V1400 and V1401 into the accumulator

AND the value in the accumulator with the value in V1420 and V1421

Copy the value in the accumulator to V1500 and V1501
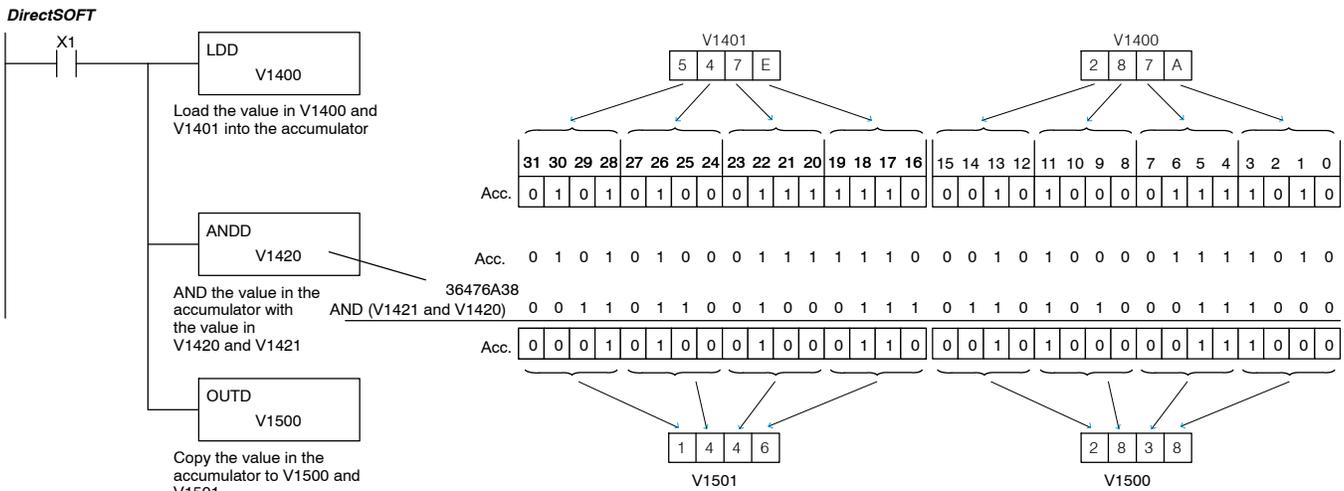
Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
|---|---|---|---|
| LD | SHFT | D | SHFT | V | 1 | 4 | 0 | 0 | ↵ |
| AND | SHFT | D | SHFT | V | 1 | 4 | 2 | 0 | ↵ |
| OUT | SHFT | D | SHFT | V | 1 | 5 | 0 | 0 | ↵ |

**And
Formatted
(ANDF)**

| 430 | 440 | 450 |
|------|------|------|
| ✕ | ✓ | ✓ |

| DS | HPP |
|------|------|
| ✓ | ✓ |

The And Formatted instruction logically ANDs the binary value in the accumulator and 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).

```
        ANDF      A aaa
                  K bbb
```

| Operand Data Type | | DL440 Range | | DL450 Range | |
|---|---|---|---|---|---|
| | A/B | aaa | bbb | aaa | bbb |
| Inputs | X | 0–477 | – – | 0–1777 | – – |
| Outputs | Y | 0–477 | – – | 0–1777 | – – |
| Control Relays | C | 0–1777 | – – | 0–3777 | – – |
| Stage Bits | S | 0–1777 | – – | 0–1777 | – – |
| Timer Bits | T | 0–377 | – – | 0–377 | – – |
| Counter Bits | CT | 0–177 | – – | 0–377 | – – |
| Special Relays | SP | 0–137 320–717 | – – | 0–137 320–717 | – – |
| Global I/O | GX | 0–1777 | – – | 0–2777 | – – |
| Constant | K | – – | 1–32 | – – | 1–32 |

| Discrete Bit Flags | Description |
|---|---|
| SP63 | Will be on if the result in the accumulator is zero |
| SP70 | Will be on is 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 Load Formatted 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 And Formatted instruction. The Out Formatted instruction outputs the accumulator's lower four bits to C20–C23.



*DirectSOFT*

```
X1        LDF          C10
                       K4
```
Load the status of 4 consecutive bits (C10–C13) into the accumulator

```
          ANDF         Y20
                       K4
```
And the binary bit pattern (Y20–Y23) with the value in the accumulator

```
          OUTF         C20
                       K4
```
Copy the value in the lower 4 bits in accumulator to C20–C23

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
| LD | SHFT | F | SHFT | C(CR) | 1 | 0 | K(CON) | 4 | ↵ |
| AND | SHFT | F | SHFT | Y(OUT) | 2 | 0 | K(CON) | 4 | ↵ |
| OUT | SHFT | F | SHFT | C(CR) | 2 | 0 | K(CON) | 4 | ↵ |

**And with Stack (ANDS)**

| × | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

The And with Stack 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 And with Stack is zero or a negative number (the most significant bit is on).
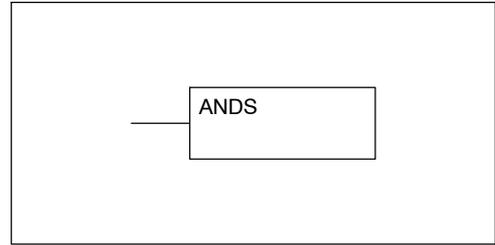
| Discrete Bit Flags | Description |
|---|---|
| SP63 | Will be on if the result in the accumulator is zero |
| SP70 | Will be on is the result in the accumulator is negative |

**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 Anted 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.



*DirectSOFT*

LDD
V1400
Load the value in V1400 and V1401 into the accumulator

ANDS
AND the value in the accumulator with the first level of the accumulator stack

OUTD
V1500
Copy the value in the accumulator to V1500 and V1501

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
| LD | SHFT | D | SHFT | V | 1 | 4 | 0 | 0 | ↵ |
| AND | SHFT | S | ↵ |
| OUT | SHFT | D | SHFT | V | 1 | 5 | 0 | 0 | ↵ |

## Or
## (OR)

✓ ✓ ✓
**430 440 450**
✓ ✓
**DS HPP**
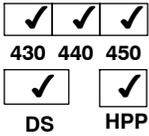
The Or instruction is a 16 bit instruction that logically ors the value in the lower 16 bits of the accumulator with a specified V–memory location (Aaaa). The result resides in the in the accumulator. The discrete status flag indicates if the result of the Or is zero.

```
┌──────────────┐
│ OR           │
│     A aaa    │
└──────────────┘
```

| Operand Data Type | | DL430 Range | DL440 Range | DL440 Range |
|---|---|---|---|---|
| | **A** | **aaa** | **aaa** | **aaa** |
| V-memory | V | All (See p. 3–40) | All (See p. 3–41) | All (See p. 3–42) |
| Pointer | P | – – | All V mem (See p. 3–41) | All V mem (See p. 3–42) |

| Discrete Bit Flags | Description |
|---|---|
| SP63 | Will be on if the result in 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 V1400 will be loaded into the accumulator using the Load instruction. The value in the accumulator is Hoped with V1420 using the Or instruction. The value in the lower 16 bits of the accumulator is output to V1500 using the Out instruction.

*DirectSOFT*



```
X1
─┤ ├──────┌────────────┐
           │ LD         │
           │     V1400  │
           └────────────┘
```
Load the value in V1400 into the lower 16 bits of the accumulator

```
┌────────────┐
│ OR         │
│     V1420  │
└────────────┘
```
Or the value in the accumulator with the value in V1420

```
┌────────────┐
│ OUT        │
│     V1500  │
└────────────┘
```
Copy the value in the lower 16 bits of the accumulator to V1500

The upper 16 bits of the accumulator will be set to 0

V1400
| 2 | 8 | 7 | A |

| 31 30 29 28 | 27 26 25 24 | 23 22 21 20 | 19 18 17 16 | 15 14 13 12 | 11 10 9 8 | 7 6 5 4 | 3 2 1 0 |
|---|---|---|---|---|---|---|---|
Acc. | 0 0 0 0 | 0 0 0 0 | 0 0 0 0 | 0 0 0 0 | 0 0 1 0 | 1 0 0 0 | 0 1 1 1 | 1 0 1 0 |

Acc. 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 1 1 1 0 1 0

6A38
OR (V1420) 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 1 0 0 0 1 1 1 0 0 0

Acc. | 0 0 0 0 | 0 0 0 0 | 0 0 0 0 | 0 0 0 0 | 0 1 1 0 | 1 0 1 0 | 0 1 1 1 | 1 0 1 0 |

| 6 | A | 7 | A |
V1500

**Handheld Programmer Keystrokes**

| STR | X(IN) | 1 | ↵ |
| LD | V | 1 | 4 | 0 | 0 | ↵ |
| OR | V | 1 | 4 | 2 | 0 | ↵ |
| OUT | V | 1 | 5 | 0 | 0 | ↵ |

Standard
RLL Instructions

## Or Double (ORD)

| ✔ | ✔ | ✔ |
|---|---|---|
| **430** | **440** | **450** |

| ✔ | ✔ |
|---|---|
| **DS** | **HPP** |

The OR Double is a 32 bit instruction that 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 Or Double is zero or a negative number (the most significant bit is on).

```
        ┌─────────────┐
────────┤ ORD         │
        │      A aaa   │
        └─────────────┘
```

| Operand Data Type | | DL430 Range | DL440 Range | DL450 Range |
|---|---|---|---|---|
| | **A** | **aaa** | **aaa** | **aaa** |
| V-memory | V | -- | All (See p. 3-41) | All (See p. 3-42) |
| Pointer | P | -- | All (See p. 3-41) | All (See p. 3-42) |
| Constant | K | 0-FFFF | 0-FFFF | 0-FFFF |

| Discrete Bit Flags | Description |
|---|---|
| SP63 | Will be on if the result in the accumulator is zero |
| SP70 | Will be on is 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 Load Double instruction. The value in the accumulator is Hoped with V1420 and V1421 using the Or Double instruction. The value in the accumulator is output to V1500 and V1501 using the Out Double instruction.

**DirectSOFT**

```
  X1           ┌─────────┐
──┤ ├──────────┤ LDD     │
               │   V1400 │
               └─────────┘
               Load the value in V1400 and
               V1401 into accumulator

               ┌─────────┐
               │ ORD     │
               │   V1420 │
               └─────────┘
               OR the value in the
               accumulator with            36476A38
               the value in V1420 and      OR (V1421 and V1420)
               V1421

               ┌─────────┐
               │ OUTD    │
               │   V1500 │
               └─────────┘
               Copy the value in the
               accumulator to V1500 and
               V1501
```

V1401
| 5 | 4 | 7 | E |

V1400
| 2 | 8 | 7 | A |

| 31 30 29 28 | 27 26 25 24 | 23 22 21 20 | 19 18 17 16 | 15 14 13 12 | 11 10 9 8 | 7 6 5 4 | 3 2 1 0 |
|---|---|---|---|---|---|---|---|
Acc. | 0 1 0 1 | 0 1 0 0 | 0 1 1 1 | 1 1 1 0 | 0 0 1 0 | 1 0 0 0 | 0 1 1 1 | 1 0 1 0 |

Acc. 0 1 0 1 0 1 0 0 0 1 1 1 1 1 1 0 0 0 1 0 1 0 0 0 0 1 1 1 1 0 1 0

0 0 1 1 0 1 1 0 0 1 0 0 0 1 1 1 0 1 1 0 1 0 1 0 0 0 1 1 1 0 0 0

Acc. | 0 1 1 1 | 0 1 1 0 | 0 1 1 1 | 1 1 1 1 | 0 1 1 0 | 1 0 1 0 | 0 1 1 1 | 1 0 1 0 |

| 7 | 6 | 7 | F |
V1501

| 6 | A | 7 | A |
V1500

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
|---|---|---|---|

| LD | SHFT | D | SHFT | V | 1 | 4 | 0 | 0 | ↵ |
|---|---|---|---|---|---|---|---|---|---|

| OR | SHFT | D | SHFT | V | 1 | 4 | 2 | 0 | ↵ |
|---|---|---|---|---|---|---|---|---|---|

| OUT | SHFT | D | SHFT | V | 1 | 5 | 0 | 0 | ↵ |
|---|---|---|---|---|---|---|---|---|---|

**Or
Formatted
(ORF)**

| | | |
|---|---|---|
| ✕ | ✓ | ✓ |
| **430** | **440** | **450** |

| | |
|---|---|
| ✓ | ✓ |
| **DS** | **HPP** |

The Or Formatted 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 ORed. Discrete status flags indicate if the result is zero or negative (the most significant bit =1).

```
         ┌─────────────────┐
─────────┤ ORF      A aaa   │
         │          K bbb   │
         └─────────────────┘
```

| Operand Data Type | | DL440 Range | | DL450 Range | |
|---|---|---|---|---|---|
| | A/B | aaa | bbb | aaa | bbb |
| Inputs | X | 0–477 | – – | 0–1777 | – – |
| Outputs | Y | 0–477 | – – | 0–1777 | – – |
| Control Relays | C | 0–1777 | – – | 0–3777 | – – |
| Stage Bits | S | 0–1777 | – – | 0–1777 | – – |
| Timer Bits | T | 0–377 | – – | 0–377 | – – |
| Counter Bits | CT | 0–177 | – – | 0–377 | – – |
| Special Relays | SP | 0–137 320–717 | – – | 0–137 320–717 | – – |
| Global I/O | GX | 0–1777 | – – | 0–2777 | – – |
| Constant | K | – – | 1–32 | – – | 1–32 |

| Discrete Bit Flags | Description |
|---|---|
| SP63 | Will be on if the result in the accumulator is zero |
| SP70 | Will be on is 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 Load Formatted instruction loads C10–C13 (4 binary bits) into the accumulator. The Or Formatted instruction logically ORs the accumulator contents with Y20–Y23 bit pattern. The Out Formatted instruction outputs the accumulator's lower four bits to C20–C23.

*DirectSOFT*

```
  X1          ┌─────────────┐
──┤ ├─────┬───┤ LDF     C10  │
          │   │         K4   │
          │   └─────────────┘
          │   Load the status of 4
          │   consecutive bits (C10-C13)
          │   into the accumulator
          │   ┌─────────────┐
          ├───┤ ORF     Y20  │
          │   │         K4   │
          │   └─────────────┘
          │   Or the binary bit pattern
          │   (Y20–Y23) with the value in
          │   the accumulator
          │   ┌─────────────┐
          └───┤ OUTF    C20  │
              │         K4   │
              └─────────────┘
              Copy the specified number
              of bits from the accumulator
              to C20-C23
```

Location: C10  Constant: K4

| C13 | C12 | C11 | C10 |
|---|---|---|---|
| OFF | ON | ON | OFF |

The unused accumulator bits are set to zero

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Acc. 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0

OR (Y20–Y23)

| Y23 | Y22 | Y21 | Y20 |
|---|---|---|---|
| ON | OFF | OFF | OFF |

1 0 0 0

Acc. 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0

Location: C20  Constant: K4

| C23 | C22 | C21 | C20 |
|---|---|---|---|
| ON | ON | ON | OFF |

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ | | | | | |
|---|---|---|---|---|---|---|---|---|
| LD | SHFT | F | SHFT | C(CR) | 1 | 0 | K(CON) | 4 | ↵ |
| OR | SHFT | F | SHFT | Y(OUT) | 2 | 0 | K(CON) | 4 | ↵ |
| OUT | SHFT | F | SHFT | C(CR) | 2 | 0 | K(CON) | 4 | ↵ |

Standard
RLL Instructions

## Or with Stack (ORS)

| | | |
|---|---|---|
| ✗ | ✓ | ✓ |
| **430** | **440** | **450** |
| ✓ | | ✓ |
| **DS** | | **HPP** |

The Or with Stack 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 Or with Stack is zero or a negative number (the most significant bit is on).
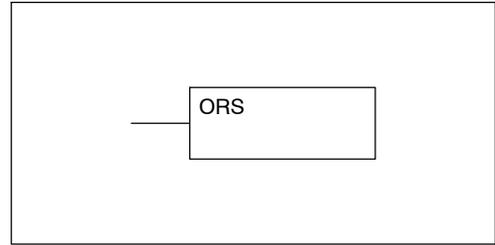


| Discrete Bit Flags | Description |
|---|---|
| SP63 | Will be on if the result in the accumulator is zero |
| SP70 | Will be on is the result in the accumulator is negative |

In the following example when X1 is on, the binary value in the accumulator will be Hoped with the binary value in the first level of the stack. The result resides in the accumulator.
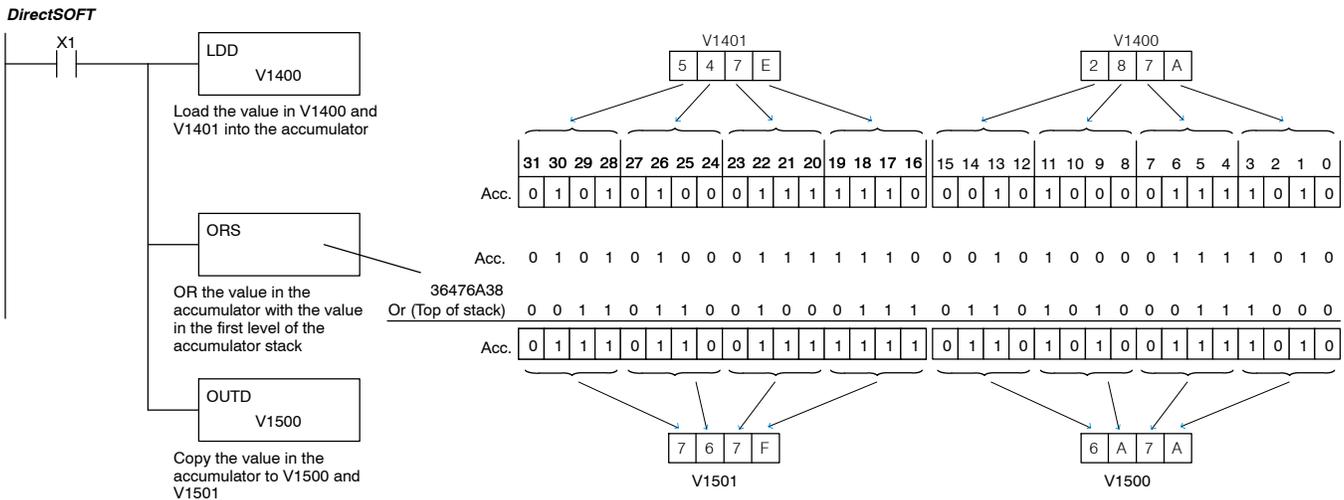
*DirectSOFT*



Load the value in V1400 and V1401 into the accumulator

OR the value in the accumulator with the value in the first level of the accumulator stack

Copy the value in the accumulator to V1500 and V1501

Handheld Programmer Keystrokes

## Exclusive Or (XOR)

✓ ✓ ✓
**430 440 450**
✓ ✓
**DS HPP**

The Exclusive Or 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 discrete status flag indicates if the result of the Xor is zero.

XOR
A aaa

| Operand Data Type | | DL430 Range | DL440 Range | DL440 Range |
|---|---|---|---|---|
| | **A** | **aaa** | **aaa** | **aaa** |
| V-memory | V | All (See p. 3-40) | All (See p. 3-41) | All (See p. 3-42) |
| Pointer | P | -- | All V mem (See p. 3-41) | All V mem (See p. 3-42) |

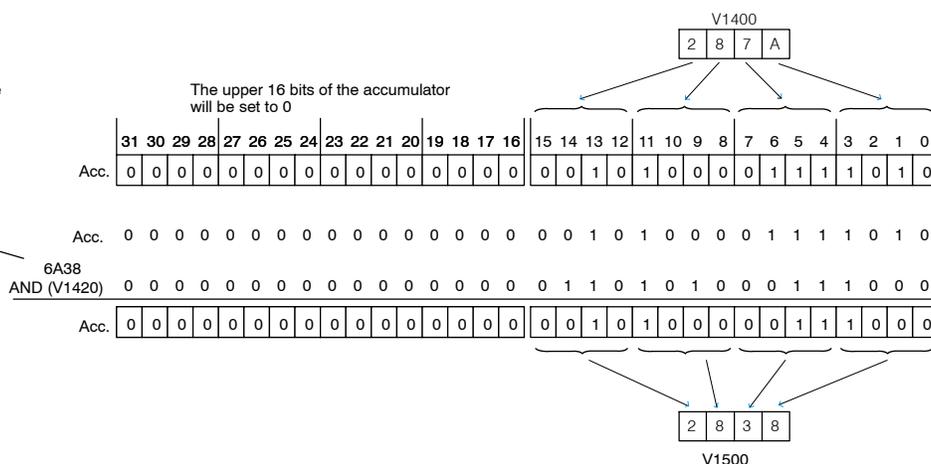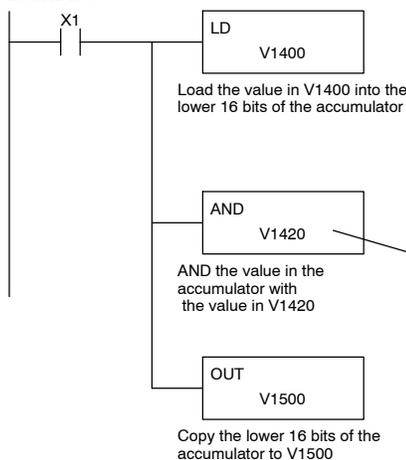| Discrete Bit Flags | Description |
|---|---|
| SP63 | Will be on if the result in the accumulator is zero |

**NOTE:** Status flags are valid only until another instruction uses the same flag.

In the following example, when X1 is on, the value in V1400 will be loaded into the accumulator using the Load instruction. The value in the accumulator is exclusive Hoped with V1420 using the Exclusive Or instruction. The value in the lower 16 bits of the accumulator is output to V1500 using the Out instruction.
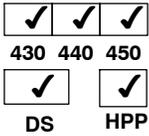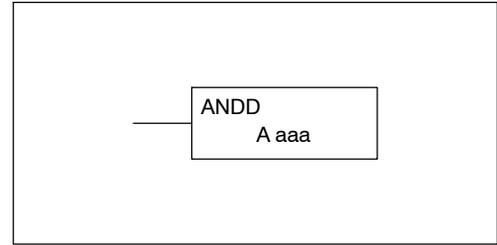
*DirectSOFT*



X1

LD
V1400

Load the value in V1400 into the lower 16 bits of the accumulator

XOR
V1420

XOR the value in the accumulator with the value in V1420

OUT
V1500

Copy the lower 16 bits of the accumulator to V1500

The upper 16 bits of the accumulator will be set to 0

| 31 30 29 28 | 27 26 25 24 | 23 22 21 20 | 19 18 17 16 | 15 14 13 12 | 11 10 9 8 | 7 6 5 4 | 3 2 1 0 |
|---|---|---|---|---|---|---|---|
Acc. 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 1 1 1 0 1 0

V1400
2 8 7 A

Acc. 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 1 1 1 0 1 0

6A38
XOR (V1420) 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 1 0 0 0 1 1 1 0 0 0

Acc. 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 0 0 0 0 1 0

4 E 4 2
V1500

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ← |
| LD | V | 1 | 4 | 0 | 0 | ← |
| SHFT | X | SHFT | OR | V | 1 | 4 | 2 | 0 | ← |
| OUT | V | 1 | 5 | 0 | 0 | ← |

Standard
RLL Instructions

**Exclusive Or Double (XORD)**

| ✔ | ✔ | ✔ |
|---|---|---|
| **430** | **440** | **450** |

| ✔ | | ✔ |
|---|---|---|
| **DS** | | **HPP** |

The Exclusive OR Double 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 Exclusive Or Double is zero or a negative number (the most significant bit is on).

```
          ┌──────────┐
──────────┤ XORD     │
          │   A aaa  │
          └──────────┘
```

| Operand Data Type | | DL430 Range | DL440 Range | DL450 Range |
|---|---|---|---|---|
| | **A** | **aaa** | **aaa** | **aaa** |
| V-memory | V | -- | All (See p. 3-41) | All (See p. 3-42) |
| Pointer | P | -- | All (See p. 3-41) | All (See p. 3-42) |
| Constant | K | 0-FFFF | 0-FFFF | 0-FFFF |

| Discrete Bit Flags | Description |
|---|---|
| SP63 | Will be on if the result in the accumulator is zero |
| SP70 | Will be on is 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 Load Double instruction. The value in the accumulator is exclusively Hoped with V1420 and V1421 using the Exclusive Or Double instruction. The value in the accumulator is output to V1500 and V1501 using the Out Double instruction.
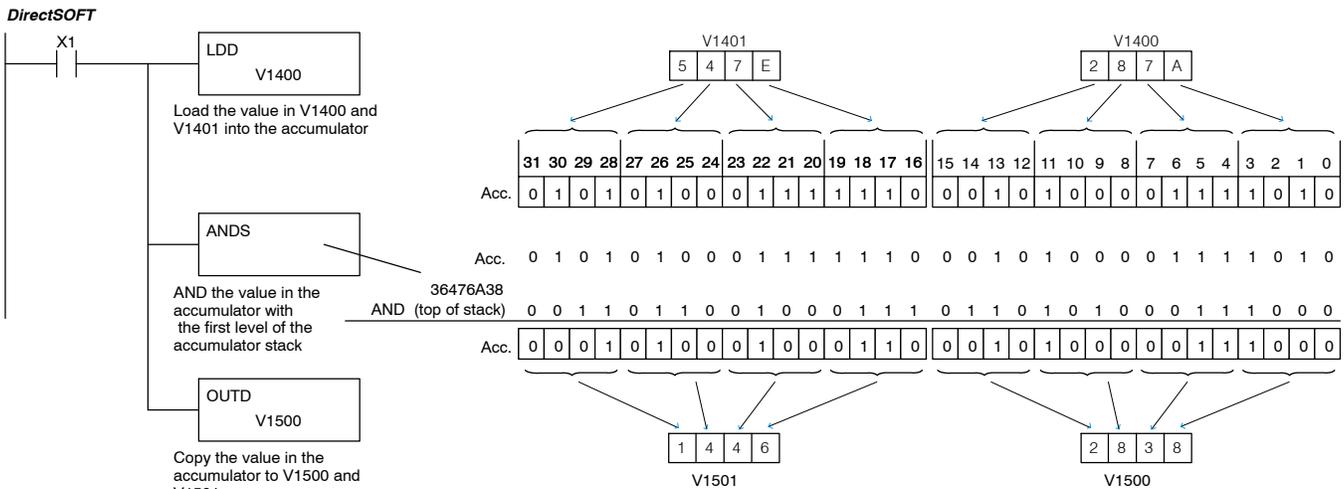
*DirectSOFT*

```
 X1      ┌──────────┐
──┤ ├────┤ LDD      │
         │   V1400  │
         └──────────┘
```
Load the value in V1400 and V1401 into the accumulator

```
         ┌──────────┐
─────────┤ XORD     │
         │   V1420  │
         └──────────┘
```
XORD the value in the accumulator with the value in V1420 and V1421

```
         ┌──────────┐
─────────┤ OUTD     │
         │   V1500  │
         └──────────┘
```
Copy the value in the accumulator to V1500 and V1501

V1401
| 5 | 4 | 7 | E |

V1400
| 2 | 8 | 7 | A |

Acc.

| 31 30 29 28 | 27 26 25 24 | 23 22 21 20 | 19 18 17 16 | 15 14 13 12 | 11 10 9 8 | 7 6 5 4 | 3 2 1 0 |
|---|---|---|---|---|---|---|---|
| 0 1 0 1 | 0 1 0 0 | 0 1 1 1 | 1 1 1 0 | 0 0 1 0 | 1 0 0 0 | 0 1 1 1 | 1 0 1 0 |

Acc.  0 1 0 1 0 1 0 0 0 1 1 1 1 1 1 0  0 0 1 0 1 0 0 0 0 1 1 1 1 0 1 0

36476A38
XORD (V1421 and V1420)  0 0 1 1 0 1 1 0 0 1 0 0 0 1 1 1  0 1 1 0 1 0 1 0 0 0 1 1 1 0 0 0

Acc.
| 0 1 1 0 | 0 0 1 0 | 0 0 1 1 | 1 0 0 1 | 0 1 0 0 | 0 0 1 0 | 0 1 0 0 | 0 0 1 0 |

| 6 | 2 | 3 | 9 |
V1501

| 4 | 2 | 4 | 2 |
V1500

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
|---|---|---|---|

| LD | SHFT | D | SHFT | V | 1 | 4 | 0 | 0 | ↵ |
|---|---|---|---|---|---|---|---|---|---|

| SHFT | X | SHFT | OR | SHFT | D | SHFT | V | 1 | 4 | 2 | 0 | ↵ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|

| OUT | SHFT | D | SHFT | V | 1 | 5 | 0 | 0 | ↵ |
|---|---|---|---|---|---|---|---|---|---|---|

Standard RLL Instructions

## Exclusive Or Formatted (XORF)

| ✕ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | | ✓ |
|---|---|---|
| **DS** | | **HPP** |

The Exclusive Or Formatted instruction performs an exclusive OR of the binary value in the accumulator and a specified range of discrete memory bits (1–32).

```
            XORF        A aaa
                        K bbb
```
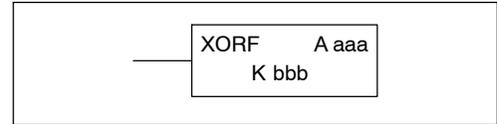
The instruction requires a starting location (Aaaa) and the number of bits (Bbbb) to be exclusive ORed. Discrete status flags indicate if the result of the Exclusive Or Formatted is zero or negative (the most significant bit =1).

| Operand Data Type | | DL440 Range | | DL450 Range | |
|---|---|---|---|---|---|
| | A/B | aaa | bbb | aaa | bbb |
| Inputs | X | 0–477 | –– | 0–1777 | –– |
| Outputs | Y | 0–477 | –– | 0–1777 | –– |
| Control Relays | C | 0–1777 | –– | 0–3777 | –– |
| Stage Bits | S | 0–1777 | –– | 0–1777 | –– |
| Timer Bits | T | 0–377 | –– | 0–377 | –– |
| Counter Bits | CT | 0–177 | –– | 0–377 | –– |
| Special Relays | SP | 0–137 320–717 | –– | 0–137 320–717 | –– |
| Global I/O | GX | 0–1777 | –– | 0–2777 | –– |
| Constant | K | –– | 1–32 | –– | 1–32 |

| Discrete Bit Flags | Description |
|---|---|
| SP63 | Will be on if the result in the accumulator is zero |
| SP70 | Will be on is 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 Load Formatted instruction. The value in the accumulator will be logically Exclusive Hoped with the bit pattern from Y20–Y23 using the Exclusive Or Formatted instruction. The value in the lower 4 bits of the accumulator is output to C20–C23 using the Out Formatted instruction.

## Exclusive Or with Stack (XORS)

| × | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

The Exclusive Or with Stack 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 Exclusive Or with Stack is zero or a negative number (the most significant bit is on).

| Discrete Bit Flags | Description |
|---|---|
| SP63 | Will be on if the result in the accumulator is zero |
| SP70 | Will be on is the result in the accumulator is negative |

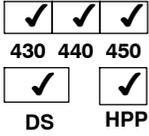**NOTE:** Status flags are valid only until another instruction uses the same flag.

In the following example when X1 is on, the binary value in the accumulator will be exclusive Hoped with the binary value in the first level of the accumulator stack. The result will reside in the accumulator.

*DirectSOFT*

Handheld Programmer Keystrokes

## Compare (CMP)

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

The compare 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. You can compare either binary or BCD numbers, as long as both numbers are of the same data type.

```
        CMP
         A aaa
```

| Operand Data Type | | DL430 Range | DL440 Range | DL450 Range |
|---|---|---|---|---|
| | **A** | **aaa** | **aaa** | **aaa** |
| V–memory | V | All (See p. 3–40) | All (See p. 3–41) | All (See p. 3–42) |
| Pointer | P | –– | All (See p. 3–41) | All (See p. 3–42) |

| 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 Load instruction. The value in the accumulator is compared with the value in V1410 using the Compare 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.

*DirectSOFT*

```
X1
─┤├─────────┬──────  LD          Load the constant value
             │            K4526    4526 into the lower 16 bits of
             │                     the accumulator
             │
             │        CMP          Compare the value in the
             └──────      V1410    accumulator with the value
                                   in V1410
      SP60              C30
      ─┤├───────────────( OUT )
```

Constant

| 4 | 5 | 2 | 6 |

The unused accumulator bits are set to zero

Acc. | 0 | 0 | 0 | 0 | 4 | 5 | 2 | 6 |

Compared with

| 8 | 9 | 4 | 5 |
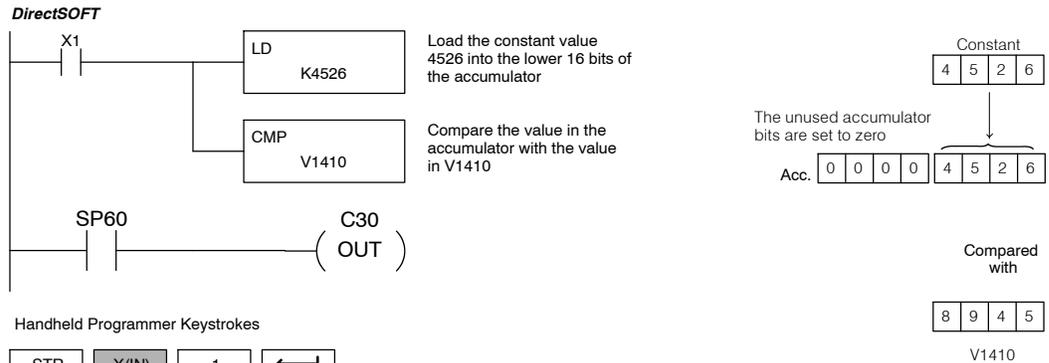
V1410

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ← |
|---|---|---|---|
| LD | K(CON) | 4 | 5 | 2 | 6 | ← |
| CMP | V | 1 | 4 | 1 | 0 | ← |
| STR | SPCL | 6 | 0 | ← |
| OUT | C(CR) | 3 | 0 | ← |

## Compare Double (CMPD)

✓ ✓ ✓
**430 440 450**

✓ | ✓
**DS** | **HPP**

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. You can compare either binary or BCD numbers, as long as both numbers are of the same data type.

```
        ┌─────────────┐
────────┤ CMPD        │
        │    A aaa    │
        └─────────────┘
```
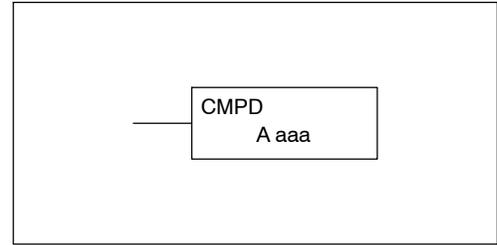
| Operand Data Type | | DL430 Range | DL440 Range | DL450 Range |
|---|---|---|---|---|
| | **A** | **aaa** | **aaa** | **aaa** |
| V-memory | V | -- | All (See p. 3-41) | All (See p. 3-42) |
| Pointer | P | -- | All (See p. 3-41) | All (See p. 3-42) |
| Constant | K | 0-FFFFFFFF | 0-FFFFFFFF | 0-FFFFFFFF |

| 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 V1400 and V1401 will be loaded into the accumulator using the Load Double instruction. The value in the accumulator is compared with the value in V1410 and V1411 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.

*DirectSOFT*

```
  X1              ┌─────────┐
──┤ ├─────┬───────┤ LDD     │      Load the value in V1400 and
          │       │   V1400 │      V1401 into the accumulator
          │       └─────────┘
          │       ┌─────────┐
          └───────┤ CMPD    │      Compare the value in the
                  │   V1410 │      accumulator with the value
                  └─────────┘      in V1410 and V1411

  SP60                   C30
──┤ ├──────────────────( OUT )
```

```
        V1401       V1400
       ┌─┬─┬─┬─┐   ┌─┬─┬─┬─┐
       │4│5│2│6│   │7│2│9│9│
       └─┴─┴─┴─┘   └─┴─┴─┴─┘
          │           │
          ▼           ▼
Acc.   ┌─┬─┬─┬─┐   ┌─┬─┬─┬─┐
       │4│5│2│6│   │7│2│9│9│
       └─┴─┴─┴─┘   └─┴─┴─┴─┘

          Compared
           with

       ┌─┬─┬─┬─┐   ┌─┬─┬─┬─┐
       │6│7│3│9│   │5│0│2│6│
       └─┴─┴─┴─┘   └─┴─┴─┴─┘
        V1411       V1410
```
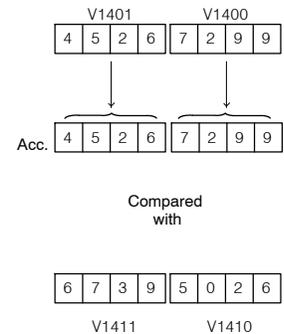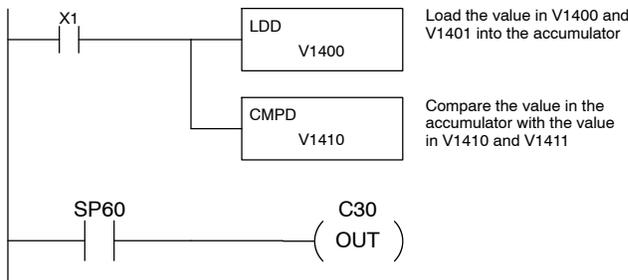
Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
| LD | SHFT | D | SHFT | V | 1 | 4 | 0 | 0 | ↵ |
| CMP | SHFT | D | SHFT | V | 1 | 4 | 1 | 0 | ↵ |
| STR | SPCL | 6 | 0 | ↵ |
| OUT | C(CR) | 3 | 0 | ↵ |

**Compare Formatted (CMPF)**

| X | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

The Compare Formatted 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.

```
          CMPF      A aaa
                    K bbb
```

| Operand Data Type | | DL440 Range | | DL450 Range | |
|---|---|---|---|---|---|
| | A/B | aaa | bbb | aaa | bbb |
| Inputs | X | 0–477 | –– | 0–1777 | –– |
| Outputs | Y | 0–477 | –– | 0–1777 | –– |
| Control Relays | C | 0–1777 | –– | 0–3777 | –– |
| Stage Bits | S | 0–1777 | –– | 0–1777 | –– |
| Timer Bits | T | 0–377 | –– | 0–377 | –– |
| Counter Bits | CT | 0–177 | –– | 0–377 | –– |
| Special Relays | SP | 0–137 320–717 | –– | 0–137 320–717 | –– |
| Global I/O | GX | 0–1777 | –– | 0–2777 | –– |
| Constant | K | –– | 1–32 | –– | 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*



Load the value of the specified discrete locations (C10–C13) into the accumulator

Compare the value in the accumulator with the value of the specified discrete location (Y20–Y23)

The unused accumulator bits are set to zero
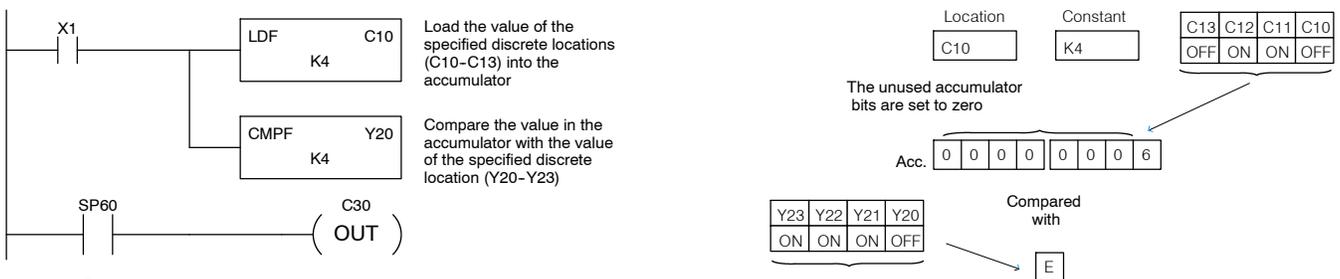
Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |

| LD | SHFT | F | SHFT | C | 1 | 0 | K(CON) | 4 |

| CMP | SHFT | F | SHFT | Y(OUT) | 2 | 0 | K(CON) | 4 | ↵ |

| STR | SPCL | 6 | 0 | ↵ |

| OUT | C(CR) | 3 | 0 | ↵ |

## Compare with Stack (CMPS)

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |
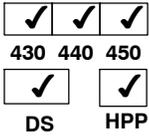
The Compare with Stack instruction is a 32-bit instruction that compares the value in the accumulator with the value in the first level of the accumulator stack.

```
          ┌─────────────┐
          │   CMPS      │
  ────────┤             │
          └─────────────┘
```
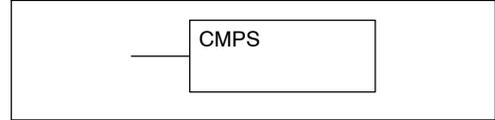
The corresponding status flag will be turned on indicating the result of the comparison. This does not affect the value in the accumulator.

| 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.

See Errata Sheet at the beginning of this file. The descriptions in this table have been updated.

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 or 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.

*DirectSOFT*

```
  X1              ┌──────────────┐   Load the value in V1400 and
  ─┤ ├─────┬──────┤ LDD          │   V1401 into the accumulator
          │      │      V1400   │
          │      └──────────────┘
          │      ┌──────────────┐   Load the value in V1410 and
          ├──────┤ LDD          │   V1411 into the accumulator
          │      │      V1410   │
          │      └──────────────┘
          │      ┌──────────────┐   Compare the value in the
          └──────┤ CMPS         │   accumulator with the value
                 │              │   in the first level of the
                 └──────────────┘   accumulator stack

  SP60                          C30
  ─┤ ├──────────────────────────( OUT )
```

|  | V1401 |  |  |  | V1400 |  |  |
|---|---|---|---|---|---|---|---|
| 6 | 5 | 0 | 0 | 3 | 5 | 4 | 4 |

| Acc. | 6 | 5 | 0 | 0 | 3 | 5 | 4 | 4 |

|  | V1411 |  |  |  | V1410 |  |  |
|---|---|---|---|---|---|---|---|
| 5 | 5 | 0 | 0 | 3 | 5 | 4 | 4 |

| Acc. | 5 | 5 | 0 | 0 | 3 | 5 | 4 | 4 |

Compared with Top of Stack
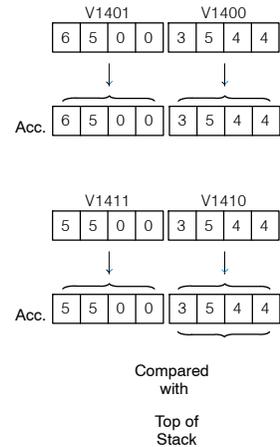
Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
| LD | SHFT | D | SHFT | V | 1 | 4 | 0 | 0 | ↵ |
| LD | SHFT | D | SHFT | V | 1 | 4 | 1 | 0 | ↵ |
| CMP | SHFT | S | ↵ |
| STR | SPCL | 6 | 0 | ↵ |
| OUT | C(CR) | 3 | 0 | ↵ |

Standard RLL Instructions

**Compare Real Number (CMPR)**

| ✕ | ✕ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | | ✓ |
|---|---|---|
| **DS** | | **HPP** |

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.

```
        ┌──────────────┐
────────┤ CMPR         │
        │    A aaa     │
        └──────────────┘
```

| Operand Data Type | | DL450 Range |
|---|---|---|
| | **A** | **aaa** |
| V–memory | V | All (See p. 3–42) |
| Pointer | P | All (See p. 3–42) |
| Constant | R | –3.402823E+038 to<br>+ –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. |
| SP75 | On when a real number instruction is executed and a non–real 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 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).

*DirectSOFT*

```
      X1              ┌──────────────┐
    ──┤ ├─────────────┤ LDR          │     Load the real number
                      │      R7.0    │     representation for decimal 7
                      └──────────────┘     into the accumulator.

                      ┌──────────────┐     Compare the value with the
                  ────┤ CMPR         │     real number representation
                      │      R6.0    │     for decimal 6.
                      └──────────────┘

      SP62                              C1
    ──┤ ├─────────────────────────────( OUT )
```

Acc.

| 4 | 0 | E | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|

CMPR
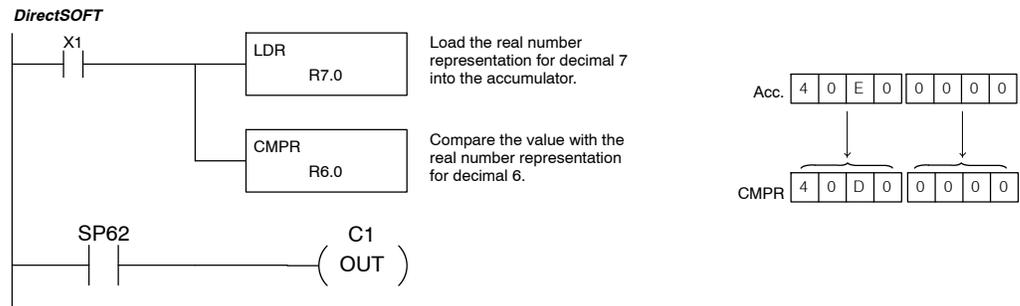
| 4 | 0 | D | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | LD | SHFT | D | SHFT | ←┘ |
|---|---|---|---|---|---|---|---|

| K(CON) | 4 | 0 | E | 0 | 0 | 0 | 0 | 0 | ←┘ |
|---|---|---|---|---|---|---|---|---|---|

| CMP | SHFT | R | SHFT |
|---|---|---|---|

| K(CON) | 4 | 0 | D | 0 | 0 | 0 | 0 | 0 | ←┘ |
|---|---|---|---|---|---|---|---|---|---|

| STR | SPCL | 6 | 0 | ←┘ |
|---|---|---|---|---|

| OUT | C(CR) | 3 | 0 | ←┘ |
|---|---|---|---|---|

# Math Instructions

**Add
(ADD)**

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

Add is a 16 bit instruction that adds a BCD value in the accumulator with a BCD value in a V-memory location (Aaaa). The result resides in the accumulator.

```
        ┌──────────┐
────────┤ ADD      │
        │   A aaa  │
        └──────────┘
```

| Operand Data Type | | DL430 Range | DL440 Range | DL450 Range |
|---|---|---|---|---|
| | **A** | **aaa** | **aaa** | **aaa** |
| V-memory | V | All (See p. 3-40) | All (See p. 3-41) | All (See p. 3-42) |
| Pointer | P | All (See p. 3-40) | All (See p. 3-41) | All (See p. 3-42) |

| 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 V1400 will be loaded into the accumulator using the Load instruction. The value in the lower 16 bits of the accumulator are added to the value in V1420 using the Add instruction. The value in the accumulator is copied to V1500 using the Out instruction.



*DirectSOFT*

```
  X1
──┤ ├──────┬───── LD        Load the value in V1400 into the
           │      V1400     lower 16 bits of the accumulator
           │
           ├───── ADD       Add the value in the lower
           │      V1420     16 bits of the accumulator
           │                with the value in V1420
           │
           └───── OUT        Copy the value in the lower 16
                  V1500      bits of the accumulator to V1500
```

V1400
| 4 | 9 | 3 | 5 |

The unused accumulator bits are set to zero

```
      0 0 0 0  4 9 3 5    (Accumulator)
   +           2 5 0 0    (V1420)
   ─────────────────────
   Acc.  7  4  3  5
```

| 7 | 4 | 3 | 5 |
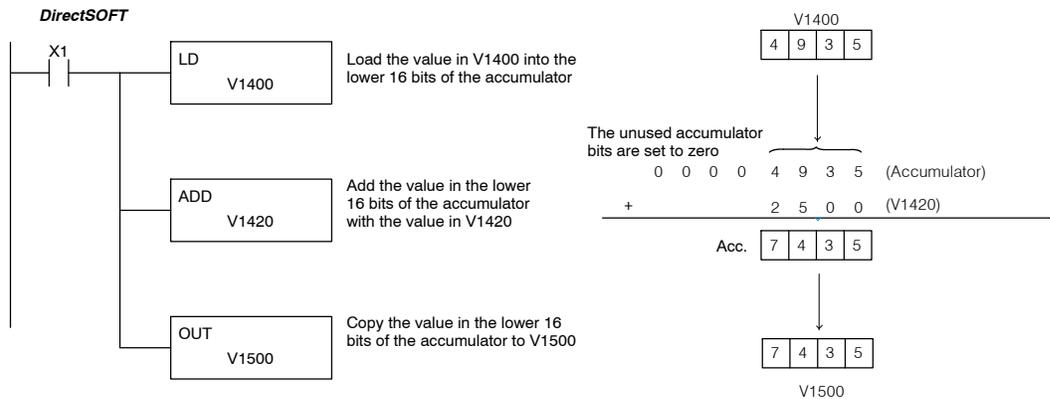
V1500

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ← |
|---|---|---|---|
| LD | V | 1 | 4 | 0 | 0 | ← |
| ADD | V | 1 | 4 | 2 | 0 | ← |
| OUT | V | 1 | 5 | 0 | 0 | ← |

## Add Double (ADDD)

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

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.

```
        ┌──────────┐
────────┤ ADDD     │
        │    A aaa │
        └──────────┘
```

| Operand Data Type | | DL430 Range | DL440 Range | DL450 Range |
|---|---|---|---|---|
| | **A** | **aaa** | **aaa** | **aaa** |
| V-memory | V | All (See p. 3-40) | All (See p. 3-41) | All (See p. 3-42) |
| Pointer | P | All (See p. 3-40) | All (See p. 3-41) | All (See p. 3-42) |
| Constant | K | 0-99999999 | 0-99999999 | 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 V1400 and V1401 will be loaded into the accumulator using the Load Double instruction. The value in the accumulator is added with the value in V1420 and V1421 using the Add Double instruction. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.

**DirectSOFT**

```
X1      ┌──────────┐
─┤ ├────┤ LDD      │
        │    V1400 │
        └──────────┘
     Load the value in V1400 and
     V1401 into the accumulator

        ┌──────────┐
        │ ADDD     │
        │    V1420 │
        └──────────┘
     Add the value in the
     accumulator with the value
     in V1420 and V1421

        ┌──────────┐
        │ OUTD     │
        │    V1500 │
        └──────────┘
     Copy the value in the
     accumulator to V1500
     and V1501
```

|  V1401  |  V1400  |
|---|---|
| 6 7 3 9 | 5 0 2 6 |

```
      6  7  3  9    5  0  2  6   (Accumulator)
   +  2  0  0  0    4  0  4  6   (V1421 and V1420)
   ─────────────────────────────
Acc. 8  7  3  9    9  0  7  2
```

| 8 7 3 9 | 9 0 7 2 |
|---|---|
| V1501 | V1500 |
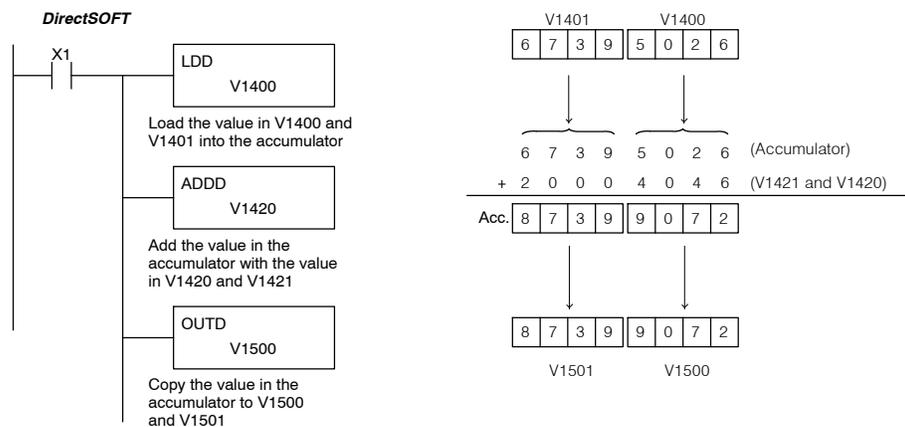
Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
|---|---|---|---|

| LD | SHFT | D | SHFT | V | 1 | 4 | 0 | 0 | ↵ |
|---|---|---|---|---|---|---|---|---|---|

| ADD | SHFT | D | SHFT | V | 1 | 4 | 2 | 0 | ↵ |
|---|---|---|---|---|---|---|---|---|---|

| OUT | SHFT | D | SHFT | V | 1 | 5 | 0 | 0 | ↵ |
|---|---|---|---|---|---|---|---|---|---|

Standard
RLL Instructions

**Add Real (ADDR)**

| × | × | ✓ |
|---|---|---|
| 430 | 440 | 450 |

| ✓ | × |
|---|---|
| DS | HPP |

Add Real is a 32-bit instruction that adds a real number, which is either two consecutive V-memory locations or a 32-bit constant, to a real number in the accumulator. Both numbers must conform to the IEEE floating point format. The result is a 32-bit real number that resides in the accumulator.

```
            ADDR
            A aaa
```

| Operand Data Type | | DL450 Range |
|---|---|---|
| | **A** | **aaa** |
| V-memory | V | All (See p. 3-42) |
| Pointer | P | All V mem (See p. 3-42) |
| Constant | R | -3.402823E+038 to +3.402823E+038 |

| 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 a valid 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. |
| SP75 | On when a real number instruction is executed and a non-real number was encountered. |

**NOTE:** Status flags are valid only until another instruction uses the same flag.

*DirectSOFT*

X1 — LDR R7.0
Load the real number 7.0 into the accumulator

ADDR R15.0
Add the real number 15.0 to the accumulator contents, which is in real number format.

OUTD V1400
Copy the result in the accumulator to V1400 and V1401.

```
4 0 E 0  0 0 0 0

  7 (decimal)    4 0 E 0  0 0 0 0  (Accumulator)
+ 1 5          + 4 1 7 0  0 0 0 0  (ADDR)
  2 2       Acc. 4 1 B 0  0 0 0 0
```

V1401  V1400
4 1 B 0  0 0 0 0  (Hex number)

Real Value

| 8 4 2 1 | 8 4 2 1 | 8 4 2 1 | 8 4 2 1 | 8 4 2 1 | 8 4 2 1 | 8 4 2 1 | 8 4 2 1 |
Acc. 0 1 0 0 | 0 0 0 1 | 1 0 1 1 | 0 0 0 0 | ...

Sign Bit  Exponent (8 bits)  Mantissa (23 bits)

128 + 2 + 1 = 131     1.011 x 2 (exp 4) = 10110. binary= 22 decimal
131 - 127 = 4

**NOTE:** If the value being added to a real number is 16,777,216 times smaller than the real number, the calculation will not work.

**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.

## Subtract (SUB)

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

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.

```
——————|  SUB
           A aaa
```
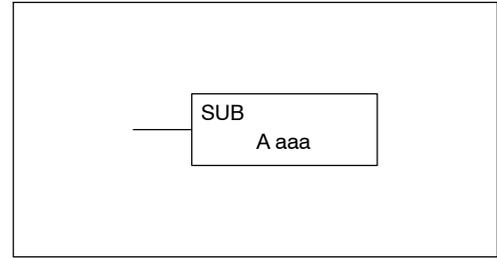
| Operand Data Type | | DL430 Range | DL440 Range | DL450 Range |
|---|---|---|---|---|
| | A | aaa | aaa | aaa |
| | A | aaa | aaa | aaa |
| V-memory | V | All (See p. 3-40) | All (See p. 3-41) | All (See p. 3-42) |
| Pointer | P | All (See p. 3-40) | All (See p. 3-41) | All (See p. 3-42) |

| 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 V1400 will be loaded into the accumulator using the Load instruction. The value in V1420 is subtracted from the value in the accumulator using the Subtract instruction. The value in the accumulator is copied to V1500 using the Out instruction.

*DirectSOFT*

```
    X1        LD
——| |——————————
              V1400
```
Load the value in V1400 into the lower 16 bits of the accumulator

```
              SUB
              V1420
```
Subtract the value in V1420 from the value in the lower 16 bits of the accumulator

```
              OUT
              V1500
```
Copy the value in the lower 16 bits of the accumulator to V1500

V1400

| 2 | 4 | 7 | 5 |
|---|---|---|---|

The unused accumulator bits are set to zero

```
         0  0  0  0   2  4  7  5   (Accumulator)
       —             1  5  9  2    (V1420)
       ————————————————————————————
       Acc.     | 0 | 8 | 8 | 3 |
```

| 0 | 8 | 8 | 3 |
|---|---|---|---|
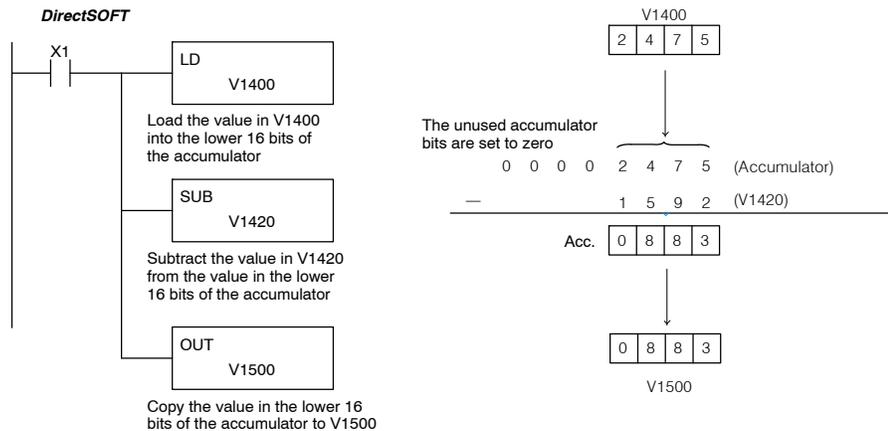
V1500

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
| LD | V | 1 | 4 | 0 | 0 | ↵ |
| SUB | V | 1 | 4 | 2 | 0 | ↵ |
| OUT | V | 1 | 5 | 0 | 0 | ↵ |

## Subtract Double (SUBD)

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |
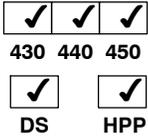
Subtract Double is a 32 bit instruction that subtracts the BCD value (Aaaa), which is either two consecutive V-memory locations or an 8-digit (max.) constant, from the BCD value in the accumulator. The result resides in the accumulator.

```
        SUBD
         A aaa
```

| Operand Data Type | | DL430 Range | DL440 Range | DL450 Range |
|---|---|---|---|---|
| | **A** | **aaa** | **aaa** | **aaa** |
| V-memory | V | All (See p. 3–40) | All (See p. 3–41) | All (See p. 3–42) |
| Pointer | P | All (See p. 3–40) | All (See p. 3–41) | All (See p. 3–42) |
| Constant | K | 0–99999999 | 0–99999999 | 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 V1400 and V1401 will be loaded into the accumulator using the Load Double instruction. The value in V1420 and V1421 is subtracted from the value in the accumulator. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.

**Subtract Real (SUBR)**

| × | × | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | × |
|---|---|
| **DS** | **HPP** |

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. Both numbers must conform to the IEEE floating point format. The result is a 32-bit real number that resides in the accumulator.

```
SUBR
   A aaa
```

| Operand Data Type | | DL450 Range |
|---|---|---|
| | **A** | **aaa** |
| V-memory | V | All (See p. 3-42) |
| Pointer | P | All V mem (See p. 3-42) |
| Constant | R | -3.402823E+038 to +3.402823E+038 |

| 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 a valid 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. |
| SP75 | On when a real number instruction is executed and a non-real number was encountered. |

**NOTE:** Status flags are valid only until another instruction uses the same flag.

*DirectSOFT*



```
X1
─┤ ├────────────┌─────────────┐
              │ LDR         │
              │    R22.0    │
              └─────────────┘
```
Load the real number 22.0 into the accumulator.

```
              ┌─────────────┐
              │ SUBR        │
              │    R15.0    │
              └─────────────┘
```
Subtract the real number 15.0 from the accumulator contents, which is in real number format.

```
              ┌─────────────┐
              │ OUTD        │
              │    V1400    │
              └─────────────┘
```
Copy the result in the accumulator to V1400 and V1401.

```
        4 1 B 0   0 0 0 0

  2  2  (decimal)     4 1 B 0  0 0 0 0   (Accumulator)
- 1  5            +   4 1 7 0  0 0 0 0   (SUBR)
─────────────────────────────────────────
     7              Acc. 4 0 E 0  0 0 0 0
```

```
              V1401      V1400
            4 0 E 0    0 0 0 0   (Hex number)
```

Real Value

```
     8 4 2 1  8 4 2 1  8 4 2 1  8 4 2 1   8 4 2 1  8 4 2 1  8 4 2 1  8 4 2 1
Acc. 0 1 0 0  0 0 0 0  0 1 1 1  0 0 0 0   0 0 0 0  0 0 0 0  0 0 0 0  0 0 0 0
```

Sign Bit    Exponent (8 bits)      Mantissa (23 bits)

128 + 1 = 129
129 - 127 = 2
Implies 2 (exp 2)

1.11 x 2 (exp 2) = 111. binary= 7 decimal

**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.

## Multiply (MUL)

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |
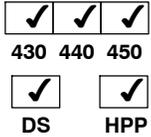
Multiply is a 16 bit instruction that multiplies the BCD value (Aaaa), which is either a V-memory location or a 4-digit (max.) constant, by the BCD value in the lower 16 bits of the accumulator The result can be up to 8 digits and resides in the accumulator.

```
        ┌─────────┐
────────┤ MUL     │
        │   A aaa │
        └─────────┘
```

| Operand Data Type | | DL430 Range | DL440 Range | DL450 Range |
|---|---|---|---|---|
| | **A** | **aaa** | **aaa** | **aaa** |
| V-memory | V | All (See p. 3-40) | All (See p. 3-41) | All (See p. 3-42) |
| Pointer | P | All (See p. 3-40) | All (See p. 3-41) | All (See p. 3-42) |
| Constant | K | 0-9999 | 0-9999 | 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 V1400 will be loaded into the accumulator using the Load instruction. The value in V1420 is multiplied by the value in the accumulator. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.

*DirectSOFT*

```
        X1          ┌─────────┐
        ┤/├─────────┤ LD      │
                    │   V1400 │
                    └─────────┘
                    Load the value in V1400
                    into the lower 16 bits of
                    the accumulator

                    ┌─────────┐
                    │ MUL     │
                    │   V1420 │
                    └─────────┘
                    The value in V1420 is
                    multiplied by the value in
                    the accumulator

                    ┌─────────┐
                    │ OUTD    │
                    │   V1500 │
                    └─────────┘
                    Copy the value in the
                    accumulator to V1500
                    and V1501
```

V1400

| 1 | 0 | 0 | 0 |
|---|---|---|---|

The unused accumulator bits are set to zero

```
0 0 0 0 1 0 0 0   (Accumulator)
×         2 5     (V1420)
─────────────────
```

Acc. | 0 | 0 | 0 | 2 | 5 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|

| 0 | 0 | 0 | 2 | 5 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|

V1501      V1500

**Handheld Programmer Keystrokes**

| STR | X(IN) | 1 | ↵ |
|---|---|---|---|

| LD | V | 1 | 4 | 0 | 0 | ↵ |
|---|---|---|---|---|---|---|

| MUL | V | 1 | 4 | 2 | 0 | ↵ |
|---|---|---|---|---|---|---|

| OUT | SHFT | D | SHFT | V | 1 | 5 | 0 | 0 | ↵ |
|---|---|---|---|---|---|---|---|---|---|

**Multiply Double (MULD)**

| | | |
|---|---|---|
| ✕ | ✕ | ✔ |
| **430** | **440** | **450** |

| | |
|---|---|
| ✔ | ✕ |
| **DS** | **HPP** |

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.

```
┌──────────┐
│ MULD     │
│   A aaa  │
└──────────┘
```
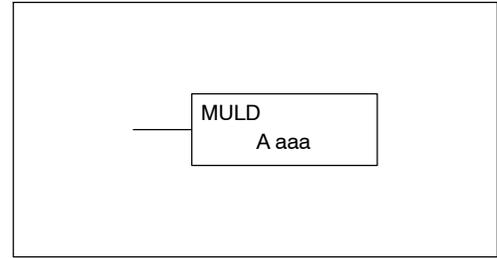
| Operand Data Type | | DL450 Range |
|---|---|---|
| | **A** | **aaa** |
| V-memory | V | All (See p. 3–42) |
| Pointer | P | – – |

| 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 is 24691356.

*DirectSOFT*

| | | |
|---|---|---|
| X1 ─┤├─┬─ | LDD Kbc614e | Load the hex equivalent of 12345678 decimal into the accumulator. |
| | BCD | Convert the value to BCD format. It will occupy eight BCD digits (32 bits). |
| | OUTD V1400 | Output the number to V1400 and V1401 using the OUTD instruction. |
| | LD K2 | Load the constant K2 into the accumulator. |
| | MULD V1400 | Multiply the accumulator contents (2) by the 8-digit number in V1400 and V1401. |
| | OUTD V1402 | Move the result in the accumulator to V1402 and V1403 using the OUTD instruction. |

```
                    1  2  3  4   5  6  7  8   (Accumulator)
                      │              │
                      ▼              ▼
                  ┌──────────┐  ┌──────────┐
                    V1401         V1400
                  │1│2│3│4│    │5│6│7│8│

            ×                         2   (Accumulator)
           ─────────────────────────────
       Acc. │2│4│6│9│ │1│3│5│6│
                      │              │
                      ▼              ▼
           │2│4│6│9│ │1│3│5│6│
             V1403        V1500
```

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↲ | LD | SHFT | D | SHFT | | |
|---|---|---|---|---|---|---|---|---|---|
| K(CON) | SHFT | B(H) | C(H) | 6 | 1 | 4 | E(H) | ↲ | BCD | ↲ |
| OUT | SHFT | D | SHFT | V | 1 | 4 | 0 | 0 | ↲ |
| LD | K(CON) | 2 | ↲ | MUL | SHFT | D | | | |
| V | 1 | 4 | 0 | 0 | ↲ | | | | |
| OUT | SHFT | D | SHFT | V | 1 | 4 | 0 | 2 | ↲ |

## Multiply Real (MULR)

| | | |
|---|---|---|
| ✕ | ✕ | ✓ |
| **430** | **440** | **450** |
| ✓ | ✕ | |
| **DS** | **HPP** | |

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 conform to the IEEE floating point format.

```
            ┌──────────────┐
────────────┤ MULR         │
            │    A aaa     │
            └──────────────┘
```

| Operand Data Type | | DL450 Range |
|---|---|---|
| | **A** | **aaa** |
| V-memory | V | All (See p. 3-42) |
| Pointer | P | All (See p. 3-42) |
| Constant | R | -3.402823E+038 to +3.402823E+038 |

| 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 a valid 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. |
| SP75 | On when a real number instruction is executed and a non-real number was encountered. |

**NOTE:** Status flags are valid only until another instruction uses the same flag.

*DirectSOFT*



```
X1        ┌──────────┐
──┤/├──────┤ LDR      │
          │    R7.0   │
          └──────────┘
          Load the real number 7.0
          into the accumulator.

          ┌──────────┐
          │ MULR     │
          │   R15.0  │
          └──────────┘
          Multiply the accumulator
          contents by the real number
          15.0

          ┌──────────┐
          │ OUTD     │
          │   V1400  │
          └──────────┘
          Copy the result in the accumulator
          to V1400 and V1401.
```

| 4 | 0 | E | 0 | 0 | 0 | 0 | 0 |

```
        7   (decimal)    4 0 E 0  0 0 0 0   (Accumulator)
      x 1 5            + 4 1 7 0  0 0 0 0   (MULR)
      1 0 5       Acc. | 4 2 D 2 | 0 0 0 0 |
```

V1401    V1400

| 4 | 2 | D | 2 | 0 | 0 | 0 | 0 | (Hex number) |

Real Value

| 8 4 2 1 | 8 4 2 1 | 8 4 2 1 | 8 4 2 1 | 8 4 2 1 | 8 4 2 1 | 8 4 2 1 | 8 4 2 1 |
|---|---|---|---|---|---|---|---|
Acc. | 0 1 0 0 | 0 0 1 0 | 1 1 0 1 | 0 0 1 0 | 0 0 0 0 | 0 0 0 0 | 0 0 0 0 | 0 0 0 0 |

Sign Bit   Exponent (8 bits)              Mantissa (23 bits)

128 + 4 + 1 = 133          1.101001 x 2 (exp 6) = 1101001. binary= 105 decimal
133 - 127 = 6
Implies 2 (exp 6)

**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.

Standard
RLL Instructions

**Divide
(DIV)**

✓ ✓ ✓
**430 440 450**

✓ ✓
**DS HPP**

Divide is a 16 bit instruction that divides the BCD value in the 32-bit 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.

```
            ┌───────────┐
────────────┤ DIV       │
            │     A aaa │
            └───────────┘
```

| Operand Data Type | | DL430 Range | DL440 Range | DL450 Range |
|---|---|---|---|---|
| | **A** | **aaa** | **aaa** | **aaa** |
| V-memory | V | All (See p. 3-40) | All (See p. 3-41) | All (See p. 3-42) |
| Pointer | P | All (See p. 3-40) | All (See p. 3-41) | All V mem (See p. 3-42) |
| Constant | K | 0-9999 | 0-9999 | 0-9999 |

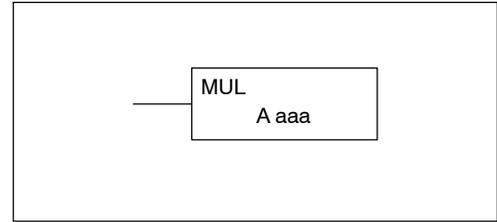| Discrete Bit Flags | Description |
|---|---|
| SP53 | On when the value of the operand is larger than the accumulator can work with. |
| SP63 | On when the result of the instruction causes the value in the accumulator to be zero. |
| SP70 | On anytime the value in the accumulator is negative. |
| SP75 | 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 V1400 will be loaded into the accumulator using the Load instruction. The value in the accumulator will be divided by the value in V1420 using the Divide instruction. The value in the accumulator is copied to V1500 using the Out instruction.

*DirectSOFT*

```
 X1                ┌─────────────┐
──┤ ├──────────────┤ LD          │
                   │     V1400   │
                   └─────────────┘
                   Load the value in V1400 into the
                   lower 16 bits of the accumulator

                   ┌─────────────┐
                   │ DIV         │
                   │     V1420   │
                   └─────────────┘
                   The value in the accumulator is
                   divided by the value in V1420

                   ┌─────────────┐
                   │ OUT         │
                   │     V1500   │
                   └─────────────┘
                   Copy the value in the lower 16
                   bits of the accumulator to V1500
```

V1400

| 5 | 0 | 0 | 0 |
|---|---|---|---|

The unused accumulator bits are set to zero

0  0  0  0  5  0  0  0    (Accumulator)

÷                    5  0    (V1420)

Acc. | 1 | 0 | 0 |     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

First stack location contains the remainder

| 1 | 0 | 0 |
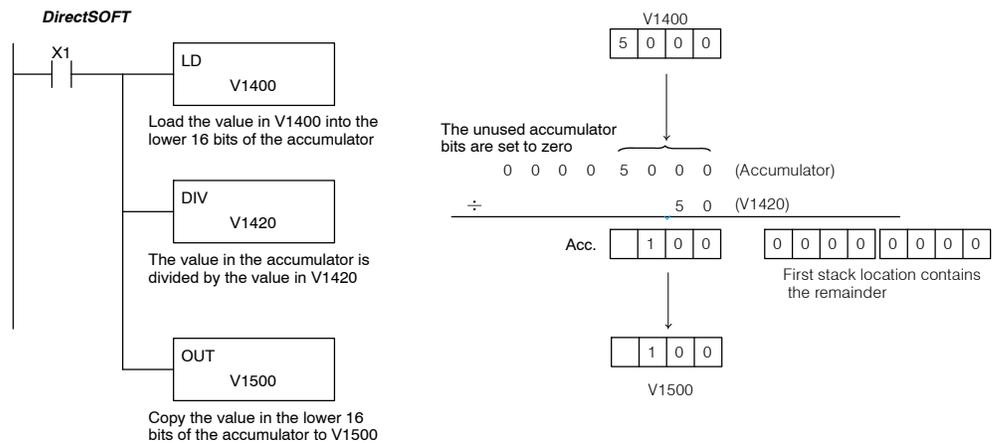
V1500

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
|---|---|---|---|
| LD | V | 1 | 4 | 0 | 0 | ↵ |
| DIV | V | 1 | 4 | 2 | 0 | ↵ |
| OUT | V | 1 | 5 | 0 | 0 | ↵ |

**Divide Double
(DIVD)**

| × | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

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.

```
          ┌──────────┐
──────────┤ DIVD     │
          │     A aaa│
          └──────────┘
```

| Operand Data Type | | DL440 Range | DL450 Range |
|---|---|---|---|
| | **A** | **aaa** | **aaa** |
| V-memory | V | All (See p. 3-41) | All (See p. 3-42) |
| Pointer | P | All (See p. 3-41) | All (See p. 3-42) |

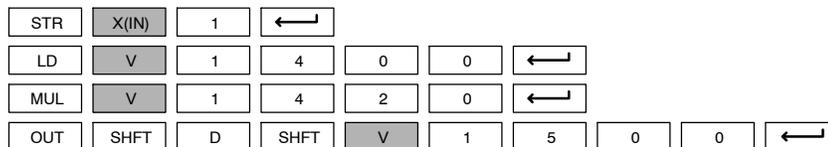| Discrete Bit Flags | Description |
|---|---|
| SP53 | On when the value of the operand is larger than the accumulator can work with. |
| SP63 | On when the result of the instruction causes the value in the accumulator to be zero. |
| SP70 | On anytime the value in the accumulator is negative. |
| SP75 | 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 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.

*DirectSOFT*

```
  X1           ┌──────────┐
──┤ ├──┬───────┤ LDD      │
       │       │     V1400│
       │       └──────────┘
       │       Load the value in V1400 and
       │       V1401 into the accumulator
       │
       │       ┌──────────┐
       ├───────┤ DIVD     │
       │       │     V1420│
       │       └──────────┘
       │       The value in the accumulator
       │       is divided by the value in
       │       V1420 and V1421
       │
       │       ┌──────────┐
       └───────┤ OUTD     │
               │     V1500│
               └──────────┘
               Copy the value in the
               accumulator to V1500
               and V1501
```

```
        V1401        V1400
       ┌─┬─┬─┬─┐    ┌─┬─┬─┬─┐
       │0│1│5│0│    │0│0│0│0│
       └─┴─┴─┴─┘    └─┴─┴─┴─┘

The unused accumulator
bits are set to zero

      0 1 5 0   0 0 0 0    (Accumulator)
  ÷   0 0 0 0   0 0 5 0    (V1421 and V1420)

Acc. ┌─┬─┬─┬─┬─┬─┬─┬─┐   ┌─┬─┬─┬─┬─┬─┬─┬─┐
     │0│0│0│3│0│0│0│0│   │0│0│0│0│0│0│0│0│
     └─┴─┴─┴─┴─┴─┴─┴─┘   └─┴─┴─┴─┴─┴─┴─┴─┘
                         First stack location contains
                         the remainder

     ┌─┬─┬─┬─┬─┬─┬─┬─┐
     │0│0│0│3│0│0│0│0│
     └─┴─┴─┴─┴─┴─┴─┴─┘
      V1501       V1500
```
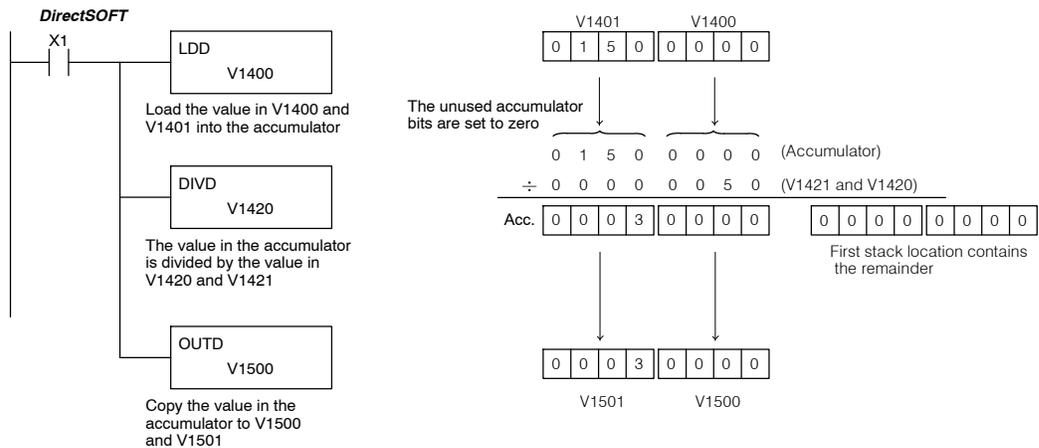
Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ | | | | | |
|---|---|---|---|---|---|---|---|---|
| LD | SHFT | D | SHFT | V | 1 | 4 | 0 | 0 | ↵ |
| DIV | SHFT | D | SHFT | V | 1 | 4 | 2 | 0 | ↵ |
| OUT | SHFT | D | SHFT | V | 1 | 5 | 0 | 0 | ↵ |

Standard
RLL Instructions

## Divide Real (DIVR)

| | | |
|---|---|---|
| ✕ | ✕ | ✓ |
| 430 | 440 | 450 |

| | |
|---|---|
| ✓ | ✕ |
| DS | HPP |

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 conform to the IEEE floating point format.
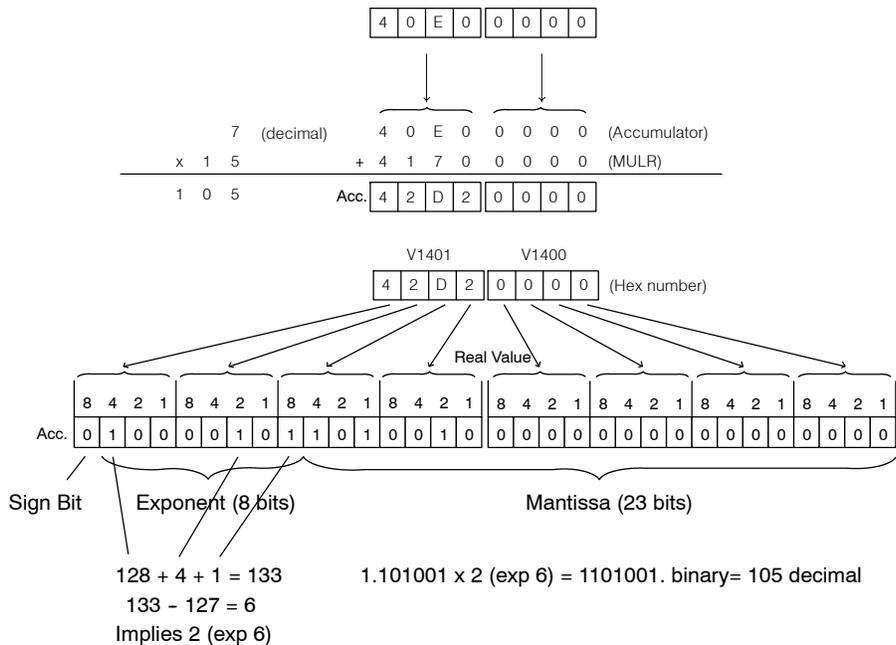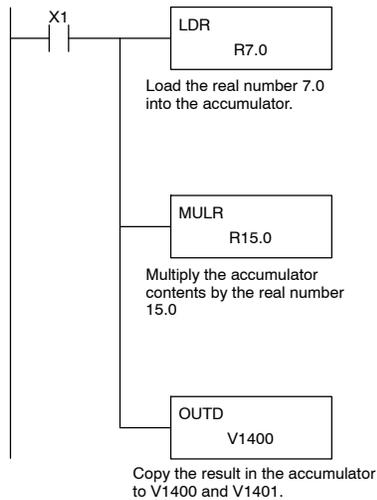
```
      DIVR
         A aaa
```

| Operand Data Type | | DL450 Range |
|---|---|---|
| | A | aaa |
| V-memory | V | All (See p. 3–42) |
| Pointer | P | All (See p. 3–42) |
| Constant | R | –3.402823E+038 to +3.402823E+038 |

| 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 a valid 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. |
| SP75 | On when a real number instruction is executed and a non-real number was encountered. |

**NOTE:** Status flags are valid only until another instruction uses the same flag.

**DirectSOFT**

X1 — LDR R15.0
Load the real number 15.0 into the accumulator.

DIVR R10.0
Divide the accumulator contents by the real number 10.0.

OUTD V1400
Copy the result in the accumulator to V1400 and V1401.

15 (decimal) / 10 = 1.5

Sign Bit | Exponent (8 bits) | Mantissa (23 bits)

64 + 32 + 16 + 8 + 4 + 2 + 1 = 127
127 – 127 = 0
Implies 2 (exp 0)

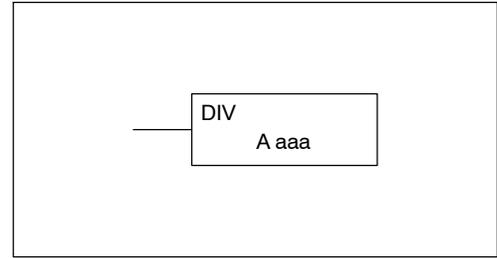1.1 x 2 (exp 0) = 1.1 binary= 1.5 decimal

**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.

## Add Binary (ADDB)

✓ ✓ ✓
**430 440 450**

✓ ✓
**DS HPP**

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.

```
          ADDB
          A aaa
```

| Operand Data Type | | DL430 Range | DL440 Range | DL450 Range |
|---|---|---|---|---|
| | **A** | **aaa** | **aaa** | **aaa** |
| V-memory | V | All (See p. 3-40) | All (See p. 3-41) | All (See p. 3-42) |
| Pointer | P | All (See p. 3-40) | All V mem (See p. 3-41) | All V mem (See p. 3-42) |
| Constant | K | 0-FFFF | 0-FFFF | 0-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 a 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 (constant) 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 OUTD instruction.



**DirectSOFT**

Use either V-memory OR Constant

```
X1        LD              LD
          V1400           K2565
```
Load the value in V1400 into the lower 16 bits of the accumulator

```
          BIN
```

```
          ADDB
          V1420
```
The binary value in the accumulator is added to the binary value in V1420

```
          OUTD
          V1500
```
Copy the value in the lower 16 bits of the accumulator to V1500 and V1501

V1400
| 0 | A | 0 | 5 |

The unused accumulator bits are set to zero

```
0 0 0 0   0 A 0 5   (Accumulator)
+             1 2 C 4   (V1420)
Acc.      1 C C 9
```

| 1 | C | C | 9 |

V1500

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
| LD | V | 1 | 4 | 0 | 0 | ↵ |
| ADD | SHFT | B | SHFT | V | 1 | 4 | 2 | 0 | ↵ |
| OUT | SHFT | D | SHFT | V | 1 | 5 | 0 | 0 | ↵ |

Standard
RLL Instructions

**Add Binary Double (ADDBD)**

| × | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

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.

```
        ┌──────────────┐
────────┤  ADDBD       │
        │    A aaa     │
        └──────────────┘
```

| Operand Data Type | | DL440 Range | DL450 Range |
|---|---|---|---|
| | **A** | **aaa** | **aaa** |
| V-memory | V | All (See p. 3–41) | All (See p. 3–42) |
| Pointer | P | All (See p. 3–41) | All V mem (See p. 3–42) |
| Constant | K | 0–FFFFFFFF | 0–FFFFFFFF |

| 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 a 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 (constant) will be loaded into the accumulator using the LDD instruction. The binary value in the accumulator is added with the binary value in V1420 and V1421 using the ADDBD instruction. The value in the accumulator is copied to V1500 and V1501 using the OUTD instruction.



*DirectSOFT*

Use either
V–memory OR Constant

```
 X1          ┌──────────┐              ┌──────────┐
──┤├─────────┤ LDD      │              │ LDD      │
             │   V1400  │              │   K2561  │
             └──────────┘              └──────────┘
   Load the value in V1400 and
   V1401 into the accumulator         ┌──────────┐
                                      │ BIN      │
                                      └──────────┘
             ┌──────────┐
             │ ADDBD    │
             │   V1420  │
             └──────────┘
   The binary value in the
   accumulator is added with the
   value in V1420 and V1421

             ┌──────────┐
             │ OUTD     │
             │   V1500  │
             └──────────┘
   Copy the value in the
   accumulator to V1500
   and V1501
```

|  | V1401 | | | | V1400 | | | |
|---|---|---|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | 0 | A | 0 | 1 |

| (Accumulator) | 0 | 0 | 0 | 0 | 0 | A | 0 | 1 |
|---|---|---|---|---|---|---|---|---|
| + (V1421 and V1420) | 1 | 0 | 0 | 0 | C | 0 | 1 | 0 |
| Acc. | 1 | 0 | 0 | 0 | C | A | 1 | 1 |

| | 1 | 0 | 0 | 0 | C | A | 1 | 1 |
|---|---|---|---|---|---|---|---|---|

V1501  V1500

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
|---|---|---|---|

| LD | SHFT | D | SHFT | V | 1 | 4 | 0 | 0 | ↵ |
|---|---|---|---|---|---|---|---|---|---|

| ADD | SHFT | B | D | SHFT | V | 1 | 4 | 2 | 0 | ↵ |
|---|---|---|---|---|---|---|---|---|---|---|

| OUT | SHFT | D | SHFT | V | 1 | 5 | 0 | 0 | ↵ |
|---|---|---|---|---|---|---|---|---|---|

Standard
RLL Instructions

## Subtract Binary (SUBB)

| | | |
|:---:|:---:|:---:|
| ✓ | ✓ | ✓ |
| **430** | **440** | **450** |

| | |
|:---:|:---:|
| ✓ | ✓ |
| **DS** | **HPP** |

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.

```
        ┌──────────────┐
────────┤ SUBB         │
        │    A aaa     │
        └──────────────┘
```

| Operand Data Type | | DL430 Range | DL440 Range | DL450 Range |
|---|---|---|---|---|
| | **A** | **aaa** | **aaa** | **aaa** |
| V-memory | V | All (See p. 3-40) | All (See p. 3-41) | All (See p. 3-42) |
| Pointer | P | All (See p. 3-40) | All (See p. 3-41) | All (See p. 3-42) |
| Constant | K | 0-FFFF | 0-FFFF | 0-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 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.

*DirectSOFT*

Use either
V-memory  OR  Constant

```
X1        ┌─────────┐              ┌─────────┐
─┤├───────┤ LD      │        ┌─────┤ LD      │
          │   V1400 │        │     │   K1024 │
          └─────────┘        │     └─────────┘
   Load the value in V1400 into the
   lower 16 bits of the accumulator │
                                    │  ┌─────────┐
          ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ───┴──┤ BIN     │
                                       └─────────┘
          ┌─────────┐
          │ SUBB    │
          │   V1420 │
          └─────────┘
   The binary value in V1420 is
   subtracted from the value in
   the accumulator

          ┌─────────┐
          │ OUT     │
          │   V1500 │
          └─────────┘
   Copy the value in the lower 16
   bits of the accumulator to V1500
```

V1400
| 1 | 0 | 2 | 4 |

The unused accumulator bits are set to zero

| 0 | 0 | 0 | 0 | 1 | 0 | 2 | 4 | (Accumulator)
— | | | | | | 0 | A | 0 | B | (V1420)

Acc. | 0 | 6 | 1 | 9 |

| 0 | 6 | 1 | 9 |
V1500
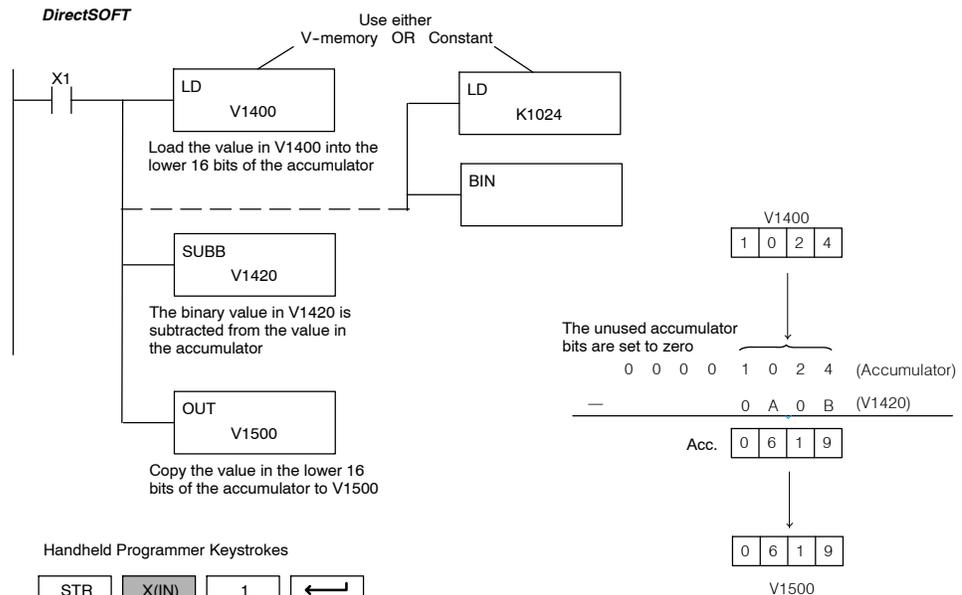
Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ | | | | | |
| LD | V | 1 | 4 | 0 | 0 | ↵ | | |
| SUB | SHFT | B | SHFT | V | 1 | 4 | 2 | 0 | ↵ |
| OUT | V | 1 | 5 | 0 | 0 | ↵ | | |

**Subtract Binary Double (SUBBD)**

| X | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | | ✓ |
|---|---|---|
| **DS** | | **HPP** |

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.

```
            ┌─────────────┐
────────────┤ SUBBD       │
            │   A aaa     │
            └─────────────┘
```

| Operand Data Type | | DL440 Range | DL450 Range |
|---|---|---|---|
| | **A** | **aaa** | **aaa** |
| V-memory | V | All (See p. 3-41) | All (See p. 3-42) |
| Pointer | P | All (See p. 3-42) | All (See p. 3-42) |
| Constant | K | 0-FFFFFFFF | 0-FFFFFFFF |

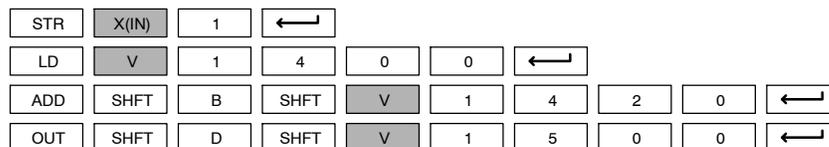| Discrete Bit Flags | Description |
|---|---|
| SP63 | On when the result of the instruction causes the value in the accumulator to be zero. |
| SP64 | On when the 16 bit subtraction instruction results in a borrow. |
| SP65 | On when the 32 bit subtraction instruction results in a borrow. |
| SP70 | 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.
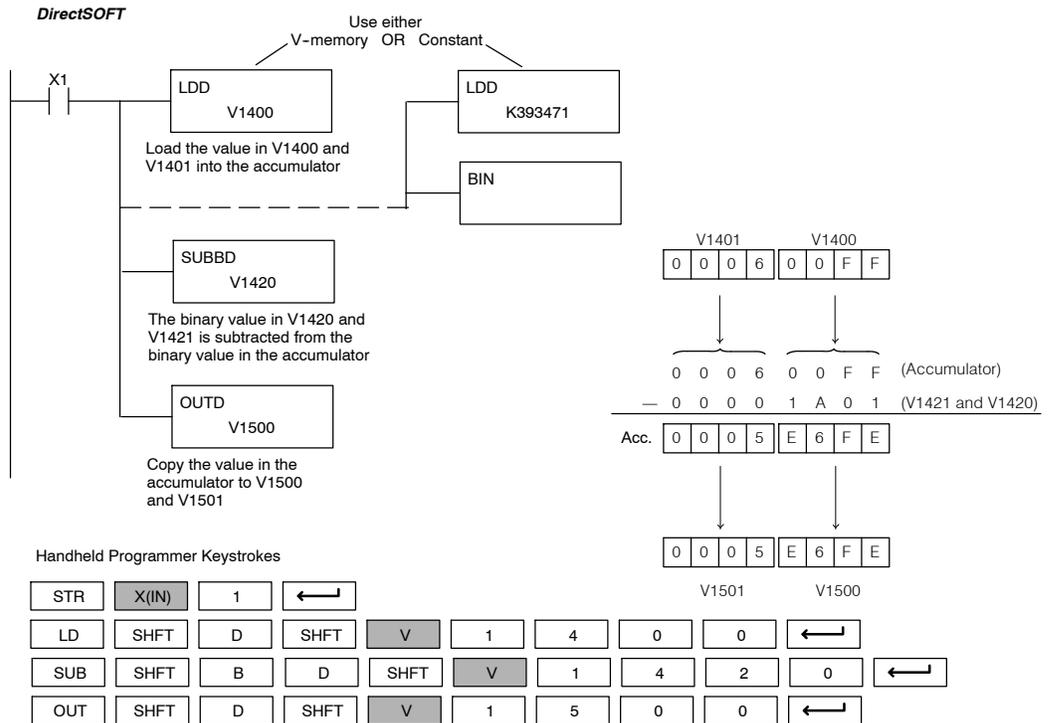
## Multiply Binary (MULB)

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |
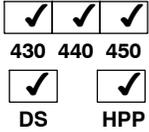
Multiply Binary is a 16 bit instruction that multiplies the binary value (Aaaa), which is either a V-memory location or a 4-digit (max.) binary constant, by the binary value in the accumulator. The result can be up to 32 bits and resides in the accumulator.

```
        ┌─────────┐
────────┤ MULB    │
        │   A aaa │
        └─────────┘
```

| Operand Data Type | | DL430 Range | DL440 Range | DL450 Range |
|---|---|---|---|---|
| | **A** | **aaa** | **aaa** | **aaa** |
| V-memory | V | All (See p. 3-40) | All (See p. 3-41) | All (See p. 3-42) |
| Pointer | P | All (See p. 3-40) | All (See p. 3-41) | All (See p. 3-42) |
| Constant | K | 0-FFFF | 0-FFFF | 0-FFFF |

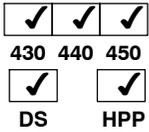| Discrete Bit Flags | Description |
|---|---|
| SP63 | On when the result of the instruction causes the value in the accumulator to be zero. |
| SP70 | On anytime the value in the accumulator is negative. |

**NOTE:** Status flags are valid only until another instruction uses the same flag.

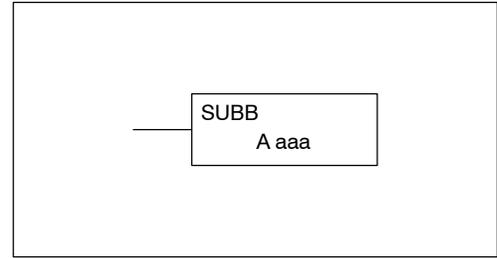In the following example, when X1 is on, the value in V1400 will be loaded into the accumulator using the Load instruction. The binary value in V1420 is multiplied by the binary value in the accumulator using the Multiply Binary instruction. The value in the accumulator is copied to V1500 using the Out instruction.



*DirectSOFT*

Use either V-memory OR Constant

LD
V1400
Load the value in V1400 into the lower 16 bits of the accumulator

LDD
K2561

BIN

MULB
V1420
The binary value in V1420 is multiplied by the binary value in the accumulator

OUTD
V1500
Copy the value in the lower 16 bits of the accumulator to V1500 and V1501

V1400
| 0 | A | 0 | 1 |

The unused accumulator bits are set to zero

| 0 | 0 | 0 | 0 | 0 | A | 0 | 1 | (Accumulator)
× | | | | 0 | 0 | 2 | E | (V1420)
Acc. | 0 | 0 | 0 | 1 | C | C | 2 | E |

| 0 | 0 | 0 | 1 | C | C | 2 | E |
V1501    V1500

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
| LD | V | 1 | 4 | 0 | 0 | ↵ |
| MUL | SHFT | B | SHFT | V | 1 | 4 | 2 | 0 | ↵ |
| OUT | SHFT | D | SHFT | V | 1 | 5 | 0 | 0 | ↵ |

**Divide Binary (DIVB)**

| 430 | 440 | 450 |
|:---:|:---:|:---:|
| ✓ | ✓ | ✓ |

| DS | HPP |
|:---:|:---:|
| ✓ | ✓ |

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.

```
        ┌──────────┐
────────┤ DIVB     │
        │     A aaa│
        └──────────┘
```

| Operand Data Type | | DL430 Range | DL440 Range | DL450 Range |
|---|---|---|---|---|
| | **A** | **aaa** | **aaa** | **aaa** |
| V–memory | V | All (See p. 3–40) | All (See p. 3–41) | All (See p. 3–42) |
| Pointer | P | All (See p. 3–40) | All (See p. 3–41) | All (See p. 3–42) |
| Constant | K | 0–FFFF | 0–FFFF | 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.



*DirectSOFT*
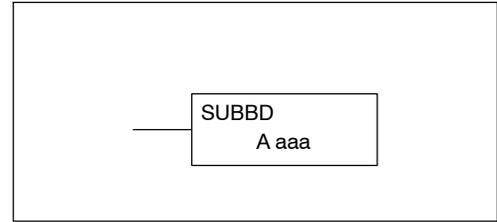
Use either V–memory OR Constant

Load the value in V1400 into the lower 16 bits of the accumulator

The binary value in the accumulator is divided by the binary value in V1420

Copy the value in the lower 16 bits of the accumulator to V1500

The unused accumulator bits are set to zero

| 0 | 0 | 0 | 0 | F | A | 0 | 1 | (Accumulator) |

÷ 0 0 5 0 (V1420)

Acc. | 0 | 3 | 2 | 0 |   | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

First stack location contains the remainder

| 0 | 3 | 2 | 0 |

V1500

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
| LD | V | 1 | 4 | 0 | 0 | ↵ |
| DIV | SHFT | B | SHFT | V | 1 | 4 | 2 | 0 | ↵ |
| OUT | V | 1 | 5 | 0 | 0 | ↵ |

## Add Formatted (ADDF)

| | | |
|---|---|---|
| ✕ | ✓ | ✓ |
| **430** | **440** | **450** |

| | |
|---|---|
| ✓ | ✓ |
| **DS** | **HPP** |

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 | | DL440 Range | | DL450 Range | |
|---|---|---|---|---|---|
| | **A** | **aaa** | **bbb** | **aaa** | **bbb** |
| Inputs | X | 0–477 | -- | 0–1777 | -- |
| Outputs | Y | 0–477 | -- | 0–1777 | -- |
| Control Relays | C | 0–1777 | -- | 0–3777 | -- |
| Stage Bits | S | 0–1777 | -- | 0–1777 | -- |
| Timer Bits | T | 0–377 | -- | 0–377 | -- |
| Counter Bits | CT | 0–177 | -- | 0–377 | -- |
| Special Relays | SP | 0–137 320–717 | -- | 0–137 320–717 | -- |
| Global I/O | GX | 0–1777 | -- | 0–2777 | -- |
| Constant | K | -- | 1–32 | -- | 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 | 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 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 added to the value in the accumulator using the Add Formatted instruction. The value in the lower four bits of the accumulator is copied to Y10–Y13 using the Out Formatted instruction.



| X3 | X2 | X1 | X0 |
|---|---|---|---|
| ON | OFF | OFF | OFF |

The unused accumulator bits are set to zero

```
            0  0  0  0   0  0  0  8   (Accumulator)
         +                        3   (C0-C3)
   Acc.  0  0  0  0   0  0  1  1
```

| C3 | C2 | C1 | C0 |
|---|---|---|---|
| OFF | OFF | ON | ON |

| Y13 | Y12 | Y11 | Y10 |
|---|---|---|---|
| OFF | OFF | OFF | ON |

*DirectSOFT*

```
       X6        ┌─────────────┐
   ───┤ ├────────┤ LDF      X0 │   Load the value represented
                 │          K4 │   by discrete locations X0-X3
                 └─────────────┘   into the accumulator

                 ┌─────────────┐
                 │ ADDF      C0 │   Add the value in the
                 │          K4 │   accumulator with the value
                 └─────────────┘   represented by discrete
                                   location C0-C3

                 ┌─────────────┐
                 │ OUTF     Y10 │   Copy the lower 4 bits of the
                 │          K5 │   accumulator to discrete
                 └─────────────┘   locations Y10-Y14
```
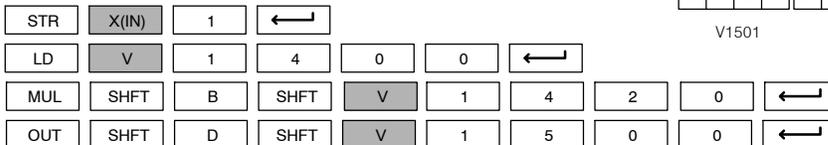
Handheld Programmer Keystrokes

| STR | X(IN) | 6 | ↵ |
|---|---|---|---|

| LD | SHFT | F | SHFT | X(IN) | 0 | K(CON) | 4 | ↵ |
|---|---|---|---|---|---|---|---|---|

| ADD | SHFT | F | SHFT | C(CR) | 0 | K(CON) | 4 | ↵ |
|---|---|---|---|---|---|---|---|---|

| OUT | SHFT | F | SHFT | Y(OUT) | 1 | 0 | K(CON) | 4 | ↵ |
|---|---|---|---|---|---|---|---|---|

Standard RLL Instructions

**Subtract Formatted (SUBF)**

| × | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | | ✓ |
|---|---|---|
| **DS** | | **HPP** |

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.

```
                ┌─────────────────┐
────────────────┤ SUBF      A aaa │
                │           K bbb │
                └─────────────────┘
```

| Operand Data Type | | DL440 Range | | DL450 Range | |
|---|---|---|---|---|---|
| | **A** | **aaa** | **bbb** | **aaa** | **bbb** |
| Inputs | X | 0–477 | –– | 0–1777 | –– |
| Outputs | Y | 0–477 | –– | 0–1777 | –– |
| Control Relays | C | 0–1777 | –– | 0–3777 | –– |
| Stage Bits | S | 0–1777 | –– | 0–1777 | –– |
| Timer Bits | T | 0–377 | –– | 0–377 | –– |
| Counter Bits | CT | 0–177 | –– | 0–377 | –– |
| Special Relays | SP | 0–137 320–717 | –– | 0–137 320–717 | –– |
| Global I/O | GX | 0–1777 | –– | 0–2777 | –– |
| Constant | K | –– | 1–32 | –– | 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 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 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 location C0–C3 is subtracted from the value in the accumulator using the Subtract Formatted instruction. The value in the lower four bits of the accumulator is copied to Y10–Y13 using the Out Formatted instruction.

*DirectSOFT*

```
  X6
──┤ ├──────┬─────┐ LDF        X0      Load the value represented
           │     │ K4                 by discrete locations X0–X3
           │     └─────┘              into the accumulator
           │
           ├─────┐ SUBF       C0      Subtract the value
           │     │ K4                 represented by C0–C3 from
           │     └─────┘              the value in the accumulator
           │
           ├─────┐ OUTF      Y10      Copy the lower 4 bits of the
           │     │ K4                 accumulator to discrete
           │     └─────┘              locations Y10–Y13
```

| X3 | X2 | X1 | X0 |
|---|---|---|---|
| ON | OFF | OFF | ON |

The unused accumulator bits are set to zero

```
0 0 0 0  0 0 0 9   (Accumulator)
—              8   (C0–C3)
ACC. 0 0 0 0  0 0 0 1
```

| C3 | C2 | C1 | C0 |
|---|---|---|---|
| ON | OFF | OFF | OFF |

| Y13 | Y12 | Y11 | Y10 |
|---|---|---|---|
| OFF | OFF | OFF | ON |
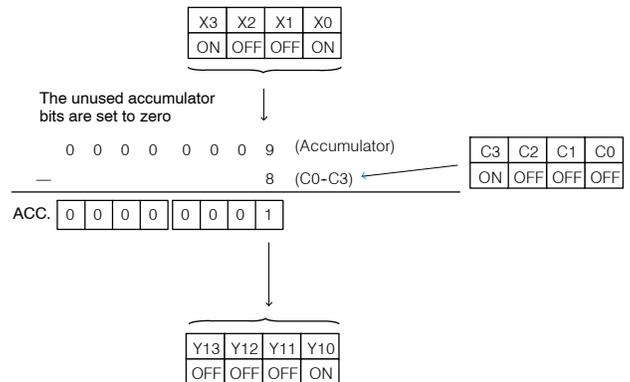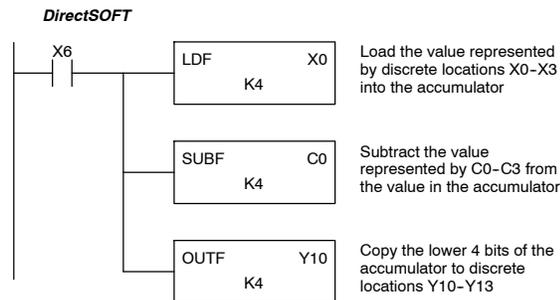
Handheld Programmer Keystrokes

| STR | X(IN) | 6 | ↵ |
|---|---|---|---|

| LD | SHFT | F | SHFT | X(IN) | 0 | K(CON) | 4 | ↵ |
|---|---|---|---|---|---|---|---|---|

| SUBF | SHFT | F | SHFT | C(CR) | 0 | K(CON) | 4 | ↵ |
|---|---|---|---|---|---|---|---|---|

| OUT | SHFT | F | SHFT | Y(OUT) | 1 | 0 | K(CON) | 4 | ↵ |
|---|---|---|---|---|---|---|---|---|---|

**Multiply Formatted (MULF)**

| | | |
|---|---|---|
| ✕ | ✓ | ✓ |
| 430 | 440 | 450 |

| | |
|---|---|
| ✓ | ✓ |
| DS | HPP |

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 | | DL440 Range | | DL450 Range | |
|---|---|---|---|---|---|
| | A/B | aaa | bbb | aaa | bbb |
| Inputs | X | 0–477 | – – | 0–1777 | – – |
| Outputs | Y | 0–477 | – – | 0–1777 | – – |
| Control Relays | C | 0–1777 | – – | 0–3777 | – – |
| Stage Bits | S | 0–1777 | – – | 0–1777 | – – |
| Timer Bits | T | 0–377 | – – | 0–377 | – – |
| Counter Bits | CT | 0–177 | – – | 0–377 | – – |
| Special Relays | SP | 0–137 320–717 | – – | 0–137 320–717 | – – |
| Global I/O | GX | 0–1777 | – – | 0–2777 | – – |
| Constant | K | – – | 1–16 | – – | 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 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 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.

*DirectSOFT*

```
     X6
─────┤ ├─────┌─────────────┐     Load the value represented
             │  LDF     X0 │     by discrete locations X0–X3
             │         K4  │     into the accumulator
             └─────────────┘
             ┌─────────────┐     Multiply the value in the
             │  MULF    C0 │     accumulator with the value
             │         K4  │     represented by discrete
             └─────────────┘     locations C0–C3
             ┌─────────────┐     Copy the lower 4 bits of the
             │  OUTF   Y10 │     accumulator to discrete
             │         K4  │     locations Y10–Y13
             └─────────────┘
```

| X3 | X2 | X1 | X0 |
|---|---|---|---|
| OFF | OFF | ON | ON |

The unused accumulator bits are set to zero

0 0 0 0   0 0 0 3   (Accumulator)
×          2   (C0–C3)

| C3 | C2 | C1 | C0 |
|---|---|---|---|
| OFF | OFF | ON | OFF |

Acc. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 6 |

| Y13 | Y12 | Y11 | Y10 |
|---|---|---|---|
| OFF | ON | ON | OFF |

Handheld Programmer Keystrokes

| STR | X(IN) | 6 | ↵ |
|---|---|---|---|

| LD | SHFT | F | SHFT | X(IN) | 0 | K(CON) | 4 | ↵ |
|---|---|---|---|---|---|---|---|---|

| MUL | SHFT | F | SHFT | C(CR) | 0 | K(CON) | 4 | ↵ |
|---|---|---|---|---|---|---|---|---|

| OUT | SHFT | F | SHFT | Y(OUT) | 1 | 0 | K(CON) | 4 | ↵ |
|---|---|---|---|---|---|---|---|---|---|

Standard
RLL Instructions

## Divide Formatted (DIVF)

| | | |
|---|---|---|
| ✕ | ✓ | ✓ |
| **430** | **440** | **450** |
| ✓ | | ✓ |
| **DS** | | **HPP** |

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.

```
┌─────────────────────┐
│ DIVF      A aaa      │
│           K bbb      │
└─────────────────────┘
```
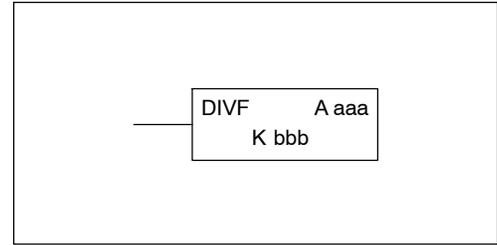
| Operand Data Type | | DL440 Range | | DL450 Range | |
|---|---|---|---|---|---|
| | A/B | aaa | bbb | aaa | bbb |
| Inputs | X | 0–477 | –– | 0–477 | –– |
| Outputs | Y | 0–477 | –– | 0–477 | –– |
| Control Relays | C | 0–1777 | –– | 0–1777 | –– |
| Stage Bits | S | 0–1777 | –– | 0–1777 | –– |
| Timer Bits | T | 0–377 | –– | 0–377 | –– |
| Counter Bits | CT | 0–177 | –– | 0–177 | –– |
| Special Relays | SP | 0–137 320–717 | –– | 0–137 320–717 | –– |
| Global I/O | GX | 0–1777 | –– | 0–1777 | –– |
| Constant | K | –– | 1–16 | –– | 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 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 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.
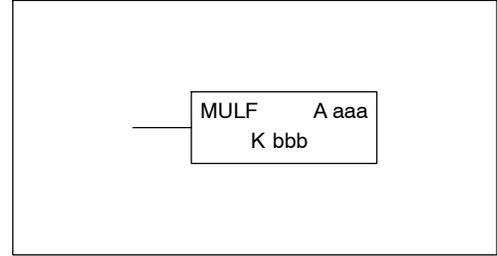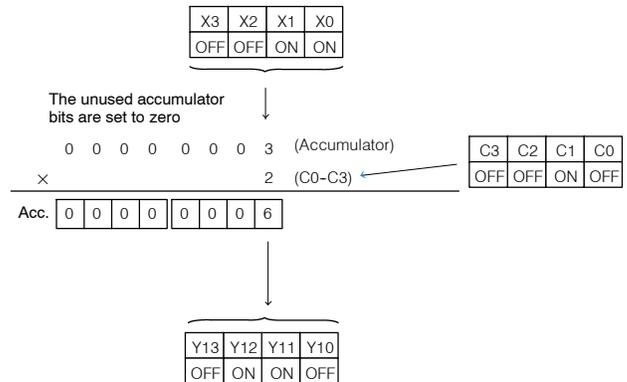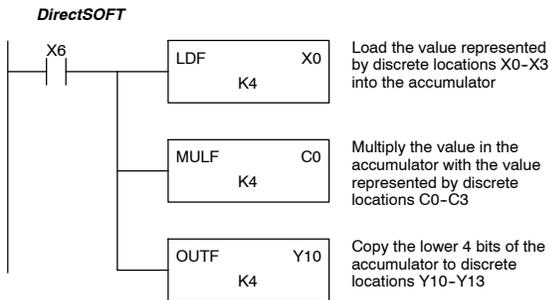
**DirectSOFT**

```
 X6
──┤ ├──┬──┌─────────────┐
        │  │ LDF      X0 │   Load the value represented
        │  │       K4    │   by discrete locations X0–X3
        │  └─────────────┘   into the accumulator
        │
        ├──┌─────────────┐
        │  │ DIVF     C0 │   Divide the value in the
        │  │       K4    │   accumulator with the value
        │  └─────────────┘   represented by discrete
        │                    location C0–C3
        │
        └──┌─────────────┐
           │ OUTF    Y10 │   Copy the lower 4 bits of the
           │       K4    │   accumulator to discrete
           └─────────────┘   locations Y10–Y13
```

| X3 | X2 | X1 | X0 |
|---|---|---|---|
| ON | OFF | OFF | OFF |

The unused accumulator bits are set to zero

```
0  0  0  0    0  0  0  8   (Accumulator)
÷                      2   (C0-C3)
```

| C3 | C2 | C1 | C0 |
|---|---|---|---|
| OFF | OFF | ON | OFF |

Acc. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4 |  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

First stack location contains the remainder

| Y13 | Y12 | Y11 | Y10 |
|---|---|---|---|
| OFF | ON | OFF | OFF |
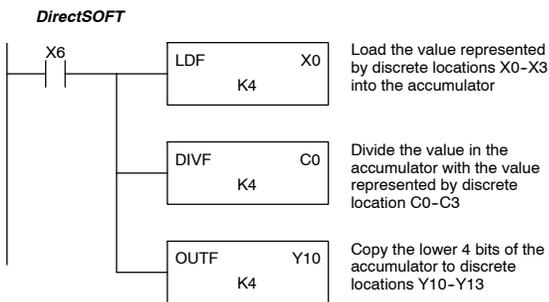
Handheld Programmer Keystrokes

| STR | X(IN) | 6 | ← |
|---|---|---|---|

| LD | SHFT | F | SHFT | X(IN) | 0 | K(CON) | 4 | ← |
|---|---|---|---|---|---|---|---|---|

| DIV | SHFT | F | SHFT | C(CR) | 0 | K(CON) | 4 | ← |
|---|---|---|---|---|---|---|---|---|

| OUT | SHFT | F | SHFT | Y(OUT) | 1 | 0 | K(CON) | 4 | ← |
|---|---|---|---|---|---|---|---|---|---|

Standard RLL Instructions

## Add Top of Stack (ADDS)

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |
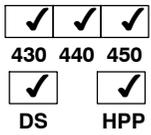
Add Top of Stack is a 32 bit instruction that adds the BCD value in the accumulator with the BCD value in the first level of the accumulator stack. The result resides in the accumulator. The value in the first level of the accumulator stack is removed and all stack values are moved up one level.

```
        ┌──────────┐
────────┤ ADDS     │
        │          │
        └──────────┘
```

| 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 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.
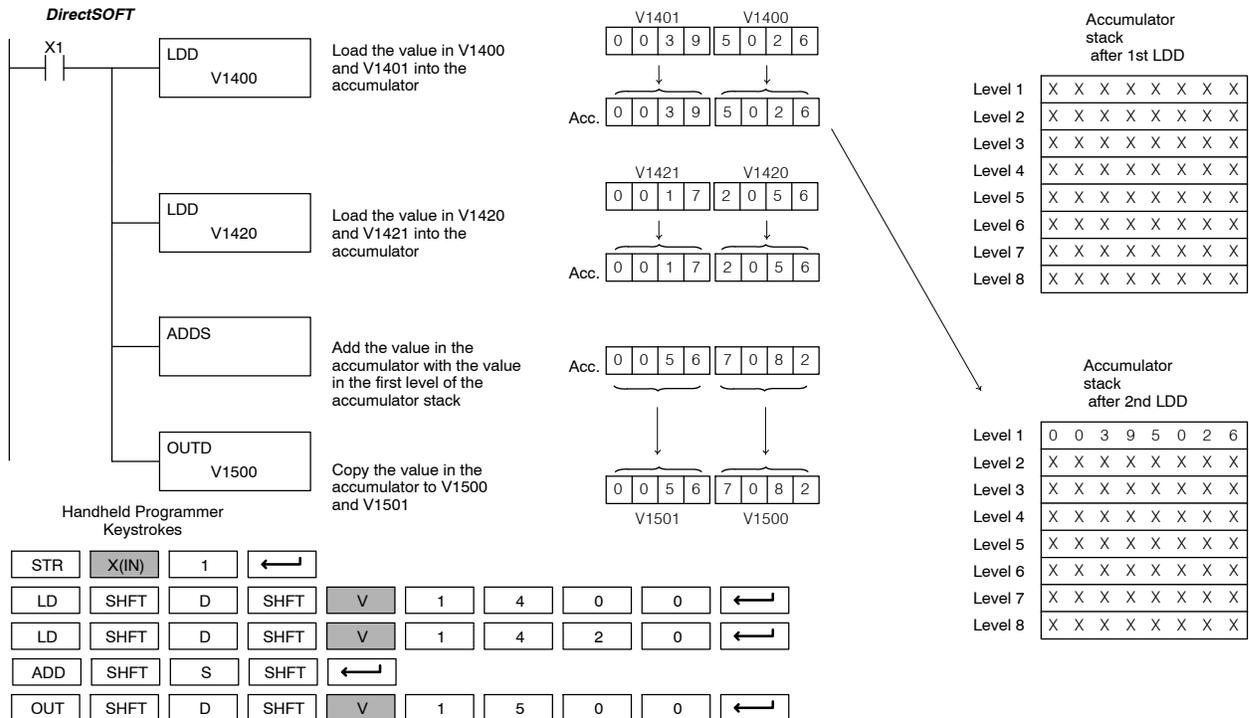
**Subtract Top
of Stack
(SUBS)**

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | | ✓ |
|---|---|---|
| **DS** | | **HPP** |

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.

SUBS

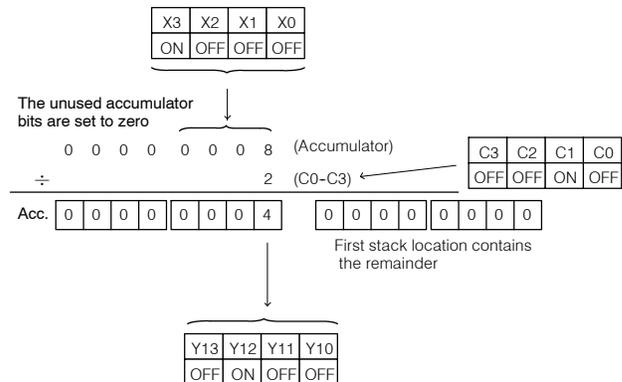| Discrete Bit Flags | Description |
|---|---|
| SP63 | On when the result of the instruction causes the value in the accumulator to be zero. |
| SP64 | On when the 16 bit subtraction instruction results in a borrow. |
| SP65 | On when the 32 bit subtraction instruction results in a borrow. |
| SP70 | 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 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.

*DirectSOFT*

X1 —| |— LDD
        V1400

Load the value in V1400 and V1401 into the accumulator

LDD
   V1420

Load the value in V1420 and V1421 into the accumulator

SUBS

Subtract the value in the first level of the accumulator stack from the value in the accumulator

OUTD
   V1500

Copy the value in the accumulator to V1500 and V1501

V1401  V1400
| 0 | 0 | 1 | 7 | 2 | 0 | 5 | 6 |

Acc.
| 0 | 0 | 1 | 7 | 2 | 0 | 5 | 6 |

V1421  V1420
| 0 | 0 | 3 | 9 | 5 | 0 | 2 | 6 |

Acc.
| 0 | 0 | 3 | 9 | 5 | 0 | 2 | 6 |

Acc.
| 0 | 0 | 2 | 2 | 2 | 9 | 7 | 0 |

| 0 | 0 | 2 | 2 | 2 | 9 | 7 | 0 |
V1501  V1500

Accumulator stack
after 1st LDD

| Level 1 | X X X X X X X X |
| Level 2 | X X X X X X X X |
| Level 3 | X X X X X X X X |
| Level 4 | X X X X X X X X |
| Level 5 | X X X X X X X X |
| Level 6 | X X X X X X X X |
| Level 7 | X X X X X X X X |
| Level 8 | X X X X X X X X |

Accumulator stack
after 2nd LDD

| Level 1 | 0 0 1 7 2 0 5 6 |
| Level 2 | X X X X X X X X |
| Level 3 | X X X X X X X X |
| Level 4 | X X X X X X X X |
| Level 5 | X X X X X X X X |
| Level 6 | X X X X X X X X |
| Level 7 | X X X X X X X X |
| Level 8 | X X X X X X X X |

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
| LD | SHFT | D | SHFT | V | 1 | 4 | 0 | 0 | ↵ |
| LD | SHFT | D | SHFT | V | 1 | 4 | 2 | 0 | ↵ |
| SUB | SHFT | S | SHFT | ↵ |
| OUT | SHFT | D | SHFT | V | 1 | 5 | 0 | 0 | ↵ |

Standard
RLL Instructions

**Multiply Top
of Stack
(MULS)**

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | | ✓ |
|---|---|---|
| **DS** | | **HPP** |

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 is removed and all stack values are moved up one level.

```
        MULS
────────┤        ├
        │        │
        └────────┘
```

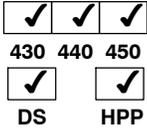| Discrete Bit Flags | Description |
|---|---|
| SP63 | On when the result of the instruction causes the value in the accumulator to be zero. |
| SP70 | On anytime the value in the accumulator is negative. |
| SP75 | On when a BCD instruction is executed and a NON–BCD number was encountered. |

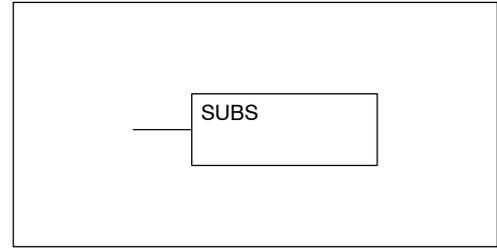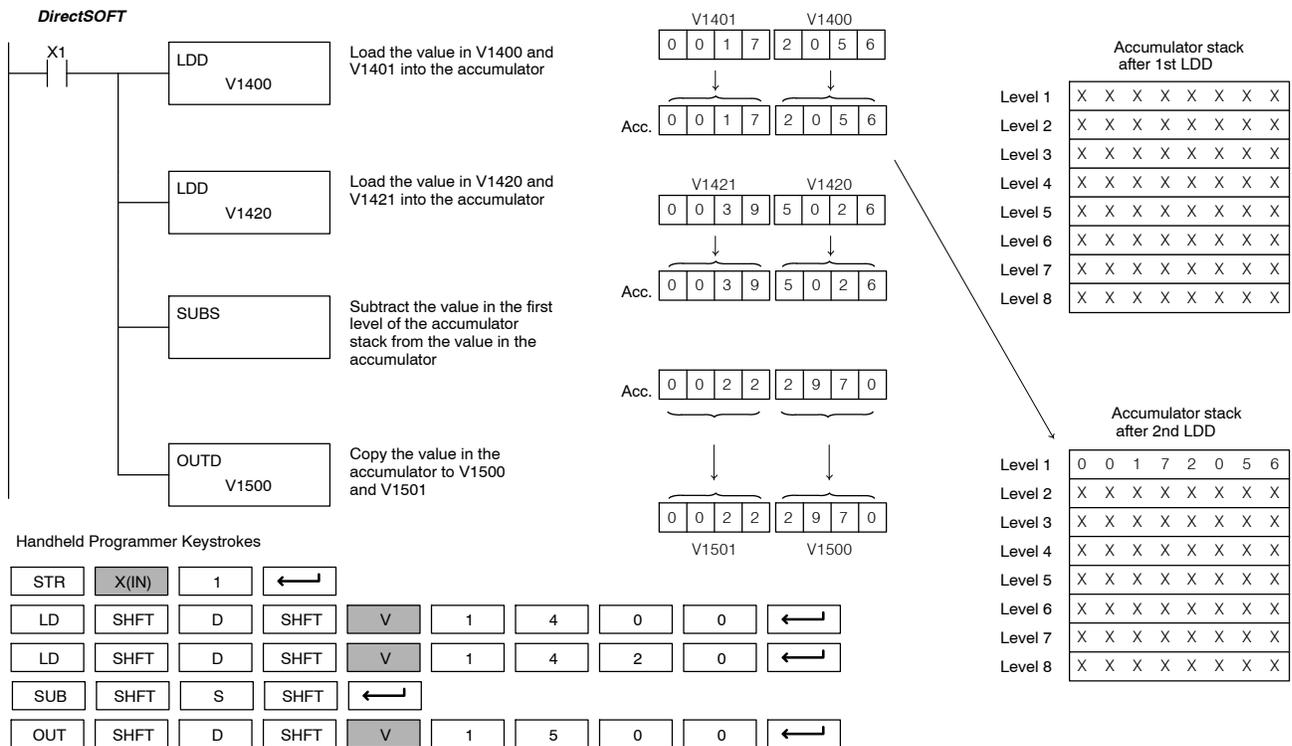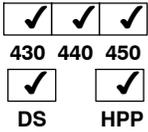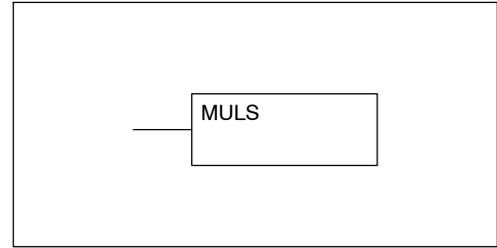**NOTE:** Status flags are valid only until another instruction uses the same flag.

In the following example, when X1 is on, the value in V1400 will be loaded into the accumulator using the Load instruction. 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 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.

*DirectSOFT*

```
  X1
──┤ ├──┬──  LD           Load the value in V1400 into
         │   V1400        the accumulator
         │
         ├──  LD           Load the value in V1420 into
         │   V1420        the accumulator
         │
         ├──  MULS         Multiply the value in the
         │                accumulator with the value
         │                in the first level of the
         │                accumulator stack
         │
         └──  OUTD         Copy the value in the
             V1500        accumulator to V1500
                          and V1501
```

V1400
| 5 | 0 | 0 | 0 |

The unused accumulator bits are set to zero

Acc. | 0 | 0 | 0 | 0 | 5 | 0 | 0 | 0 |

V1420
| 0 | 2 | 0 | 0 |

The unused accumulator bits are set to zero

Acc. | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 |

Acc. | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

| 0 | 1 | 0 | 0 | | 0 | 0 | 0 | 0 |
V1501      V1500

Accumulator stack after 1st LDD

| Level 1 | X X X X X X X X |
| Level 2 | X X X X X X X X |
| Level 3 | X X X X X X X X |
| Level 4 | X X X X X X X X |
| Level 5 | X X X X X X X X |
| Level 6 | X X X X X X X X |
| Level 7 | X X X X X X X X |
| Level 8 | X X X X X X X X |

Accumulator stack after 2nd LDD

| Level 1 | 0 0 0 0 5 0 0 0 |
| Level 2 | X X X X X X X X |
| Level 3 | X X X X X X X X |
| Level 4 | X X X X X X X X |
| Level 5 | X X X X X X X X |
| Level 6 | X X X X X X X X |
| Level 7 | X X X X X X X X |
| Level 8 | X X X X X X X X |

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
| LD | V | 1 | 4 | 0 | 0 | ↵ |
| LD | V | 1 | 4 | 2 | 0 | ↵ |
| MUL | SHFT | S | SHFT | ↵ |
| OUT | SHFT | D | SHFT | V | 1 | 5 | 0 | 0 | ↵ |

**Divide by Top of Stack (DIVS)**

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | | ✓ |
|---|---|---|
| **DS** | | **HPP** |
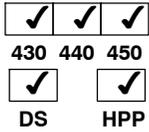
Divide Top of Stack is a 32 bit instruction that divides the 8-digit BCD value in the accumulator by a 4-digit BCD value in the first level of the accumulator stack. The result resides in the accumulator and the remainder resides in the first level of the accumulator stack.

```
        ┌──────────┐
────────┤ DIVS     │
        │          │
        └──────────┘
```

| 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 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.



*DirectSOFT*

Handheld Programmer Keystrokes

## Add Binary Top of Stack (ADDBS)

| ✕ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |
| ✓ | | ✓ |
| **DS** | | **HPP** |

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 up one level.
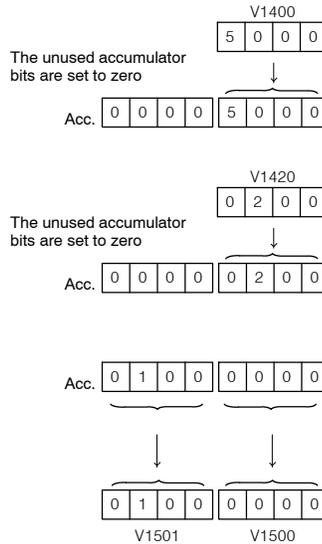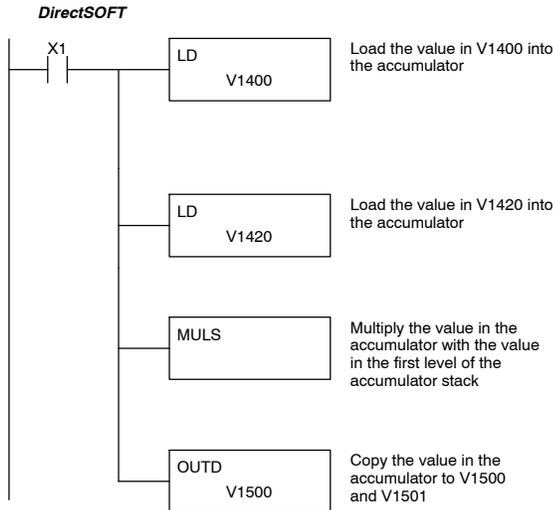
```
            ┌─────────┐
────────────┤ ADDBS   │
            └─────────┘
```

| 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 a 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.
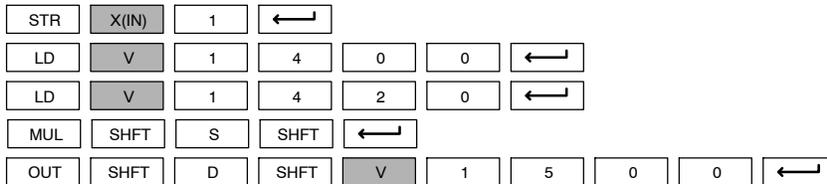
**DirectSOFT**

| | | |
|---|---|---|
| X1 | LDD V1400 | Load the value in V1400 and V1401 into the accumulator |
| | LDD V1420 | Load the value in V1420 and V1421 into the accumulator |
| | ADDBS | Add the binary value in the accumulator with the binary value in the first level of the accumulator stack |
| | OUTD V1500 | Copy the value in the accumulator to V1500 and V1501 |

V1401    V1400
| 0 | 0 | 3 | A | 5 | 0 | C | 6 |

Acc. | 0 | 0 | 3 | A | 5 | 0 | C | 6 |

V1421    V1420
| 0 | 0 | 1 | 7 | B | 0 | 5 | F |

Acc. | 0 | 0 | 1 | 7 | B | 0 | 5 | F |

Acc. | 0 | 0 | 5 | 2 | 0 | 1 | 2 | 5 |

| 0 | 0 | 5 | 2 | 0 | 1 | 2 | 5 |
V1501    V1500

Accumulator stack after 1st LDD

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Level 1 | X | X | X | X | X | X | X | X |
| Level 2 | X | X | X | X | X | X | X | X |
| Level 3 | X | X | X | X | X | X | X | X |
| Level 4 | X | X | X | X | X | X | X | X |
| Level 5 | X | X | X | X | X | X | X | X |
| Level 6 | X | X | X | X | X | X | X | X |
| Level 7 | X | X | X | X | X | X | X | X |
| Level 8 | X | X | X | X | X | X | X | X |

Accumulator stack after 2nd LDD

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Level 1 | 0 | 0 | 3 | A | 5 | 0 | C | 6 |
| Level 2 | X | X | X | X | X | X | X | X |
| Level 3 | X | X | X | X | X | X | X | X |
| Level 4 | X | X | X | X | X | X | X | X |
| Level 5 | X | X | X | X | X | X | X | X |
| Level 6 | X | X | X | X | X | X | X | X |
| Level 7 | X | X | X | X | X | X | X | X |
| Level 8 | X | X | X | X | X | X | X | X |

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ | | | | | |
|---|---|---|---|---|---|---|---|---|
| LD | SHFT | D | SHFT | V | 1 | 4 | 0 | 0 | ↵ |
| LD | SHFT | D | SHFT | V | 1 | 4 | 2 | 0 | ↵ |
| ADD | SHFT | B | S | SHFT | ↵ | | | |
| OUT | SHFT | D | SHFT | V | 1 | 5 | 0 | 0 | ↵ |

Standard RLL Instructions

**Subtract Binary
Top of Stack
(SUBBS)**

| X | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |
| | ✓ | ✓ |
| **DS** | | **HPP** |

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.

```
─────────┤ SUBBS │
         └───────┘
```

| 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 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.

*DirectSOFT*

| X1 | LDD | Load the value in V1400 and |
| | V1400 | V1401 into the accumulator |

| | LDD | Load the value in V1420 and |
| | V1420 | V1421 into the accumulator |

| | SUBBS | Subtract the binary value in the first level of the accumulator stack from the binary value in the accumulator |

| | OUTD | Copy the value in the |
| | V1500 | accumulator to V1500 and V1501 |

V1401     V1400
| 0 | 0 | 1 | A | 2 | 0 | 5 | B |

Acc. | 0 | 0 | 1 | A | 2 | 0 | 5 | B |

V1421     V1420
| 0 | 0 | 3 | A | 5 | 0 | C | 6 |

Acc. | 0 | 0 | 3 | A | 5 | 0 | C | 6 |

Acc. | 0 | 0 | 2 | 0 | 3 | 0 | 6 | B |

| 0 | 0 | 2 | 0 | 3 | 0 | 6 | B |
V1501     V1500

Accumulator stack after 1st LDD

| | |
|---|---|
| Level 1 | X X X X X X X X |
| Level 2 | X X X X X X X X |
| Level 3 | X X X X X X X X |
| Level 4 | X X X X X X X X |
| Level 5 | X X X X X X X X |
| Level 6 | X X X X X X X X |
| Level 7 | X X X X X X X X |
| Level 8 | X X X X X X X X |

Accumulator stack after 2nd LDD

| | |
|---|---|
| Level 1 | 0 0 1 A 2 0 5 B |
| Level 2 | X X X X X X X X |
| Level 3 | X X X X X X X X |
| Level 4 | X X X X X X X X |
| Level 5 | X X X X X X X X |
| Level 6 | X X X X X X X X |
| Level 7 | X X X X X X X X |
| Level 8 | X X X X X X X X |

Handheld Programmer Keystrokes

| STR | | X(IN) | | 1 | | ↵ |
| LD | SHFT | D | SHFT | V | 1 | 4 | 0 | 0 | ↵ |
| LD | SHFT | D | SHFT | V | 1 | 4 | 2 | 0 | ↵ |
| SUB | SHFT | B | S | SHFT | ↵ |
| OUT | SHFT | D | SHFT | V | 1 | 5 | 0 | 0 | ↵ |

Standard
RLL Instructions

## Multiply Binary Top of Stack (MULBS)

| X | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |
| ✓ | | ✓ |
| **DS** | | **HPP** |

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.

```
            ┌──────────┐
────────────┤  MULBS   │
            └──────────┘
```
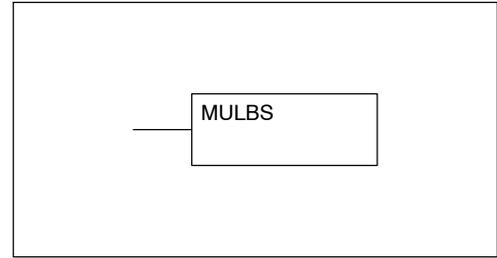
| Discrete Bit Flags | Description |
|---|---|
| SP63 | On when the result of the instruction causes the value in the accumulator to be zero. |
| SP70 | On anytime the value in the accumulator is negative. |

**NOTE:** Status flags are valid only until another instruction uses the same flag.

In the following example, when X1 is on, the 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.
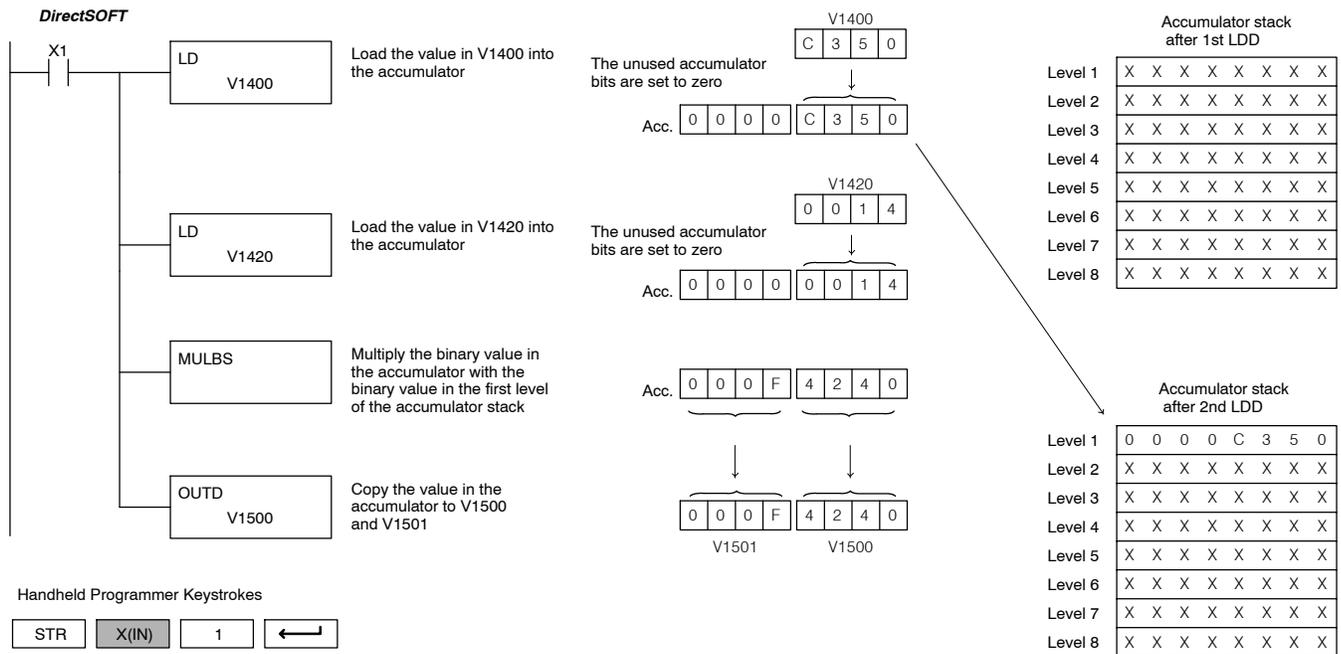


Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
| LD | V | 1 | 4 | 0 | 0 | ↵ |
| LD | V | 1 | 4 | 2 | 0 | ↵ |
| MUL | SHFT | B | S | SHFT | ↵ |
| OUT | SHFT | D | SHFT | V | 1 | 5 | 0 | 0 | ↵ |

**Divide Binary by Top OF Stack (DIVBS)**

| × | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |
| | ✓ | ✓ |
| | **DS** | **HPP** |

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.

```
      ┌──────────┐
──────┤ DIVBS    │
      └──────────┘
```

| 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 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.



*DirectSOFT*

| | |
|---|---|
| X1 — LD V1400 | Load the value in V1400 into the accumulator |
| LDD V1420 | Load the value in V1420 and V1421 into the accumulator |
| DIVBS | Divide the binary value in the accumulator by the binary value in the first level of the accumulator stack |
| OUTD V1500 | Copy the value in the accumulator to V1500 and V1501 |
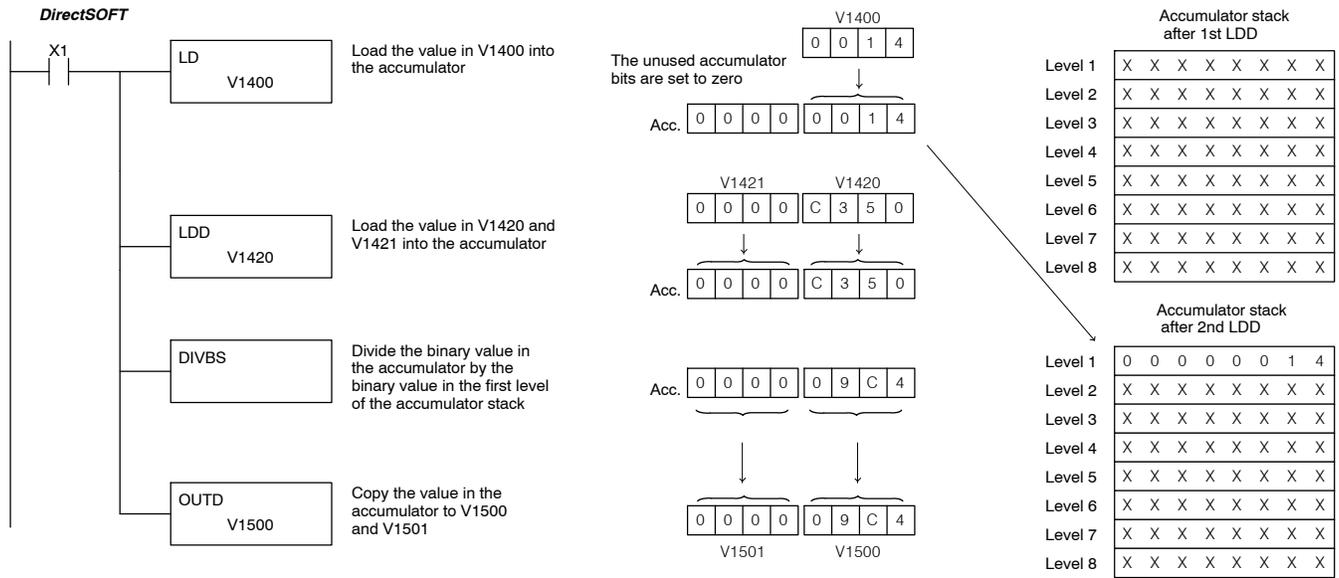
Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ | | | | |
| LD | V | 1 | 4 | 0 | 0 | ↵ | |
| LD | SHFT | D | SHFT | V | 1 | 4 | 2 | 0 | ↵ |
| DIV | SHFT | B | S | SHFT | ↵ | | |
| OUT | SHFT | D | SHFT | V | 1 | 5 | 0 | 0 | ↵ |

## Increment (INC)

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

The Increment instruction increments a BCD value in a specified V-memory location by "1" each time the instruction is executed.

```
        INC
          A aaa
```
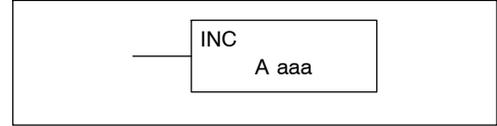
## Decrement (DEC)

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

The Decrement instruction decrements a BCD value in a specified V-memory location by "1" each time the instruction is executed.

```
        DEC
          A aaa
```

| Operand Data Type | | DL430 Range | DL440 Range | DL450 Range |
|---|---|---|---|---|
| | **A** | **aaa** | **aaa** | **aaa** |
| V-memory | V | All (See p. 3-40) | All (See p. 3-41) | All (See p. 3-42) |
| Pointer | P | All (See p. 3-40) | All (See p. 3-41) | All (See p. 3-42) |

| 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 is on the value in V1400 increases by one.

*DirectSOFT*

```
C5          INC
               V1400

        Increment the value in
        V1400 by "1".
```

V1400
| 8 | 9 | 3 | 5 |

V1400
| 8 | 9 | 3 | 6 |

Handheld Programmer Keystrokes

| STR | C(CR) | 5 | ↵ |
|---|---|---|---|

| SHFT | I | N | C | SHFT | V | 1 | 4 | 0 | 0 | ↵ |
|---|---|---|---|---|---|---|---|---|---|---|

In the following decrement example, when C5 is on the value in V1400 is decreased by one.

*DirectSOFT*

```
C5          DEC
               V1400

        Decrement the value in
        V1400 by "1".
```

V1400
| 8 | 9 | 3 | 5 |

V1400
| 8 | 9 | 3 | 4 |

Handheld Programmer Keystrokes

| STR | C(CR) | 5 | ↵ |
|---|---|---|---|

| SHFT | D | E | C | SHFT | V | 1 | 4 | 0 | 0 | ↵ |
|---|---|---|---|---|---|---|---|---|---|---|

# Transcendental Functions

The DL450 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 (RAD) instruction, then performing the trig function. All transcendental functions utilize the following flag bits.

| 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 a valid floating point number. |
| SP73 | on when a signed addition or subtraction results in a incorrect sign bit. |
| SP75 | On when a real number instruction is executed and a non-real number was encountered. |

| Math Function | Range of Argument |
|---|---|
| SP53 | On when the value of the operand is larger than the accumulator can work with. |

**Sine Real (SINR)**

| × | × | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | | × |
|---|---|---|
| **DS** | | **HPP** |

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 are in IEEE 32-bit format.

```
        ┌──────┐
────────┤ SINR │
        │      │
        └──────┘
```
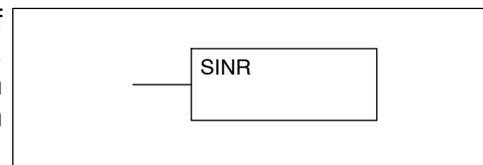
**Cosine Real (COSR)**

| × | × | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | | × |
|---|---|---|
| **DS** | | **HPP** |

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 are in IEEE 32-bit format.

```
        ┌──────┐
────────┤ COSR │
        │      │
        └──────┘
```
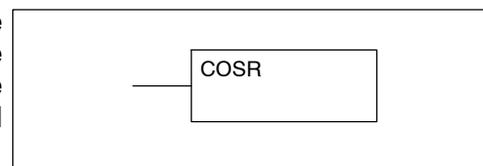
**Tangent Real (TANR)**

| × | × | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | | × |
|---|---|---|
| **DS** | | **HPP** |

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 are in IEEE 32-bit format.

```
        ┌──────┐
────────┤ SINR │
        │      │
        └──────┘
```
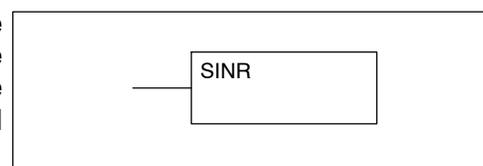
**Arc Sine Real (ASINR)**

| × | × | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | | × |
|---|---|---|
| **DS** | | **HPP** |

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 are in IEEE 32-bit format.

```
        ┌───────┐
────────┤ ASINR │
        │       │
        └───────┘
```
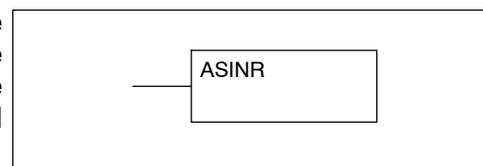
**Arc Cosine Real (ACOSR)**

| ✗ | ✗ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | | ✗ |
|---|---|---|
| **DS** | | **HPP** |

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 are in IEEE 32-bit format.

```
┌──────────┐
│  ACOSR   │
│          │
└──────────┘
```
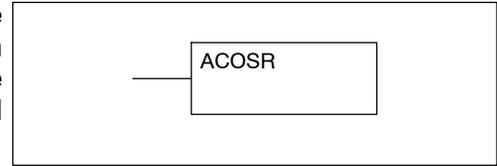
**Arc Tangent Real (ATANR)**

| ✗ | ✗ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | | ✗ |
|---|---|---|
| **DS** | | **HPP** |

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 are in IEEE 32-bit format.

```
┌──────────┐
│  ATANR   │
│          │
└──────────┘
```
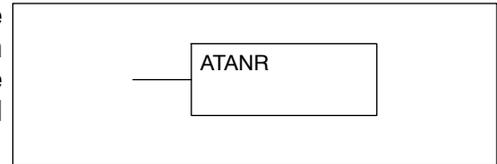
**Square Root Real (SQRTR)**

| ✗ | ✗ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | | ✗ |
|---|---|---|
| **DS** | | **HPP** |

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 are in IEEE 32-bit format.

```
┌──────────┐
│  SQRTR   │
│          │
└──────────┘
```
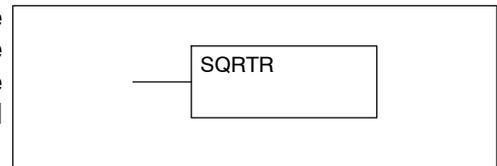
**NOTE:** The square root function can be useful in several situations. However, if you are trying to do the square-root extract function for an orifice flow meter measurement as the PV to a PID loop, note that the PID loop already has the square-root extract function built in.

The following example takes the **sine** of 45 degrees. Since these transcendental functions operate only on real numbers, we do a 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.

Accumulator contents
(viewed as real number)

*DirectSOFT*

| Ladder | Description | Accumulator |
|--------|-------------|-------------|
| X1 —[ ]— LDR R45 | Load the real number 45 into the accumulator. | 45.000000 |
| RADR | Convert the degrees into radians, leaving the result in the accumulator. | 0.7358981 |
| SINR | Take the sine of the number in the accumulator, which is in radians. | 0.7071067 |
| OUTD V2000 | Copy the value in the accumulator to V2000 and V2001. | 0.7071067 |

**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.

Standard
RLL Instructions

**Increment Binary (INCB)**

✓ ✓ ✓
**430 440 450**
✓ | ✓
**DS HPP**
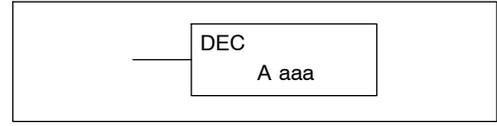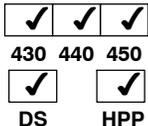
The Increment Binary instruction increments a binary value in a specified V-memory location by "1" each time the instruction is executed.

```
        ┌─────────────┐
────────┤ INCB        │
        │     A aaa   │
        └─────────────┘
```
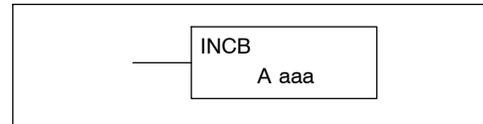
**Decrement Binary (DECB)**

✓ ✓ ✓
**430 440 450**
✓ | ✓
**DS HPP**
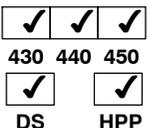
The Decrement Binary instruction decrements a binary value in a specified V-memory location by "1" each time the instruction is executed.

```
        ┌─────────────┐
────────┤ DECB        │
        │     A aaa   │
        └─────────────┘
```
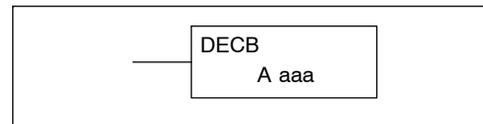
| Operand Data Type | | DL430 Range | DL440 Range | DL450 Range |
|---|---|---|---|---|
| | **A** | **aaa** | **aaa** | **aaa** |
| V-memory | V | All (See p. 3–40) | All (See p. 3–40) | All (See p. 3–42) |
| Pointer | P | All (See p. 3–40) | All (See p. 3–41) | All (See p. 3–42) |

| 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 increment binary example, when C5 is on the binary value in V1400 is increased by one.

*DirectSOFT*

```
 C5          ┌─────────────┐
─┤ ├─────────┤ INCB        │
            │     V1400    │
            └─────────────┘
```
Increment the binary value in V1400 by "1".

V1400
| 4 | A | 3 | C |
↓
V1400
| 4 | A | 3 | D |

Handheld Programmer Keystrokes

| STR | C(CR) | 5 | ↵ |

| SHFT | I | N | C | B | SHFT | V | 1 | 4 | 0 | 0 | ↵ |
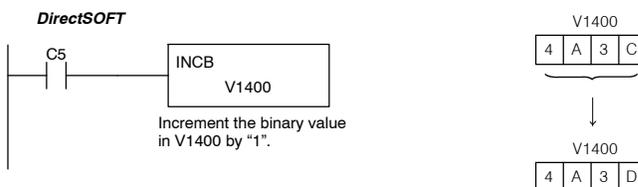
In the following decrement binary example, when C5 is on the value in V1400 is decreased by one.

*DirectSOFT*

```
 C5          ┌─────────────┐
─┤ ├─────────┤ DECB        │
            │     V1400    │
            └─────────────┘
```
Decrement the binary value in V1400 by "1".

V1400
| 4 | A | 3 | C |
↓
V1400
| 4 | A | 3 | B |

Handheld Programmer Keystrokes

| STR | C(CR) | 5 | ↵ |

| SHFT | D | E | C | B | SHFT | V | 1 | 4 | 0 | 0 | ↵ |
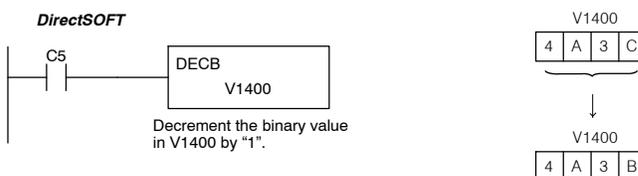
# Bit Operation Instructions

**Sum
(SUM)**

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | | ✓ |
|---|---|---|
| **DS** | | **HPP** |

The Sum instruction counts number of bits that are set to "1" in the accumulator. The HEX result resides in the accumulator.

```
        ┌──────────┐
────────┤   SUM    │
        └──────────┘
```
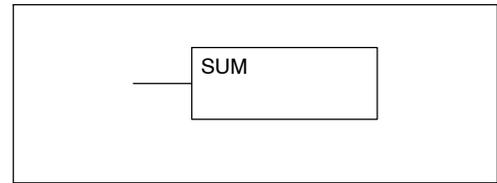
In the following example, when X1 is on, the value formed by discrete locations X10–X17 is loaded into the accumulator using the Load Formatted instruction. The number of bits in the accumulator set to "1" is counted using the Sum instruction. The value in the accumulator is copied to V1500 using the Out instruction.

*DirectSOFT*

```
    X1          ┌──────────────┐
────┤ ├─────────┤ LDF      X10  │
                │          K8   │
                └──────────────┘
```
Load the value represented by
discrete locations X10–X17
into the accumulator

| X17 | X16 | X15 | X14 | X13 | X12 | X11 | X10 |
|-----|-----|-----|-----|-----|-----|-----|-----|
| ON | ON | OFF | OFF | ON | OFF | ON | ON |

The unused accumulator
bits are set to zero

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

Acc.

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 |

```
            ┌──────────────┐
────────────┤ SUM          │
            └──────────────┘
```
Sum the number of bits in
the accumulator set to "1"

Acc.

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 5 |

```
            ┌──────────────┐
────────────┤ OUT          │
            │      V1500    │
            └──────────────┘
```
Copy the value in the lower
16 bits of the accumulator
to V1500

| 0 | 0 | 0 | 5 |
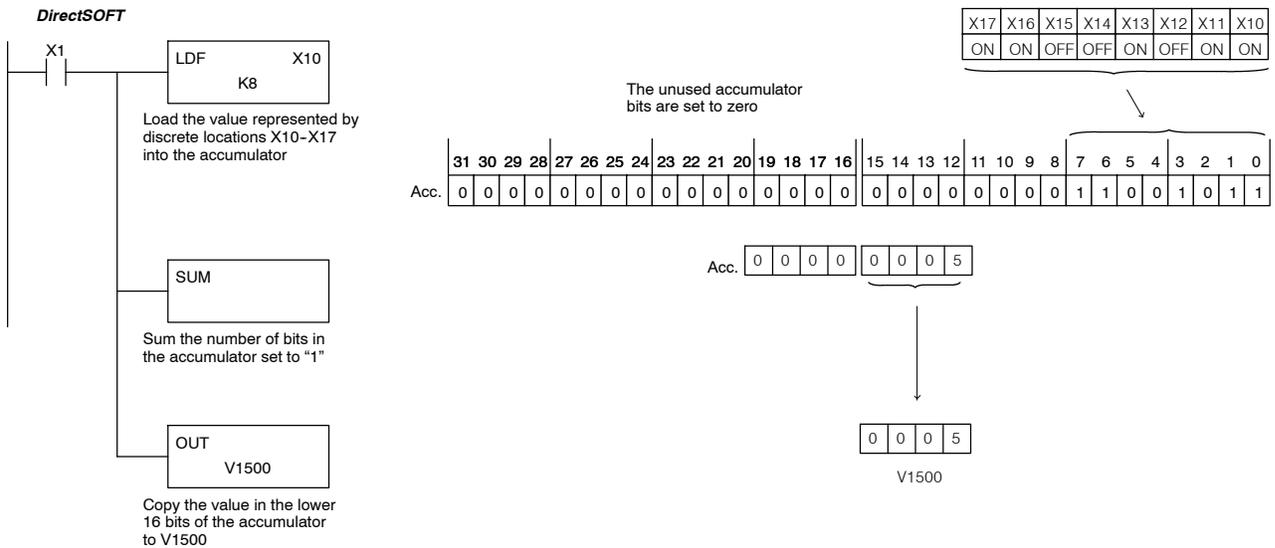
V1500

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ← |
|-----|-------|---|---|

| LD | SHFT | F | X(IN) | 1 | 0 | K(CON) | 8 | ← |
|----|------|---|-------|---|---|--------|---|---|

| SHFT | S | U | M | ← |
|------|---|---|---|---|

| OUT | V | 1 | 5 | 0 | 0 | ← |
|-----|---|---|---|---|---|---|

**Shift Left
(SHFL)**

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

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 lost.

```
        ┌──────────────┐
────────┤ SHFL         │
        │    A aaa     │
        └──────────────┘
```
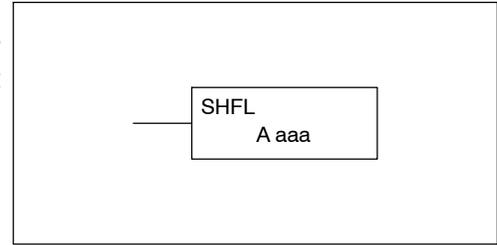
| Operand Data Type | | DL430 Range | DL440 Range | DL450 Range |
|---|---|---|---|---|
| | **A** | **aaa** | **aaa** | **aaa** |
| V-memory | V | All (See p. 3-40) | All (See p. 3-41) | All (See p. 3-42) |
| Constant | K | 1-32 | 1-32 | 1-32 |

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 shifted 2 bits to the left using the Shift Left instruction. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.

*DirectSOFT*

```
  X1        ┌──────────┐
──┤ ├───────┤ LDD      │
            │   V1400  │
            └──────────┘
```
Load the value in V1400 and V1401 into the accumulator

```
            ┌──────────┐
            │ SHFL     │
            │     K2   │
            └──────────┘
```
The bit pattern in the accumulator is shifted 2 bit positions to the left

```
            ┌──────────┐
            │ OUTD     │
            │   V1500  │
            └──────────┘
```
Copy the value in the accumulator to V1500 and V1501

V1401    V1400
6 7 0 5  3 1 0 1

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

Acc.

Shifted out of the accumulator

. . . . .

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |

Acc.

9 C 1 4
V1501

C 4 0 4
V1500
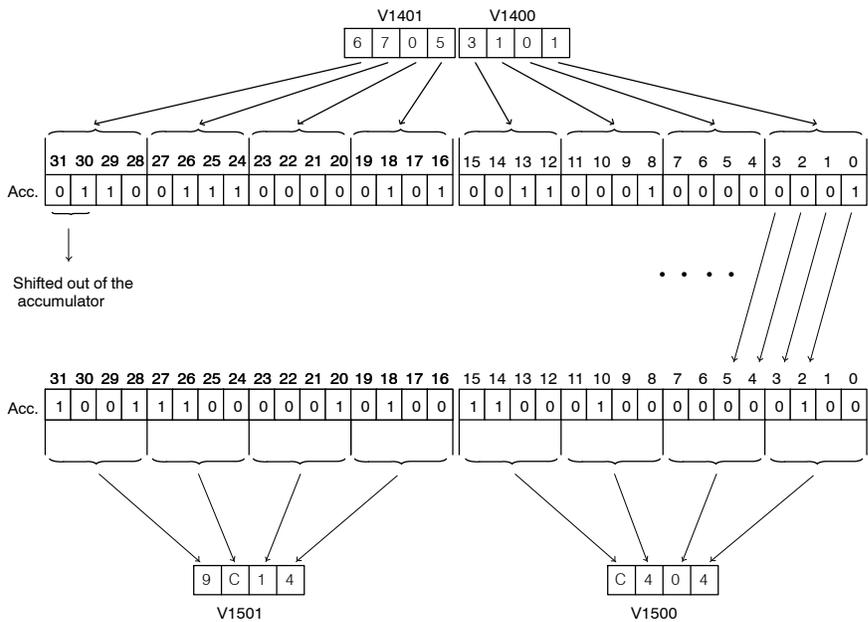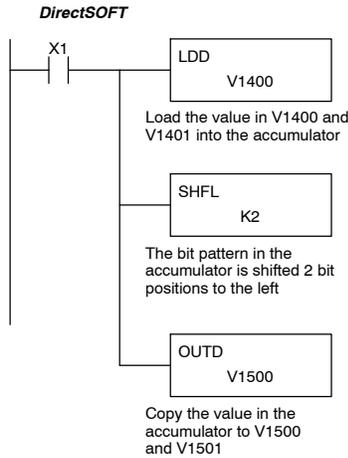
Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
|---|---|---|---|

| LD | SHFT | D | SHFT | V | 1 | 4 | 0 | 0 | ↵ |
|---|---|---|---|---|---|---|---|---|---|

| SHFT | S | H | F | L | SHFT | K(CON) | 2 | ↵ |
|---|---|---|---|---|---|---|---|---|

| OUT | SHFT | D | SHFT | V | 1 | 5 | 0 | 0 | ↵ |
|---|---|---|---|---|---|---|---|---|---|

## Shift Right (SHFR)

| | | |
|---|---|---|
| ✓ | ✓ | ✓ |
| **430** | **440** | **450** |
| ✓ | | ✓ |
| **DS** | | **HPP** |

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.

```
        SHFR
        A aaa
```
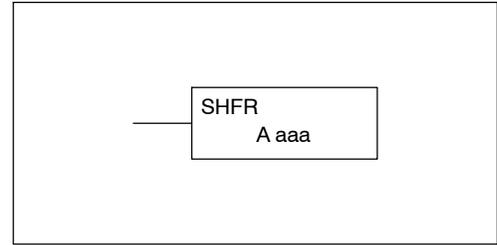
| Operand Data Type | | DL430 Range | DL440 Range | DL450 Range |
|---|---|---|---|---|
| | **A** | **aaa** | **aaa** | **aaa** |
| V–memory | V | All (See p. 3–40) | All (See p. 3–41) | All (See p. 3–42) |
| Constant | K | 1–32 | 1–32 | 1–32 |

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 shifted 2 bits to the right using the Shift Right instruction. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.



*DirectSOFT*

```
 X1
──┤ ├──────┌─ LDD ──────┐
           │   V1400     │
           └─────────────┘
Load the value in V1400 and
V1401 into the accumulator

           ┌─ SHFR ──────┐
           │   K2        │
           └─────────────┘
The bit pattern in the
accumulator is shifted 2
bit positions to the right

           ┌─ OUTD ──────┐
           │   V1500     │
           └─────────────┘
Copy the value in the
accumulator to V1500
and V1501
```
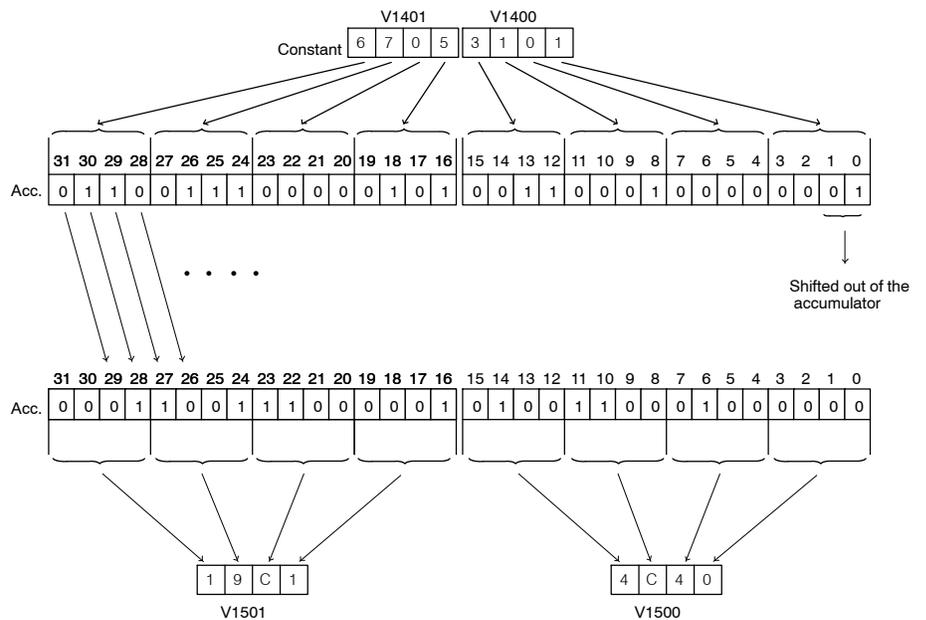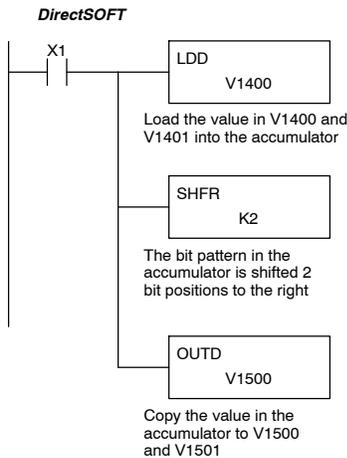
Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ← |
| LD | SHFT | D | SHFT | V | 1 | 4 | 0 | 0 | ← |
| SHFT | S | H | F | R | SHFT | K(CON) | 2 | ← |
| OUT | SHFT | D | SHFT | V | 1 | 5 | 0 | 0 | ← |

**Rotate Left (ROTL)**

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

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  │
        └──────────┘
```
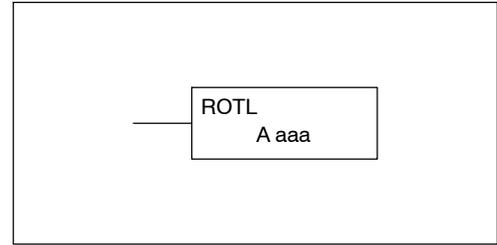
| Operand Data Type | | DL430 Range | DL440 Range | DL450 Range |
|---|---|---|---|---|
| | **A** | **aaa** | **aaa** | **aaa** |
| V–memory | V | All (See p. 3–40) | All (See p. 3–41) | All (See p. 3–42) |
| Constant | K | 1–32 | 1–32 | 1–32 |

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.

*DirectSOFT*

```
    X1        ┌──────────┐
────┤├────────┤ LDD      │
              │   V1400  │
              └──────────┘
```
Load the value in V1400 and V1401 into the accumulator

```
              ┌──────────┐
              │ ROTL     │
              │   K2     │
              └──────────┘
```
The bit pattern in the accumulator is rotated 2 bit positions to the left

```
              ┌──────────┐
              │ OUTD     │
              │   V1500  │
              └──────────┘
```
Copy the value in the accumulator to V1500 and V1501
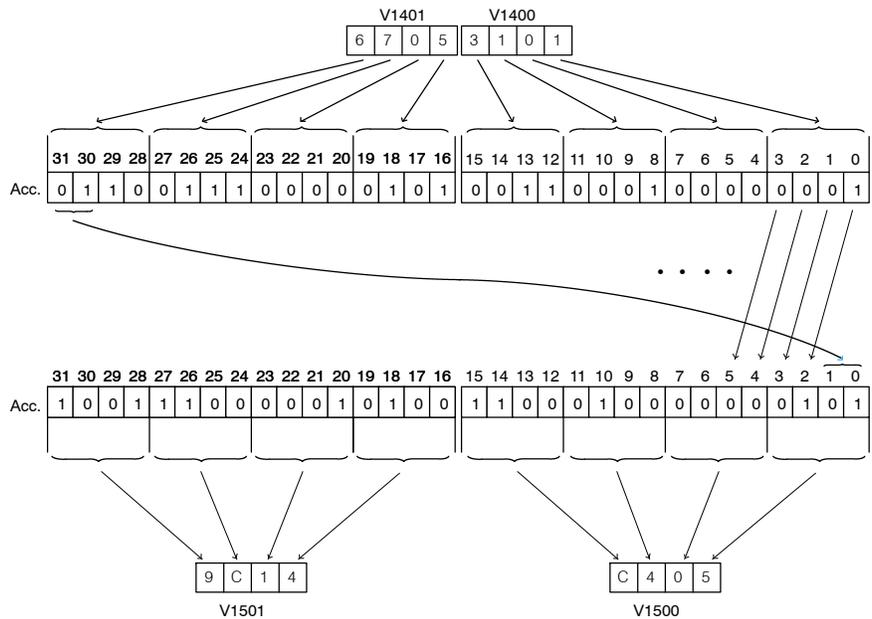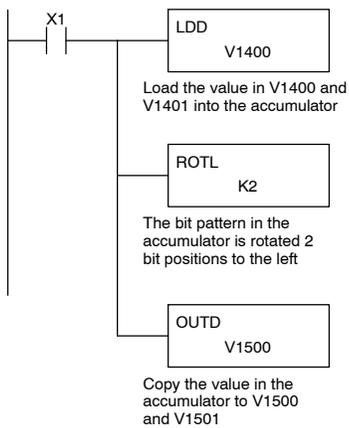


Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
| LD | SHFT | D | SHFT | V | 1 | 4 | 0 | 0 | ↵ |
| SHFT | R | O | T | L | SHFT | K(CON) | 2 | ↵ |
| OUT | SHFT | D | SHFT | V | 1 | 5 | 0 | 0 | ↵ |

## Rotate Right (ROTR)

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

Rotate Right is a 32 bit instruction that rotates the bits in the accumulator a specified number (Aaaa) of places to the right.

```
        ┌──────────┐
────────┤ ROTR     │
        │   A aaa  │
        └──────────┘
```
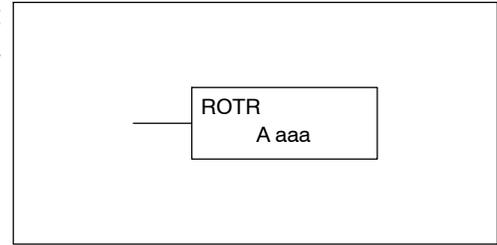
| Operand Data Type | | DL430 Range | DL440 Range | DL450 Range |
|---|---|---|---|---|
| | **A** | **aaa** | **aaa** | **aaa** |
| V-memory | V | All (See p. 3-40) | All (See p. 3-41) | All (See p. 3-42) |
| Constant | K | 1-32 | 1-32 | 1-32 |

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**

```
X1
─┤ ├──────┤ LDD       │
              V1400
```
Load the value in V1400 and V1401 into the accumulator

```
      ┌──────────┐
      │ ROTR     │
      │    K2    │
      └──────────┘
```
The bit pattern in the accumulator is rotated 2 bit positions to the right

```
      ┌──────────┐
      │ OUTD     │
      │   V1500  │
      └──────────┘
```
Copy the value in the accumulator to V1500 and V1501

V1401 V1400
6 7 0 5 3 1 0 1

Acc. 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 | 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 1 1 0 0 1 1 1 0 0 0 0 0 1 0 1 0 0 1 1 0 0 0 1 0 0 0 0 0 0 0 1

· · · · ·

Acc. 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 | 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 1 0 1 1 0 0 1 1 1 0 0 0 0 0 1 0 1 0 0 1 1 0 0 0 1 0 0 0 0 0 0

5 9 C 1
V1501

4 C 4 0
V1500
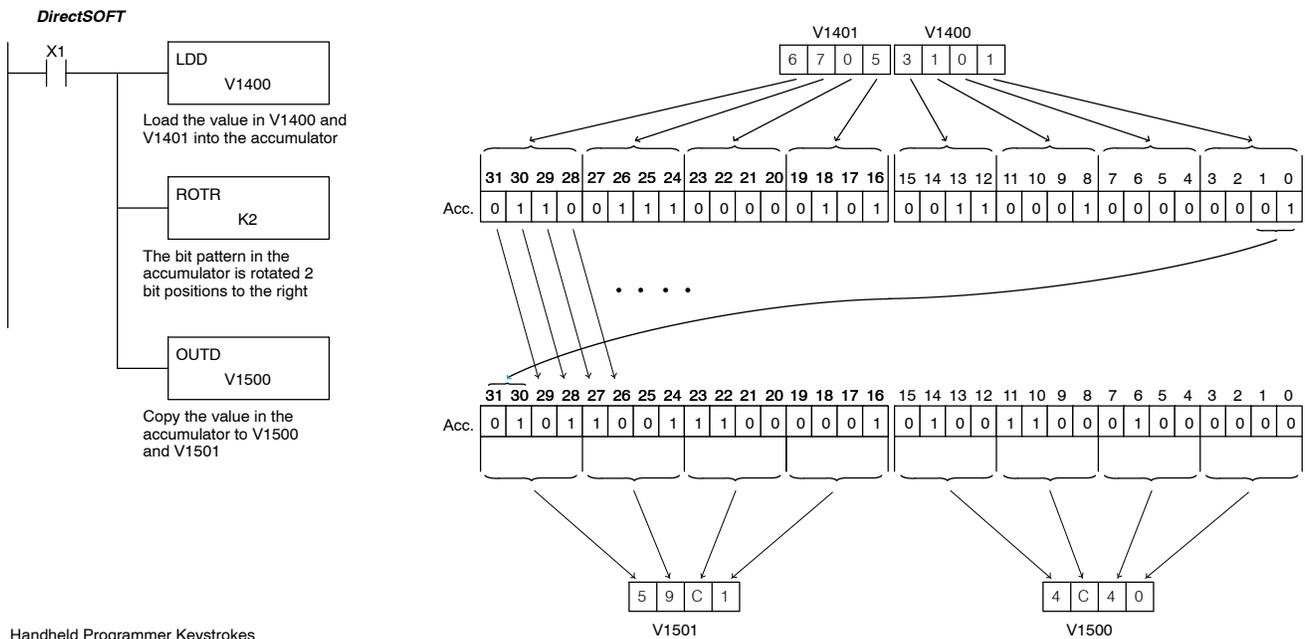
Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ← |
| LD | SHFT | D | SHFT | V | 1 | 4 | 0 | 0 | ← |
| SHFT | R | O | T | R | SHFT | K(CON) | 2 | ← |
| OUT | SHFT | D | SHFT | V | 1 | 5 | 0 | 0 | ← |

Standard
RLL Instructions

## Encode
## (ENCO)

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

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. SP53 will also be set on when the accumulator is equal to zero.
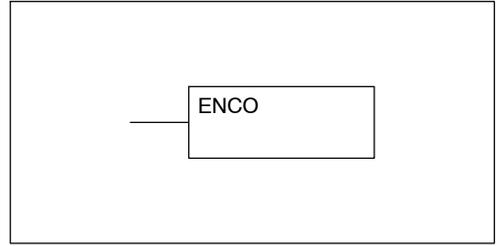
```
            ┌─────────────┐
────────────┤  ENCO       │
            │             │
            └─────────────┘
```

| Discrete Bit Flags | Description |
|---|---|
| SP53 | On when the value of the operand is larger than the accumulator can work with. SP53 will also come on when the accumulator is zero. |

**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 V1400 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 V1500 using the Out instruction.

*DirectSOFT*

```
   X1        ┌──────────────┐
───┤ ├───────┤ LD           │
              │      V1400   │
              └──────────────┘
```
Load the value in V1400 into the lower 16 bits of the accumulator

```
              ┌──────────────┐
              │ ENCO         │
              │              │
              └──────────────┘
```
Encode the bit position set to "1" in the accumulator to a 5 bit binary value

```
              ┌──────────────┐
              │ OUT          │
              │      V1500   │
              └──────────────┘
```
Copy the value in the lower 16 bits of the accumulator to V1500

V1400

| 1 | 0 | 0 | 0 |
|---|---|---|---|

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Acc.

Bit postion 12 is converted to binary

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |

Acc.

| 0 | 0 | 0 | C |
|---|---|---|---|

V1500   Binary value for 12.

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
|---|---|---|---|

| LD | V | 1 | 4 | 0 | 0 | ↵ |
|---|---|---|---|---|---|---|

| SHFT | E | N | C | O | ↵ |
|---|---|---|---|---|---|

| OUT | V | 1 | 5 | 0 | 0 | ↵ |
|---|---|---|---|---|---|---|

Standard RLL Instructions

## Decode (DECO)

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

The Decode instruction decodes a 5 bit binary value of 0–31 (0–1F HEX) in the accumulator by setting the appropriate bit position to a 1. If the accumulator contains the value F (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.

```
        ┌──────────┐
────────┤ DECO     │
        │          │
        └──────────┘
```
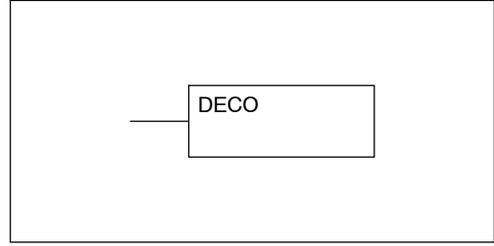
In the following example when X1 is on, the value formed by discrete locations X10–X14 is loaded into the accumulator using the Load Formatted instruction. The five bit binary pattern in the accumulator is decoded by setting the corresponding bit position to a "1" using the Decode instruction.

*DirectSOFT*

```
  X1
──┤ ├──────────┤ LDF        X10 │
                │            K5  │
                └────────────────┘
```

Load the value in represented by discrete locations X10–X14 into the accumulator

| X14 | X13 | X12 | X11 | X10 |
|-----|-----|-----|-----|-----|
| OFF | ON | OFF | ON | ON |

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Acc. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 |

The binary value is converted to bit position 11.

```
        ┌──────────┐
────────┤ DECO     │
        │          │
        └──────────┘
```

Decode the five bit binary pattern in the accumulator and set the corresponding bit position to a "1"

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Acc. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
|-----|-------|---|---|

| LD | SHFT | F | SHFT | X(IN) | 1 | 0 | SHFT | K(CON) | 5 | ↵ |
|----|------|---|------|-------|---|---|------|--------|---|---|

| SHFT | D | E | C | O | ↵ |
|------|---|---|---|---|---|

# Number Conversion Instructions

**Binary (BIN)**

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

The Binary instruction converts a BCD value in the accumulator to the equivalent binary value. The result resides in the accumulator.
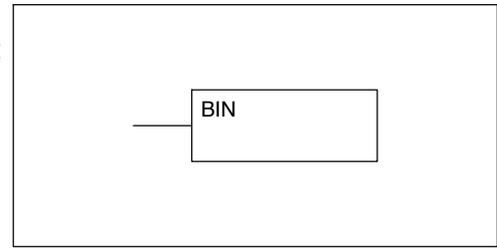
```
        ┌──────┐
────────┤ BIN  │
        │      │
        └──────┘
```

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 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 V1500 and V1501 using the Out Double instruction. The handheld programmer would display the binary value in V1500 and V1501 as a HEX value.

**DirectSOFT**

```
   X1        ┌────────┐
───┤ ├───────┤ LDD    │
             │ V1400  │
             └────────┘
```
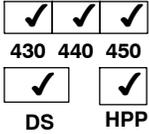Load the value in V1400 and V1401 into the accumulator

```
             ┌────────┐
─────────────┤ BIN    │
             │        │
             └────────┘
```
Convert the BCD value in the accumulator to the binary equivalent value

```
             ┌────────┐
─────────────┤ OUTD   │
             │ V1500  │
             └────────┘
```
Copy the binary value in the accumulator to V1500 and V1501

V1401   V1400

| 0 | 0 | 0 | 2 | 8 | 5 | 2 | 9 |
|---|---|---|---|---|---|---|---|

BCD Value

| | 8 4 2 1 | 8 4 2 1 | 8 4 2 1 | 8 4 2 1 | 8 4 2 1 | 8 4 2 1 | 8 4 2 1 | 8 4 2 1 |
|---|---|---|---|---|---|---|---|---|
| Acc. | 0 0 0 0 | 0 0 0 0 | 0 0 0 0 | 0 0 1 0 | 1 0 0 0 | 0 1 0 1 | 0 0 1 0 | 1 0 0 1 |

$28529 = 16384 + 8192 + 2048 + 1024 + 512 + 256 + 64 + 32 + 16 + 1$

Binary Equivalent Value

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Acc. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| | 2147483648 | 1073741824 | 536870912 | 268435456 | 134217728 | 67108864 | 33554432 | 16777216 | 8388608 | 4194304 | 2097152 | 1048576 | 524288 | 262144 | 131072 | 65536 | 32768 | 16384 | 8192 | 4096 | 2048 | 1024 | 512 | 256 | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |

| 0 | 0 | 0 | 0 | 6 | F | 7 | 1 |
|---|---|---|---|---|---|---|---|

V1501   V1500

The binary (HEX) value copied to V1500

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
|---|---|---|---|

| LD | SHFT | D | SHFT | V | 1 | 4 | 0 | 0 | ↵ |
|---|---|---|---|---|---|---|---|---|---|

| BIN | ↵ |
|---|---|

| OUT | SHFT | D | SHFT | V | 1 | 5 | 0 | 0 | ↵ |
|---|---|---|---|---|---|---|---|---|---|

Standard RLL Instructions

## Binary Coded Decimal (BCD)

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

The Binary Coded Decimal instruction converts a binary value in the accumulator to a BCD value. The result resides in the accumulator.

```
        ┌──────────┐
────────┤   BCD    │
        │          │
        └──────────┘
```
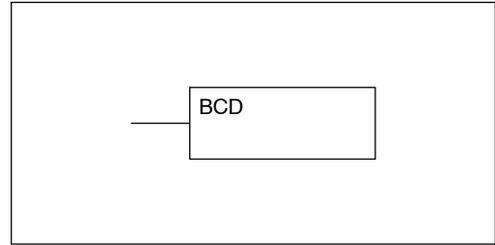
In the following example, when X1 is on, the binary (HEX) value in V1400 and V1401 is loaded into the accumulator using the Load Double instruction. The binary value in the accumulator is converted to the BCD equivalent value using the BCD instruction. The BCD value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.

**DirectSOFT**

```
  X1          ┌──────────┐
──┤ ├─────────┤  LDD     │
              │  V1400   │
              └──────────┘
```
Load the value in V1400 and V1401 into the accumulator

```
              ┌──────────┐
              │  BCD     │
              │          │
              └──────────┘
```
Convert the binary value in the accumulator to the BCD equivalent value

```
              ┌──────────┐
              │  OUTD    │
              │  V1500   │
              └──────────┘
```
Copy the BCD value in the accumulator to V1500 and V1501

V1401    V1400

| 0 | 0 | 0 | 0 | 6 | F | 7 | 1 |

Binary Value

16384 + 8192 + 2048 + 1024 + 512 + 256 + 64 + 32 + 16 + 1 = 28529

BCD Equivalent Value

| 0 | 0 | 0 | 2 | 8 | 5 | 2 | 9 |

The BCD value copied to V1500 and V1501

V1501    V1500

**Handheld Programmer Keystrokes**

| STR | X(IN) | 1 | ↵ |
|-----|-------|---|---|

| LD | SHFT | D | SHFT | V | 1 | 4 | 0 | 0 | ↵ |
|----|------|---|------|---|---|---|---|---|---|

| BCD | ↵ |
|-----|---|

| OUT | SHFT | D | SHFT | V | 1 | 5 | 0 | 0 | ↵ |
|-----|------|---|------|---|---|---|---|---|---|

**Invert (INV)**

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

The Invert instruction inverts or takes the one's complement of the 32 bit value in the accumulator. The result resides in the accumulator.
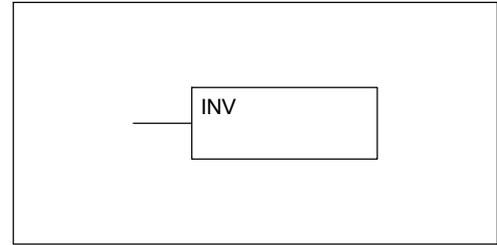
```
        INV
```

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 inverted using the Invert instruction. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.

*DirectSOFT*

```
  X1
--| |---------[ LDD        ]
               [      V1400 ]
```
Load the value in V1400 and V1401 into the accumulator

```
        --------[ INV       ]
```
Invert the binary bit pattern in the accumulator

```
        --------[ OUTD      ]
                [     V1500  ]
```
Copy the value in the accumulator to V1500 and V1501

V1401    V1400

| 0 | 4 | 0 | 5 | 0 | 2 | 5 | 0 |

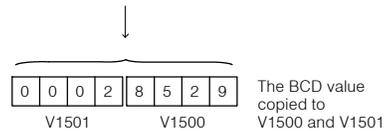| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| Acc. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| Acc. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |

| F | B | F | A | F | D | A | F |
|---|---|---|---|---|---|---|---|

V1501       V1500
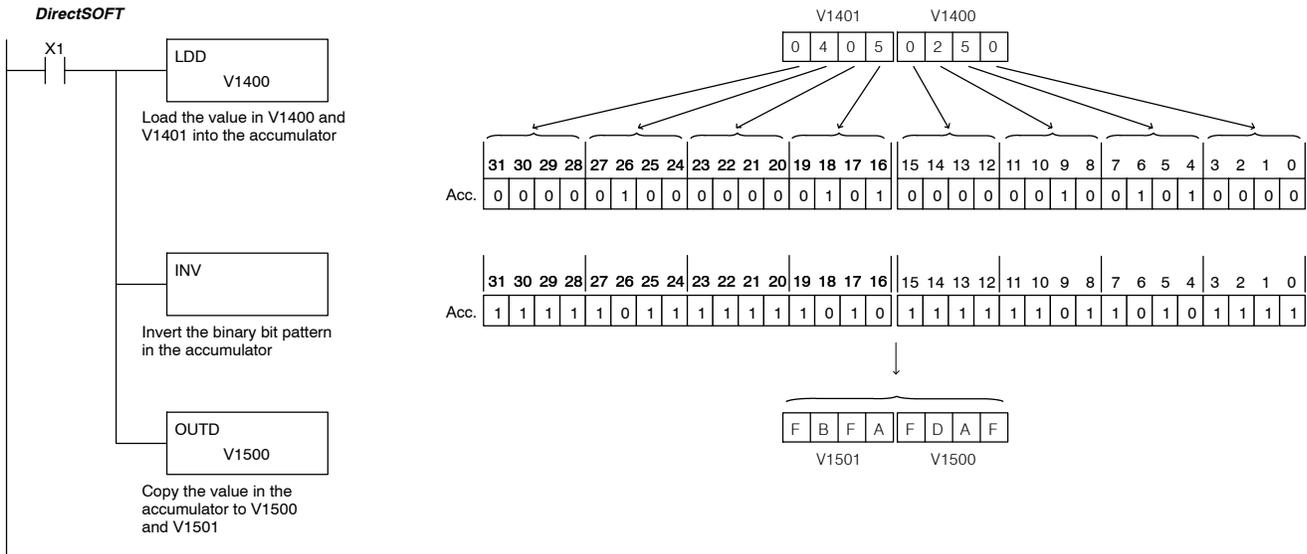
Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
|---|---|---|---|

| LD | SHFT | D | SHFT | V | 1 | 4 | 0 | 0 | ↵ |
|---|---|---|---|---|---|---|---|---|---|

| SHFT | I | N | V | ↵ |
|---|---|---|---|---|

| OUT | SHFT | D | SHFT | V | 1 | 5 | 0 | 0 | ↵ |
|---|---|---|---|---|---|---|---|---|---|

## Ten's Complement (BCDCPL)

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

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 :

$$\begin{array}{r} 100000000 \\ - \text{ accumulator value} \\ \hline \text{10's complement value} \end{array}$$
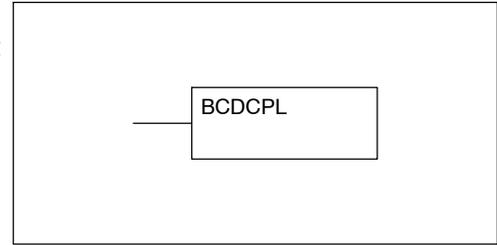
```
        ┌──────────┐
────────┤ BCDCPL   │
        └──────────┘
```

In the following example when X1 is on, the value in V1400 and V1401 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 V1500 and V1501 using the Out Double instruction.

**DirectSOFT**

```
  X1        ┌──────────┐
──┤ ├───────┤ LDD      │
            │     V1400 │
            └──────────┘
  Load the value in V1400 and
  V1401 into the accumulator

            ┌──────────┐
────────────┤ BCDCPL   │
            └──────────┘
  Takes a 10's complement of
  the value in the accumulator

            ┌──────────┐
────────────┤ OUTD     │
            │     V1500 │
            └──────────┘
  Copy the value in the
  accumulator to V1500
  and V1501
```

| V1401 | V1400 |
|-------|-------|
| 0 0 0 0 | 0 0 8 7 |

Acc. | 0 0 0 0 | 0 0 8 7 |

Acc. | 9 9 9 9 | 9 9 1 3 |

| 9 9 9 9 | 9 9 1 3 |
| V1501 | V1500 |
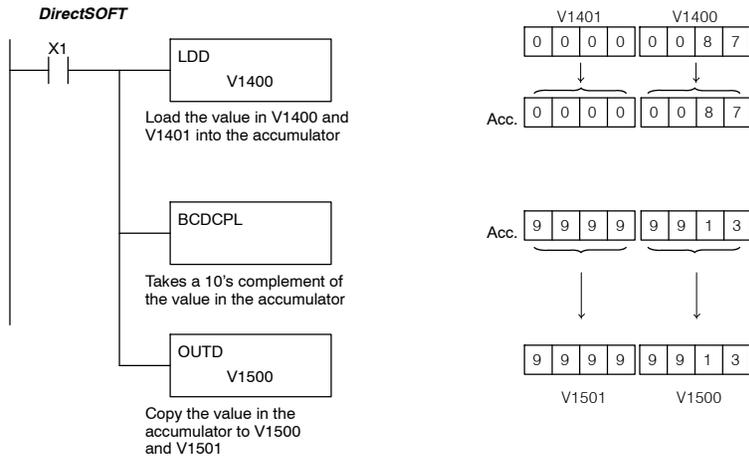
**Handheld Programmer Keystrokes**

| STR | X(IN) | 1 | ↵ |
|-----|-------|---|---|

| LD | SHFT | D | SHFT | V | 1 | 4 | 0 | 0 | ↵ |

| BCD | SHFT | C | P | L | ↵ |

| OUT | SHFT | D | SHFT | V | 1 | 5 | 0 | 0 | ↵ |

**NOTE:** If your program has a subtraction calculation which results in a borrowed digit (noted by the status flag), the BCDCPL instruction can be used to find the absolute difference between the two values in the subtraction.

Standard
RLL Instructions

**Binary to Real Conversion (BTOR)**

| × | × | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

The Binary-to-Real instruction converts a binary 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

```
        ┌──────────┐
────────┤  BTOR    │
        │          │
        └──────────┘
```
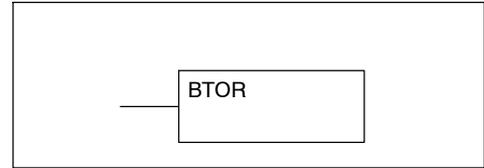
**NOTE:** This instruction will not work for a signed decimal.

| 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 value in the accumulator 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.



*DirectSOFT*

X1 — LDD V1400
Load the value in V1400 and V1401 into the accumulator

BTOR
Convert the binary value in the accumulator to the real number equivalent format

OUTD V1500
Copy the real value in the accumulator to V1500 and V1501

2 (exp 18)
127 + 18 = 145
145 = 128 + 16 + 1

Sign Bit   Exponent (8 bits)   Mantissa (23 bits)

Real Number Format

| 4 | 8 | A | E | 4 | 8 | 2 | 0 |
|---|---|---|---|---|---|---|---|
| V1501 | | | | V1500 | | | |

The real number (HEX) value copied to V1500
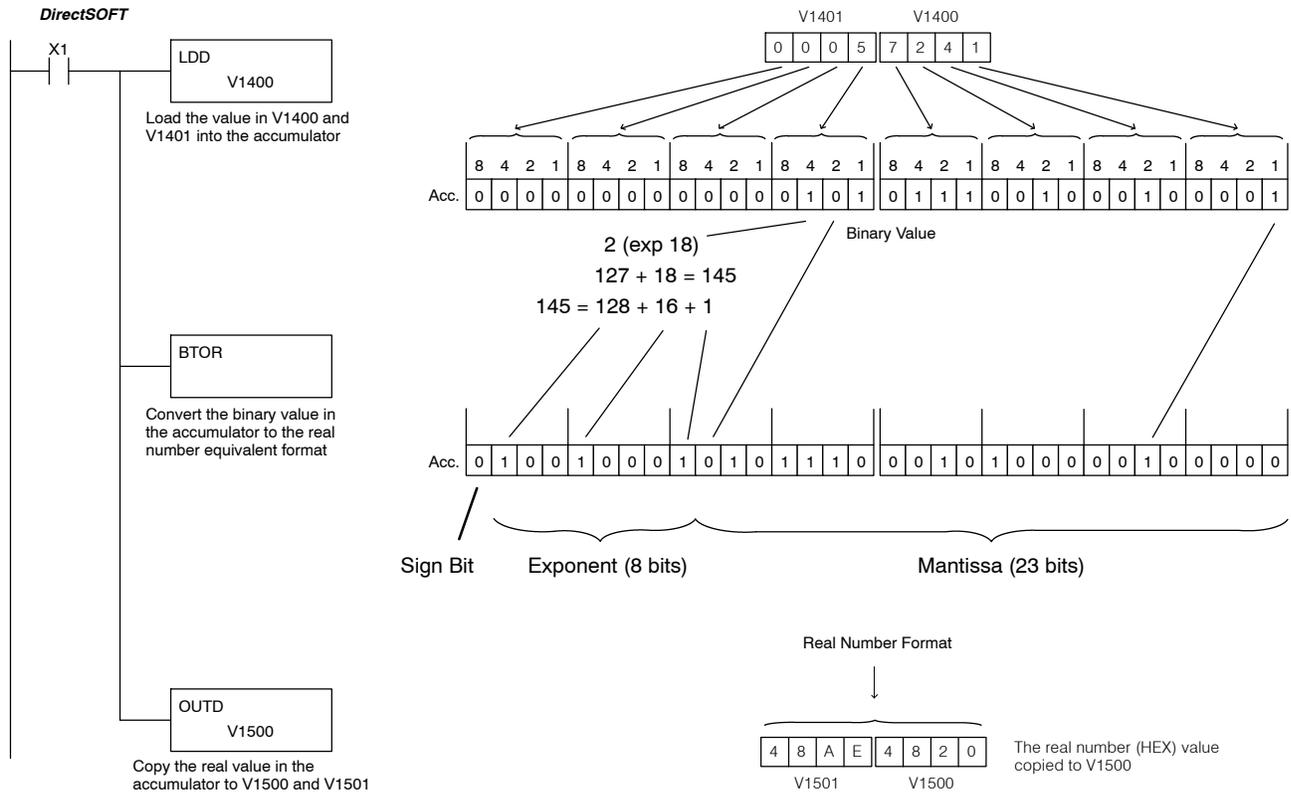
Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
| LD | SHFT | D | SHFT | V | 1 | 4 | 0 | 0 | ↵ |
| SHFT | B | T | O | R | SHFT | ↵ |
| OUT | SHFT | D | SHFT | V | 1 | 5 | 0 | 0 | ↵ |

## Real to Binary Conversion (RTOB)

| | | |
|:---:|:---:|:---:|
| × | × | ✓ |
| **430** | **440** | **450** |
| ✓ | | ✓ |
| **DS** | | **HPP** |

The Real-to-Binary instruction converts the real number in the accumulator to a binary value. If the real number is negative, it will become a signed decimal. The result resides in the accumulator. Both the binary and the real number may use all 32 bits of the accumulator. Any decimal portion will be truncated.

```
        ┌──────────────┐
   ──────┤ RTOB         │
        │              │
        └──────────────┘
```

| 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 a valid floating point number. |
| SP73 | on when a signed addition or subtraction results in a incorrect sign bit. |
| 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 V1400 and V1401 is loaded into the accumulator using the Load Double instruction. The RTOB instruction converts the real value in the accumulator to 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.
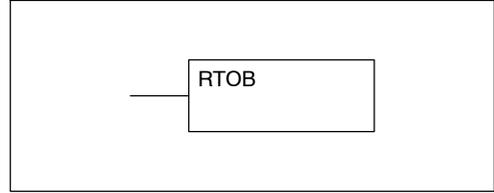


*DirectSOFT*

LDD V1400 — Load the value in V1400 and V1401 into the accumulator

RTOB — Convert the real number in the accumulator to binary format.

OUTD V1500 — Copy the real value in the accumulator to V1500 and V1501

Real Number Format: 4 8 A E 4 8 2 0 (V1401 V1400)

Sign Bit | Exponent (8 bits) | Mantissa (23 bits)

Acc. 0 1 0 0 1 0 0 0 1 0 1 0 1 1 1 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0

128 + 16 + 1 = 145
127 + 18 = 145

2 (exp 18)

Binary Value

Acc. 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 1 1 1 0 0 1 0 0 0 1 0 0 0 0 1

V1501 V1500

0 0 0 5 7 2 4 1 — The binary number copied to V1400.
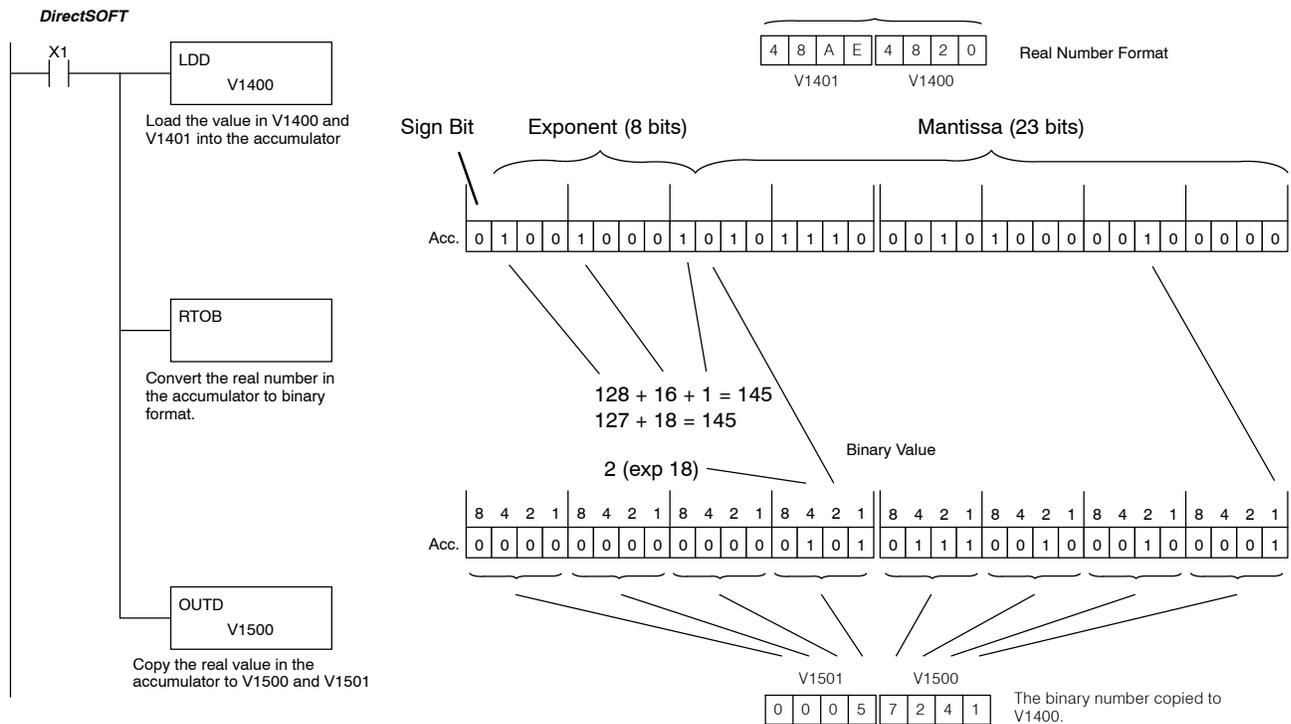
**Handheld Programmer Keystrokes**

| STR | X(IN) | 1 | ↵ |
| LD | SHFT | D | SHFT | V | 1 | 4 | 0 | 0 | ↵ |
| SHFT | R | T | O | B | SHFT | ↵ |
| OUT | SHFT | D | SHFT | V | 1 | 5 | 0 | 0 | ↵ |

Standard RLL Instructions

**Radian Real Conversion (RADR)**

| × | × | ✓ |
|---|---|---|
| 430 | 440 | 450 |

| ✓ | × |
|---|---|
| DS | HPP |

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.

```
    ┌──────────┐
────┤   RADR   │
    │          │
    └──────────┘
```

**Degree Real Conversion (DEGR)**

| × | × | ✓ |
|---|---|---|
| 430 | 440 | 450 |

| ✓ | × |
|---|---|
| DS | HPP |
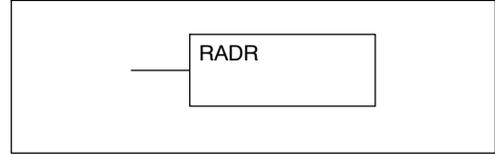
The Degree Real instruction converts the degree real radian value stored in the accumulator to the equivalent real number in degrees. The result resides in the accumulator.

```
    ┌──────────┐
────┤   DEGR   │
    │          │
    └──────────┘
```
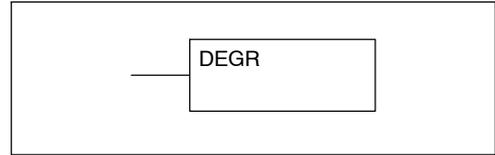
The two instructions described above convert real numbers into the accumulator from degree format to radian format, and visa-versa. In degree format, a circle contains 360 degrees. In radian format, a circle contains 2 Π radians. These convert between both positive and negative real numbers, and for angles greater than a full circle. These functions are very useful when combined with the transcendantal trigonometric functions (see the section on math instructions).

| 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 a valid floating point number. |
| SP74 | On anytime a floating point math operation results in an underflow error. |
| SP75 | On when a BCD instruction is executed and a NON–BCD number was encountered. |

**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 a 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.

Accumulator contents
(viewed as real number)

| | | | |
|---|---|---|---|
| *DirectSOFT* | | | |
| X1 | LDR<br>   R45 | Load the real number 45 into the accumulator. | 45.000000 |
| | RADR | Convert the degrees into radians, leaving the result in the accumulator. | 0.7358981 |
| | SINR | Take the sine of the number in the accumulator, which is in radians. | 0.7071067 |
| | OUTD<br>   V2000 | Copy the value in the accumulator to V2000 and V2001. | 0.7071067 |

Standard
RLL Instructions
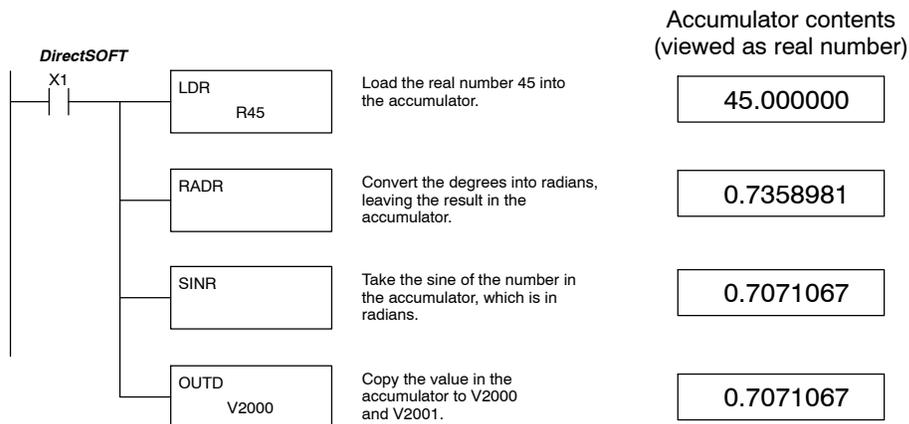
## ASCII to HEX (ATH)

| × | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

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.

```
        ┌──────────────┐
        │ ATH          │
────────│    Vaaa      │
        │              │
        └──────────────┘
```
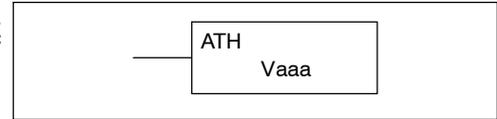
This means an ASCII table of four V–memory locations would only require two V–memory locations for the equivalent HEX table. The function parameters are loaded into the accumulator stack and the accumulator by two additional instructions. Listed below are the steps necessary to program an ASCII to HEX 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 | | DL440 Range | DL450 Range |
|---|---|---|---|
| | | aaa | aaa |
| V-memory | V | All (See p. 3–41) | All (See p. 3–42) |

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 |

Standard
RLL Instructions

**DirectSOFT**

```
X1        ┌─────────────┐
──┤ ├──┬──│ LD          │   Load the constant value
         │ │      K4     │   into the lower 16 bits of the
         │ └─────────────┘   accumulator. This value
         │                   defines the number of V
         │ ┌─────────────┐   memory location in the
         │ │ LDA         │   ASCII table
         ├─│     O 1400  │   Convert octal 1400 to HEX
         │ └─────────────┘   300 and load the value into
         │                   the accumulator
         │ ┌─────────────┐
         │ │ ATH         │   V1600 is the starting
         └─│      V1600  │   location for the HEX table
           └─────────────┘
```

ASCII TABLE

Hexadecimal Equivalents

V1400 | 33 34

V1401 | 31 32

1234 | V1600

V1402 | 37 38

5678 | V1601

V1403 | 35 36

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
| LD | K(CON) | 4 | ↵ |
| LD | SHFT | A | OCT | 1 | 4 | 0 | 0 | ↵ |
| SHFT | A | T | H | SHFT | V | 1 | 6 | 0 | 0 | ↵ |

## HEX to ASCII (HTA)

| ✕ | ✓ | ✓ |
| --- | --- | --- |
| **430** | **440** | **450** |

| ✓ | | ✓ |
| --- | --- | --- |
| **DS** | | **HPP** |

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.

```
┌─────────────┐
│ HTA         │
──│     Vaaa   │
│             │
└─────────────┘
```
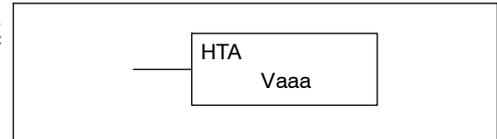
This means a HEX table of two V-memory locations would require four V-memory locations for the equivalent ASCII table. The function parameters are loaded into the accumulator stack and the accumulator by two additional instructions. Listed below are the steps necessary to program a HEX to ASCII table 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.
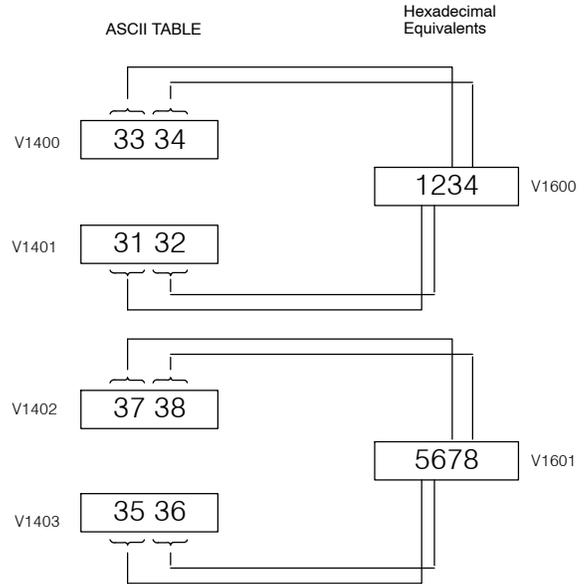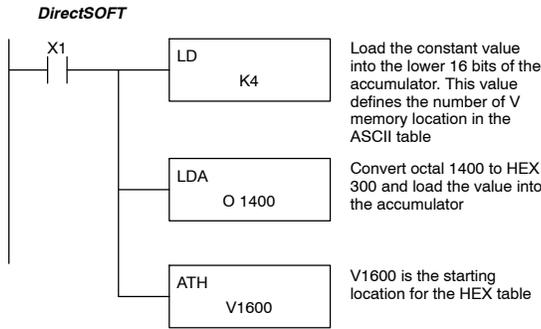
Step 3: — Specify the starting V-memory location (Vaaa) for the ASCII table in the HTA instruction.

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.
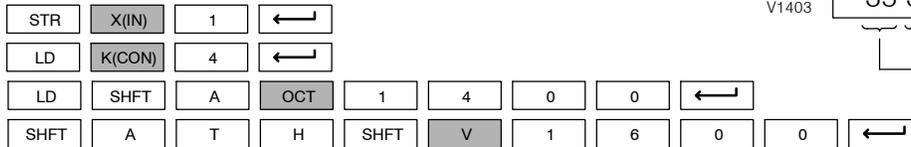
| Operand Data Type | | DL440 Range | DL440 Range |
| --- | --- | --- | --- |
| | | aaa | aaa |
| V-memory | V | All (See p. 3–41) | All (See p. 3–42) |

Standard RLL Instructions

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.

**DirectSOFT**

X1 ——| |——

LD
    K2

Load the constant value into the lower 16 bits of the accumulator. This value defines the number of V locations in the HEX table.

LDA
    O 1500

Convert octal 1500 to HEX 340 and load the value into the accumulator

HTA
    V1400

V1400 is the starting location for the ASCII table. The conversion is executed by this instruction.

Hexadecimal Equivalents

ASCII TABLE

V1500   1234

V1501   5678

33 34   V1400

31 32   V1401

37 38   V1402

35 36   V1403

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
| LD | K(CON) | 2 | ↵ |
| LD | SHFT | A | OCT | 1 | 5 | 0 | 0 | ↵ |
| SHFT | H | T | A | SHFT | V | 1 | 4 | 0 | 0 | ↵ |

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 |

Standard
RLL Instructions

**Segment (SEG)**

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

The BCD/Segment instruction converts a four digit HEX value in the accumulator to seven segment display format. The result resides in the accumulator.
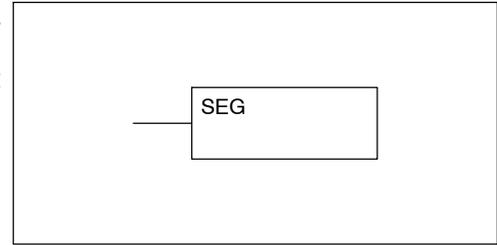
```
          SEG
```

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 binary (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.

*DirectSOFT*

```
X1
--| |--        LD
                    V1400
```
Load the value in V1400 into the lower 16 bits of the accumulator

```
               SEG
```
Convert the binary (HEX) value in the accumulator to seven segment display format

```
               OUTF      Y20
                    K32
```
Copy the value in the accumulator to Y20–Y57

V1400

| 6 | F | 7 | 1 |
|---|---|---|---|

Acc.

| 31 30 29 28 | 27 26 25 24 | 23 22 21 20 | 19 18 17 16 | 15 14 13 12 | 11 10 9 8 | 7 6 5 4 | 3 2 1 0 |
|---|---|---|---|---|---|---|---|
| 0 0 0 0 | 0 0 0 0 | 0 0 0 0 | 0 0 0 0 | 0 1 1 0 | 1 1 1 1 | 0 1 1 1 | 0 0 0 1 |

Acc.

| 31 30 29 28 | 27 26 25 24 | 23 22 21 20 | 19 18 17 16 | 15 14 13 12 | 11 10 9 8 | 7 6 5 4 | 3 2 1 0 |
|---|---|---|---|---|---|---|---|
| 0 1 1 1 | 1 1 0 1 | 0 1 1 1 | 0 0 0 1 | 0 0 0 0 | 0 1 1 1 | 0 0 0 0 | 0 1 1 0 |
| – g f e | d c b a | – g f e | d c b a | – g f e | d c b a | – g f e | d c b a |

Segment Labels

Segment Labels

```
      a
   f     b
      g
   e     c
      d
```

| Y57 | Y56 | Y55 | Y54 | Y53 | . . . . | . . . . | Y24 | Y23 | Y22 | Y21 | Y20 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| OFF | ON | ON | ON | ON | | | OFF | OFF | ON | ON | OFF |

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
|---|---|---|---|

| LD | SHFT | V | 1 | 4 | 0 | 0 | ↵ |
|---|---|---|---|---|---|---|---|

| SHFT | S | E | G | ↵ |
|---|---|---|---|---|

| OUT | SHFT | F | SHFT | Y(OUT) | 2 | 0 | K(CON) | 3 | 2 | ↵ |
|---|---|---|---|---|---|---|---|---|---|---|

Standard RLL Instructions

## Gray Code (GRAY)

| | | |
|---|---|---|
| ✗ | ✓ | ✓ |
| **430** | **440** | **450** |

| | |
|---|---|
| ✓ | ✓ |
| **DS** | **HPP** |

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 grey 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.

```
            ┌────────────┐
────────────┤  GRAY      │
            │            │
            └────────────┘
```
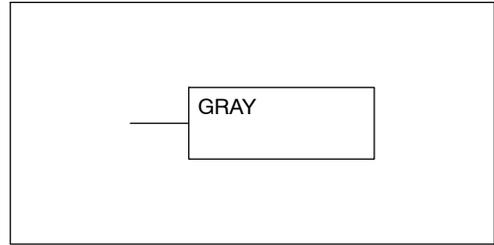
In the following example, when X1 is ON the binary value represented by X10–X27 is loaded into the accumulator using the Load Formatted instruction. The gray code value in the accumulator is converted to BCD using the Gray Code instruction. The value in the lower 16 bits of the accumulator is copied to V1500.

*DirectSOFT*

```
 X1
──┤ ├──┐   ┌──────────────┐
        │   │ LDF      X10 │
        │   │          K16 │
        │   └──────────────┘
```
Load the value represented by X10–X27 into the lower 16 bits of the accumulator

```
        │   ┌──────────────┐
        │   │ GRAY         │
        │   │              │
        │   └──────────────┘
```
Convert the 16 bit grey code value in the accumulator to a BCD value

```
        │   ┌──────────────┐
        │   │ OUT          │
        │   │     V1500    │
        │   └──────────────┘
```
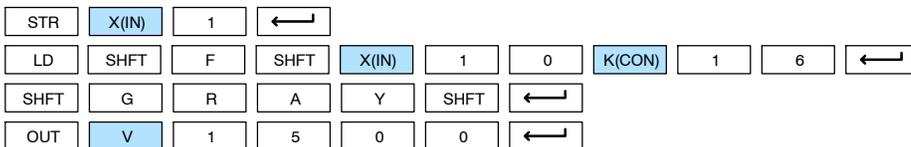Copy the value in the lower 16 bits of the accumulator to V1500

| X27 | X26 | X25 | . . . . | X12 | X11 | X10 |
|-----|-----|-----|---------|-----|-----|-----|
| OFF | OFF | OFF |         | ON  | OFF | ON  |

| | 31 30 29 28 | 27 26 25 24 | 23 22 21 20 | 19 18 17 16 | 15 14 13 12 | 11 10 9 8 | 7 6 5 4 | 3 2 1 0 |
|---|---|---|---|---|---|---|---|---|
| Acc. | 0 0 0 0 | 0 0 0 0 | 0 0 0 0 | 0 0 0 0 | 0 0 0 0 | 0 0 0 0 | 0 0 0 0 | 0 1 0 1 |

| | 31 30 29 28 | 27 26 25 24 | 23 22 21 20 | 19 18 17 16 | 15 14 13 12 | 11 10 9 8 | 7 6 5 4 | 3 2 1 0 |
|---|---|---|---|---|---|---|---|---|
| Acc. | 0 0 0 0 | 0 0 0 0 | 0 0 0 0 | 0 0 0 0 | 0 0 0 0 | 0 0 0 0 | 0 0 0 0 | 0 1 1 0 |

| 0 | 0 | 0 | 6 |
|---|---|---|---|

V1500

### Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ⏎ |
|-----|-------|---|---|

| LD | SHFT | F | SHFT | X(IN) | 1 | 0 | K(CON) | 1 | 6 | ⏎ |
|----|------|---|------|-------|---|---|--------|---|---|---|

| SHFT | G | R | A | Y | SHFT | ⏎ |
|------|---|---|---|---|------|---|

| OUT | V | 1 | 5 | 0 | 0 | ⏎ |
|-----|---|---|---|---|---|---|

| Gray Code | BCD |
|-----------|-----|
| 0000000000 | 0000 |
| 0000000001 | 0001 |
| 0000000011 | 0002 |
| 0000000010 | 0003 |
| 0000000110 | 0004 |
| 0000000111 | 0005 |
| 0000000101 | 0006 |
| 0000000100 | 0007 |
| • | • |
| • | • |
| • | • |
| 1000000001 | 1022 |
| 1000000000 | 1023 |

**Shuffle Digits
(SFLDGT)**

| × | ✓ | ✓ |
|---|---|---|
| 430 | 440 | 450 |

| ✓ | ✓ |
|---|---|
| DS | HPP |

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.

```
           ┌──────────┐
───────────│ SFLDGT   │
           │          │
           └──────────┘
```
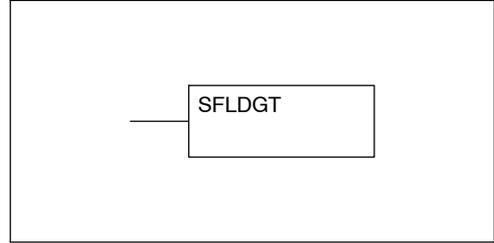
Step 1:— Load the value (digits) to be shuffled into the first level of the accumulator stack.

Step 2:— Load the order the digits will be shuffled to into the accumulator, numbered 1 through 8 ("1" being the least significant digit, and "8" being the most significant digit.

Note:— If the number used to specify the order contains a 0 or 9–F, the corresponding position will be set to 0.
See example

Note:—If the number used to specify the order contains duplicate numbers, the most significant duplicate number is valid. The result resides in the accumulator.
See example

Step 3:— Insert the SFLDGT instruction.

**Shuffle Digits
Block Diagram**

There are a maximum of 8 digits that can be shuffled. The bit positions in the first level of the accumulator stack defines 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.

Digits to be
shuffled (first stack location)

| 9 | A | B | C | D | E | F | 0 |
|---|---|---|---|---|---|---|---|

↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓

| 1 | 2 | 8 | 7 | 3 | 6 | 5 | 4 |
|---|---|---|---|---|---|---|---|

Specified order (accumulator)

Bit Positions    8   7   6   5   4   3   2   1

| B | C | E | F | 0 | D | A | 9 |
|---|---|---|---|---|---|---|---|

Result (accumulator)
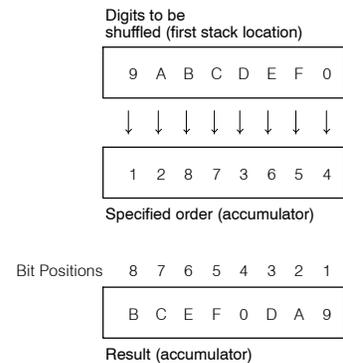
In the following example when X1 is on, The value in the first level of the accumulator stack will be reorganized in the order specified by the value in the accumulator.

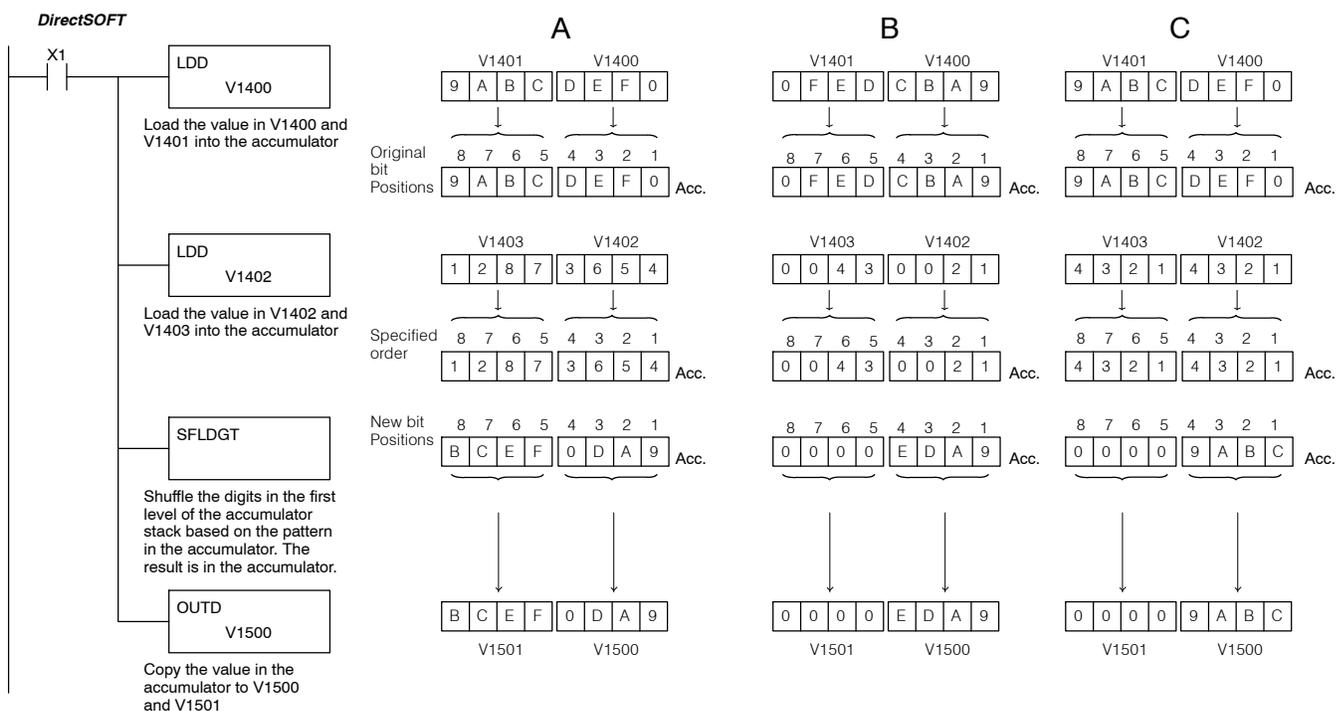Example A shows how the shuffle digits works when 0 or 9 –F is not used when specifying the order the digits are to be shuffled. Also, there are no duplicate numbers in the specified order.

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.



*DirectSOFT*

X1

LDD
V1400
Load the value in V1400 and V1401 into the accumulator

LDD
V1402
Load the value in V1402 and V1403 into the accumulator

SFLDGT
Shuffle the digits in the first level of the accumulator stack based on the pattern in the accumulator. The result is in the accumulator.

OUTD
V1500
Copy the value in the accumulator to V1500 and V1501

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ | | | | | |
|-----|-------|---|---|---|---|---|---|---|
| LD | SHFT | D | SHFT | V | 1 | 4 | 0 | 0 | ↵ |
| LD | SHFT | D | SHFT | V | 1 | 4 | 0 | 2 | ↵ |
| SHFT | S | F | L | D | G | T | ↵ | |
| OUT | SHFT | D | SHFT | V | 1 | 5 | 0 | 0 | ↵ |

# Table Instructions

**Fill
(FILL)**

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

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.

```
        ┌──────────┐
────────┤ FILL     │
        │     A aaa│
        └──────────┘
```
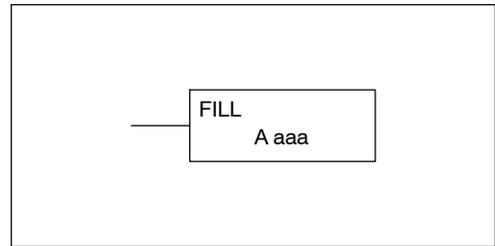
Step 1:— Load the number of V-memory locations to be filled into the first level of the accumulator stack. This parameter must be a HEX value, 0-FF.

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 instructions 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 | | DL430 Range | DL440 Range | DL450 Range |
|---|---|---|---|---|
| | **A** | **aaa** | **aaa** | **aaa** |
| V-memory | V | All (See p. 3-40) | All (See p. 3-41) | All (See p. 3-42) |
| Pointer | P | All (See p. 3-40) | All (See p. 3-41) | All (See p. 3-42) |
| Constant | K | 0-FF | 0-FF | 0-FF |

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.



*DirectSOFT*



Handheld Programmer Keystrokes

**Find
(FIND)**

| × | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | | ✓ |
|---|---|---|
| **DS** | | **HPP** |

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.

```
            ┌──────────┐
   ─────────┤ FIND     │
            │   A aaa  │
            └──────────┘
```

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.
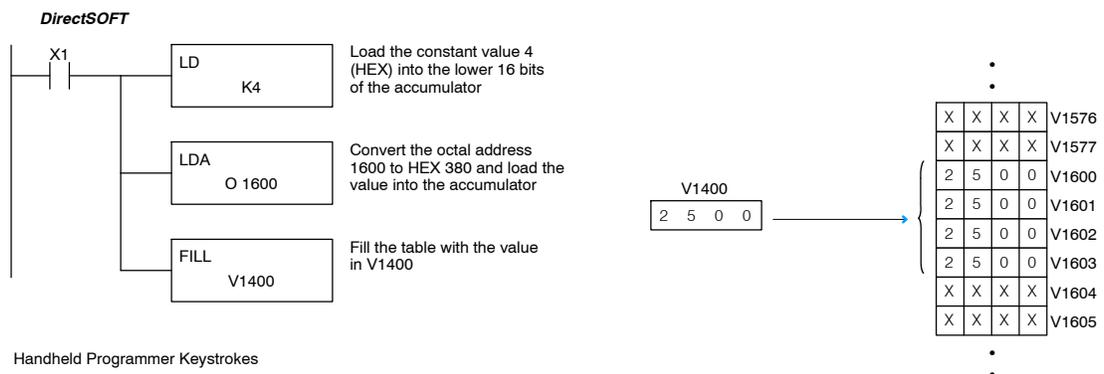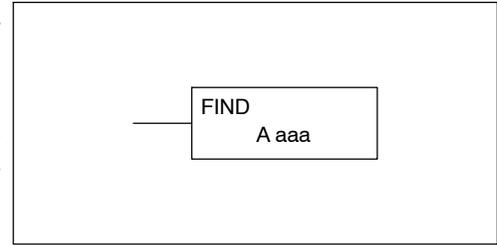
Step 3:— Load the offset from the starting location to begin the search. This parameter must be a HEX value.

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 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. The result will be a HEX value.

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 | | DL440 Range | DL450 Range |
|---|---|---|---|
| | **A** | **aaa** | **aaa** |
| V-memory | V | All (See p. 3-41) | All (See p. 3-42) |
| Constant | K | 0-FFFF | 0-FFFF |

| 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.

Standard
RLL Instructions

*DirectSOFT*



```
    X1        LD
──┤ ├──┐      K6
       │
```

Load the constant value 6 (HEX) into the lower 16 bits of the accumulator

```
          LDA
          O 1400
```

Convert octal 1400 to HEX 300 and load the value into the accumulator.

```
          LD
          K2
```

Load the constant value 2 into the lower 16 bits of the accumulator

```
          FIND
          K8989
```

Find the location in the table where the value 8989 resides

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | V1400 | 0 | Table length |
| 0 | 5 | 0 | 0 | V1401 | 1 | |
| 9 | 9 | 9 | 9 | V1402 | 2 | |
| 3 | 0 | 7 | 4 | V1403 | 3 | |
| 8 | 9 | 8 | 9 | V1404 | 4 | |
| 1 | 0 | 1 | 0 | V1405 | 5 | |
| X | X | X | X | V1406 | | |
| X | X | X | X | V1407 | | |

Offset

Begin here ──→

Accumulator

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4 |
|---|---|---|---|---|---|---|---|

V1404 contains the location where the match was found. The value 8989 was the 4th location after the start of the specified table.

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
|-----|-------|---|---|
| LD | K(CON) | 6 | ↵ |
| LD | SHFT | A | OCT | 1 | 4 | 0 | 0 | ↵ |
| LD | K(CON) | 2 | ↵ |
| SHFT | F | I | N | D |
| SHFT | K(CON) | 8 | 9 | 8 | 9 | ↵ |

**Find Greater Than (FDGT)**

| × | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | | ✓ |
|---|---|---|
| **DS** | | **HPP** |

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.

```
          FDGT
──────────  A aaa
```
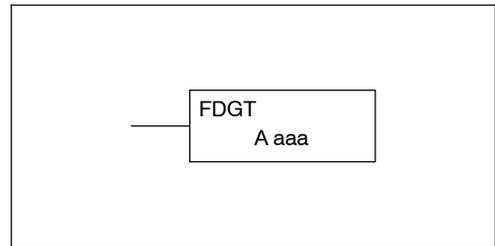
**NOTE:** This instruction does not have an offset, such as the one required for the FIND instruction.

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 instructions 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 is returned to the accumulator. SP53 will be set on if the value is not found and 0 will be returned in the accumulator. The result will be a HEX value.

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.
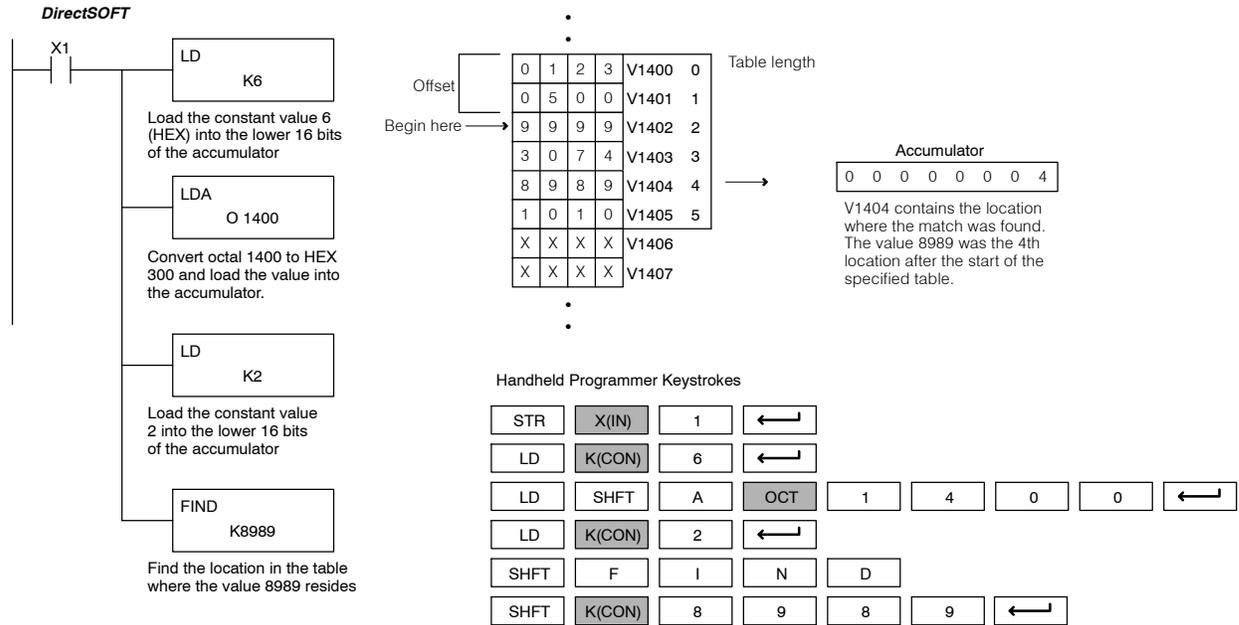
| Operand Data Type | | DL440 Range | DL450 Range |
|---|---|---|---|
| | A | aaa | aaa |
| V-memory | V | All (See p. 3–41) | All (See p. 3–42) |
| Constant | K | 0–FFFF | 0–FFFF |

| Discrete Bit Flags | Description |
|---|---|
| SP53 | on if there is no value in the table that is greater than the search value. |

**NOTE:** Status flags are only valid until another instruction that uses the same flags is executed.
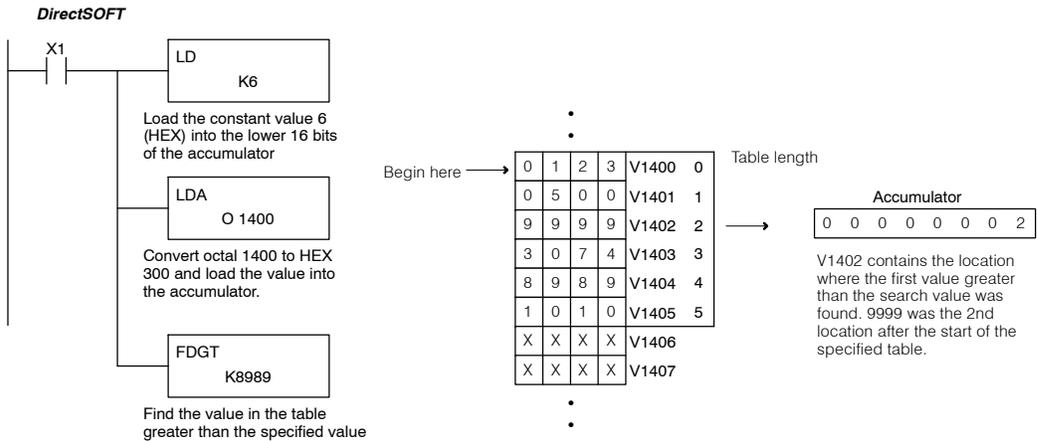The pointer for this instruction starts at 0 and resides in the accumulator.

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.

*DirectSOFT*

X1
LD
K6

Load the constant value 6 (HEX) into the lower 16 bits of the accumulator

LDA
O 1400

Convert octal 1400 to HEX 300 and load the value into the accumulator.

FDGT
K8989

Find the value in the table greater than the specified value

Begin here →

| 0 | 1 | 2 | 3 | V1400 | 0 |
| 0 | 5 | 0 | 0 | V1401 | 1 |
| 9 | 9 | 9 | 9 | V1402 | 2 |
| 3 | 0 | 7 | 4 | V1403 | 3 |
| 8 | 9 | 8 | 9 | V1404 | 4 |
| 1 | 0 | 1 | 0 | V1405 | 5 |
| X | X | X | X | V1406 | |
| X | X | X | X | V1407 | |

Table length

Accumulator

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 |

V1402 contains the location where the first value greater than the search value was found. 9999 was the 2nd location after the start of the specified table.

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |

| LD | K(CON) | 6 | ↵ |

| LD | SHFT | A | OCT | 1 | 4 | 0 | 0 | ↵ |

| SHFT | F | D | G | T | SHFT | K(CON) | 8 | 9 | 8 | 9 | ↵ |

**Move
(MOV)**

| | | |
|---|---|---|
| ✕ | ✓ | ✓ |
| **430** | **440** | **450** |

| | |
|---|---|
| ✓ | ✓ |
| **DS** | **HPP** |

The Move instruction moves up to 4095 values from a V-memory table to another V-memory table the same length. 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 Move function.

```
         ┌──────────────┐
─────────┤ MOV          │
         │      A aaa   │
         └──────────────┘
```
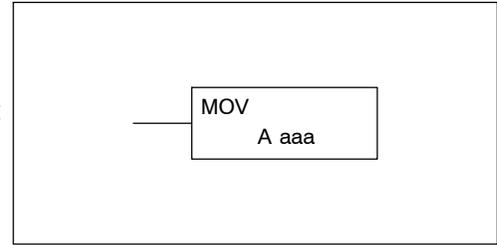
Step 1:— Load the number of V-memory locations to be moved into the first level of the accumulator stack. This parameter must be a HEX value, 0–FFF.
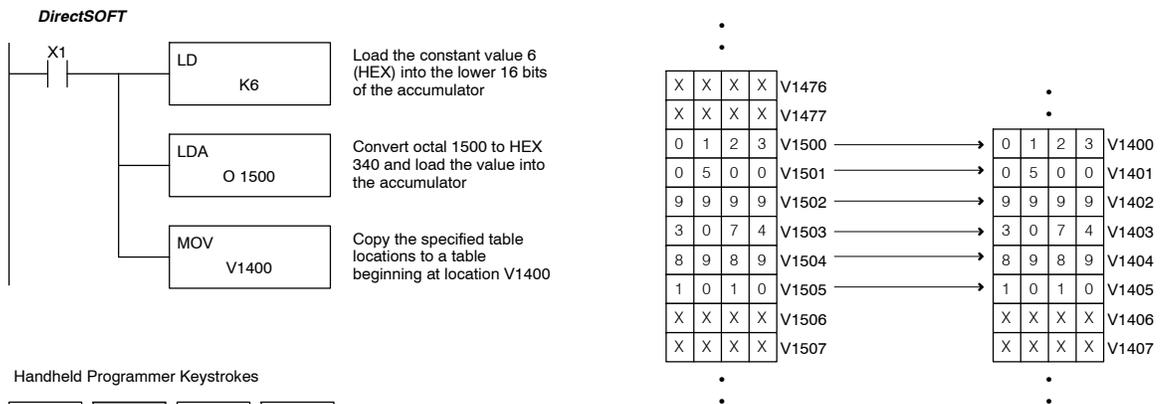
Step 2:— Load the starting V-memory location for the locations to be moved into the accumulator. This parameter must be a HEX value.

Step 3:— Insert the MOVE 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 | | DL440 Range | DL450 Range |
|---|---|---|---|
| | **A** | **aaa** | **aaa** |
| V-memory | V | All (See p. 3–41) | All (See p. 3–42) |
| Pointer | P | All (See p. 3–41) | All (See p. 3–42) |

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 1500 (V1500), the starting location for the source table is loaded into the accumulator. The destination table location (V1400) is specified in the Move instruction.

*DirectSOFT*

```
 X1        ┌──────────────┐
──┤ ├──────┤ LD           │   Load the constant value 6
           │      K6      │   (HEX) into the lower 16 bits
           └──────────────┘   of the accumulator

           ┌──────────────┐
           │ LDA          │   Convert octal 1500 to HEX
           │    O 1500    │   340 and load the value into
           └──────────────┘   the accumulator

           ┌──────────────┐
           │ MOV          │   Copy the specified table
           │    V1400     │   locations to a table
           └──────────────┘   beginning at location V1400
```

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| X | X | X | X | V1476 | | | | | | |
| X | X | X | X | V1477 | | | | | | |
| 0 | 1 | 2 | 3 | V1500 | → | 0 | 1 | 2 | 3 | V1400 |
| 0 | 5 | 0 | 0 | V1501 | → | 0 | 5 | 0 | 0 | V1401 |
| 9 | 9 | 9 | 9 | V1502 | → | 9 | 9 | 9 | 9 | V1402 |
| 3 | 0 | 7 | 4 | V1503 | → | 3 | 0 | 7 | 4 | V1403 |
| 8 | 9 | 8 | 9 | V1504 | → | 8 | 9 | 8 | 9 | V1404 |
| 1 | 0 | 1 | 0 | V1505 | → | 1 | 0 | 1 | 0 | V1405 |
| X | X | X | X | V1506 | | X | X | X | X | V1406 |
| X | X | X | X | V1507 | | X | X | X | X | V1407 |

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
|---|---|---|---|

| LD | K(CON) | 6 | ↵ |
|---|---|---|---|

| LD | SHFT | A | OCT | 1 | 5 | 0 | 0 | ↵ |
|---|---|---|---|---|---|---|---|---|

| SHFT | M | O | V | SHFT | V | 1 | 4 | 0 | 0 | ↵ |
|---|---|---|---|---|---|---|---|---|---|---|

**Table to Destination (TTD)**

| × | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

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.

```
        ┌──────────┐
────────┤ TTD      │
        │   Aaaa   │
        └──────────┘
```
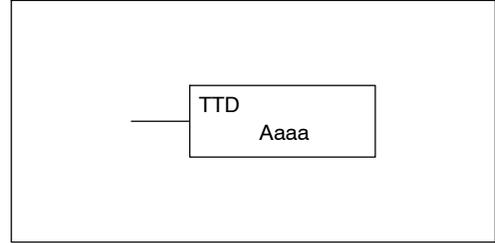
Step 1:— Load the length of the data table (number of V-memory locations) into the first level of the accumulator stack. This parameter must be a HEX value, 0 to FF.

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 | | DL440 Range | DL450 Range |
|---|---|---|---|
| | **A** | **aaa** | **aaa** |
| V-memory | V | All (See p. 3-41) | All (See p. 3-42) |

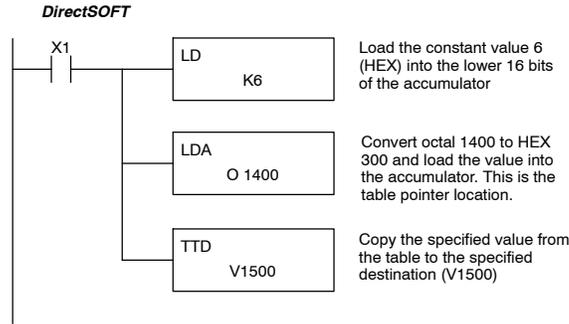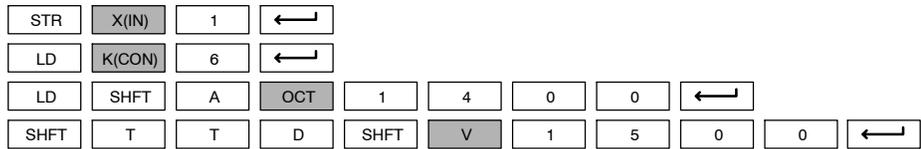| Discrete Bit Flags | Description |
|---|---|
| SP56 | ON when the table pointer equals the table length. |

**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*

| X1 | LD K6 | Load the constant value 6 (HEX) into the lower 16 bits of the accumulator |
| | LDA O 1400 | Convert octal 1400 to HEX 300 and load the value into the accumulator. This is the table pointer location. |
| | TTD V1500 | Copy the specified value from the table to the specified destination (V1500) |

Handheld Programmer Keystrokes

STR | X(IN) | 1 | ↵
LD | K(CON) | 6 | ↵
LD | SHFT | A | OCT | 1 | 4 | 0 | 0 | ↵
SHFT | T | T | D | SHFT | V | 1 | 5 | 0 | 0 | ↵

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.

Table

| | | | | | | |
| V1401 | 0 | 5 | 0 | 0 | | 0 6 |
| V1402 | 9 | 9 | 9 | 9 | | 1 |
| V1403 | 3 | 0 | 7 | 4 | | 2 |
| V1404 | 8 | 9 | 8 | 9 | | 3 |
| V1405 | 1 | 0 | 1 | 0 | | 4 |
| V1406 | 2 | 0 | 4 | 6 | | 5 |
| V1407 | X | X | X | X | | |

Table Pointer: 0 0 0 0 V1400

Destination: X X X X V1500

*DirectSOFT* (optional latch example using SP56)

| X1 | ( C0 PD ) |
| C1 | LD K6 — Load the constant value 6 (HEX) into the lower 16 bits of the accumulator |
| C0 | ( C1 SET ) |
| SP56 | ( C1 RST ) |
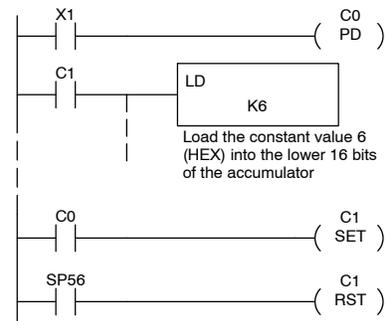
Since Special Relays are reset at the end of the scan, this latch *must* follow the TTD instruction in the program.

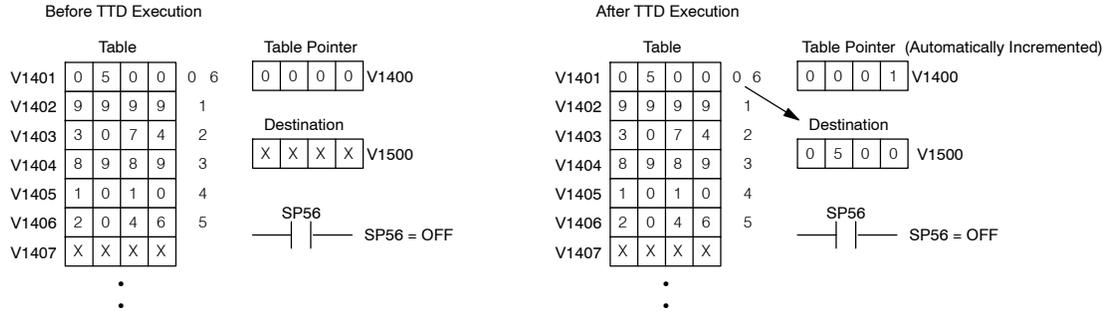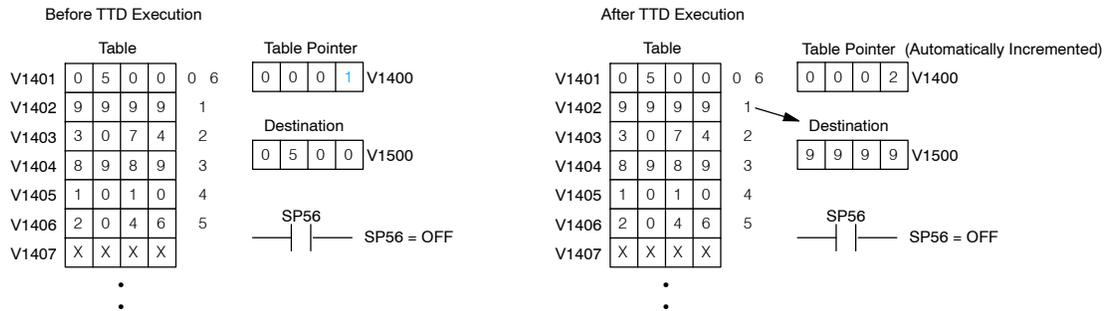The following diagram shows the scan-by-scan results of the execution for our example program. Notice how the pointer automatically cycles from 0 – 6, and then starts over at 1 instead of 0. Also, notice how SP56 is only on until the end of the scan.
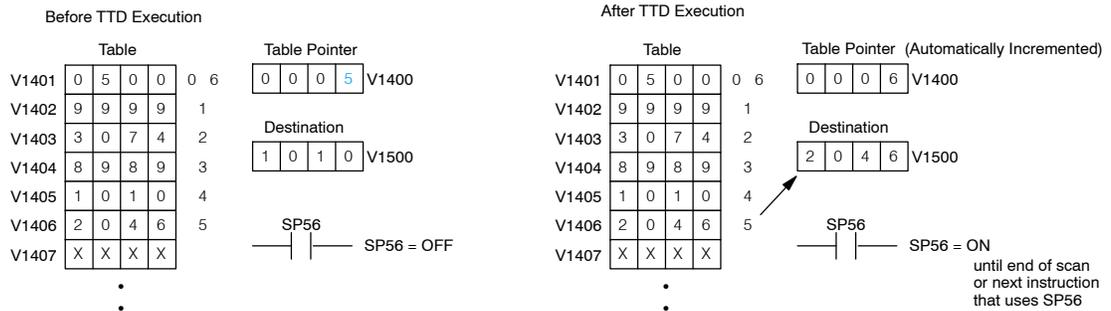
Example of Execution

### Scan N

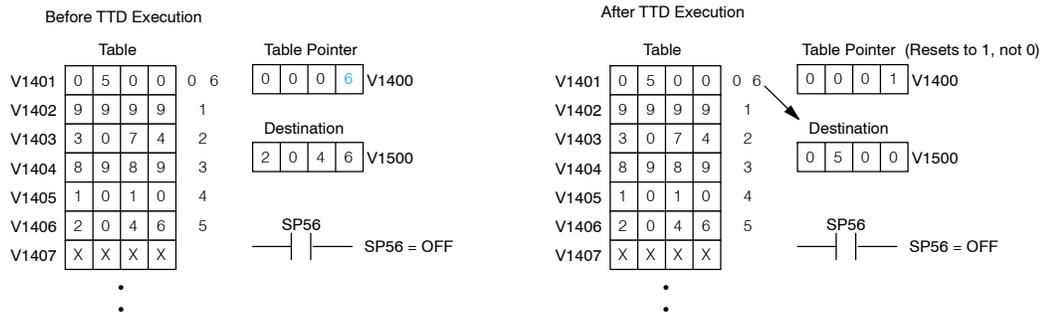**Before TTD Execution**

Table

| V1401 | 0 | 5 | 0 | 0 | 0 6 |
|---|---|---|---|---|---|
| V1402 | 9 | 9 | 9 | 9 | 1 |
| V1403 | 3 | 0 | 7 | 4 | 2 |
| V1404 | 8 | 9 | 8 | 9 | 3 |
| V1405 | 1 | 0 | 1 | 0 | 4 |
| V1406 | 2 | 0 | 4 | 6 | 5 |
| V1407 | X | X | X | X | |

Table Pointer

| 0 | 0 | 0 | 0 | V1400 |

Destination

| X | X | X | X | V1500 |

SP56 ─┤ ├─ SP56 = OFF

**After TTD Execution**

Table

| V1401 | 0 | 5 | 0 | 0 | 0 6 |
|---|---|---|---|---|---|
| V1402 | 9 | 9 | 9 | 9 | 1 |
| V1403 | 3 | 0 | 7 | 4 | 2 |
| V1404 | 8 | 9 | 8 | 9 | 3 |
| V1405 | 1 | 0 | 1 | 0 | 4 |
| V1406 | 2 | 0 | 4 | 6 | 5 |
| V1407 | X | X | X | X | |

Table Pointer (Automatically Incremented)

| 0 | 0 | 0 | 1 | V1400 |

Destination

| 0 | 5 | 0 | 0 | V1500 |

SP56 ─┤ ├─ SP56 = OFF

### Scan N+1

**Before TTD Execution**

Table

| V1401 | 0 | 5 | 0 | 0 | 0 6 |
|---|---|---|---|---|---|
| V1402 | 9 | 9 | 9 | 9 | 1 |
| V1403 | 3 | 0 | 7 | 4 | 2 |
| V1404 | 8 | 9 | 8 | 9 | 3 |
| V1405 | 1 | 0 | 1 | 0 | 4 |
| V1406 | 2 | 0 | 4 | 6 | 5 |
| V1407 | X | X | X | X | |

Table Pointer

| 0 | 0 | 0 | 1 | V1400 |

Destination

| 0 | 5 | 0 | 0 | V1500 |

SP56 ─┤ ├─ SP56 = OFF

**After TTD Execution**

Table

| V1401 | 0 | 5 | 0 | 0 | 0 6 |
|---|---|---|---|---|---|
| V1402 | 9 | 9 | 9 | 9 | 1 |
| V1403 | 3 | 0 | 7 | 4 | 2 |
| V1404 | 8 | 9 | 8 | 9 | 3 |
| V1405 | 1 | 0 | 1 | 0 | 4 |
| V1406 | 2 | 0 | 4 | 6 | 5 |
| V1407 | X | X | X | X | |

Table Pointer (Automatically Incremented)

| 0 | 0 | 0 | 2 | V1400 |

Destination

| 9 | 9 | 9 | 9 | V1500 |

SP56 ─┤ ├─ SP56 = OFF

### Scan N+5

**Before TTD Execution**

Table

| V1401 | 0 | 5 | 0 | 0 | 0 6 |
|---|---|---|---|---|---|
| V1402 | 9 | 9 | 9 | 9 | 1 |
| V1403 | 3 | 0 | 7 | 4 | 2 |
| V1404 | 8 | 9 | 8 | 9 | 3 |
| V1405 | 1 | 0 | 1 | 0 | 4 |
| V1406 | 2 | 0 | 4 | 6 | 5 |
| V1407 | X | X | X | X | |

Table Pointer

| 0 | 0 | 0 | 5 | V1400 |

Destination

| 1 | 0 | 1 | 0 | V1500 |

SP56 ─┤ ├─ SP56 = OFF

**After TTD Execution**

Table

| V1401 | 0 | 5 | 0 | 0 | 0 6 |
|---|---|---|---|---|---|
| V1402 | 9 | 9 | 9 | 9 | 1 |
| V1403 | 3 | 0 | 7 | 4 | 2 |
| V1404 | 8 | 9 | 8 | 9 | 3 |
| V1405 | 1 | 0 | 1 | 0 | 4 |
| V1406 | 2 | 0 | 4 | 6 | 5 |
| V1407 | X | X | X | X | |

Table Pointer (Automatically Incremented)

| 0 | 0 | 0 | 6 | V1400 |

Destination

| 2 | 0 | 4 | 6 | V1500 |

SP56 ─┤ ├─ SP56 = ON

until end of scan or next instruction that uses SP56

### Scan N+6

**Before TTD Execution**

Table

| V1401 | 0 | 5 | 0 | 0 | 0 6 |
|---|---|---|---|---|---|
| V1402 | 9 | 9 | 9 | 9 | 1 |
| V1403 | 3 | 0 | 7 | 4 | 2 |
| V1404 | 8 | 9 | 8 | 9 | 3 |
| V1405 | 1 | 0 | 1 | 0 | 4 |
| V1406 | 2 | 0 | 4 | 6 | 5 |
| V1407 | X | X | X | X | |

Table Pointer

| 0 | 0 | 0 | 6 | V1400 |

Destination

| 2 | 0 | 4 | 6 | V1500 |

SP56 ─┤ ├─ SP56 = OFF

**After TTD Execution**

Table

| V1401 | 0 | 5 | 0 | 0 | 0 6 |
|---|---|---|---|---|---|
| V1402 | 9 | 9 | 9 | 9 | 1 |
| V1403 | 3 | 0 | 7 | 4 | 2 |
| V1404 | 8 | 9 | 8 | 9 | 3 |
| V1405 | 1 | 0 | 1 | 0 | 4 |
| V1406 | 2 | 0 | 4 | 6 | 5 |
| V1407 | X | X | X | X | |

Table Pointer (Resets to 1, not 0)

| 0 | 0 | 0 | 1 | V1400 |

Destination

| 0 | 5 | 0 | 0 | V1500 |

SP56 ─┤ ├─ SP56 = OFF

Standard
RLL Instructions

**Remove from Bottom (RFB)**

| × | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | | ✓ |
|---|---|---|
| **DS** | | **HPP** |

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.
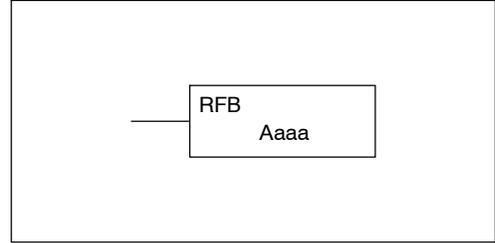
```
        ┌──────────┐
────────┤ RFB      │
        │   Aaaa   │
        └──────────┘
```

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 instructions 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.

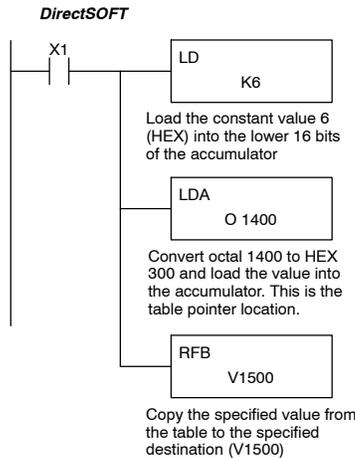| Operand Data Type | | DL440 Range | DL450 Range |
|---|---|---|---|
| | **A** | **aaa** | **aaa** |
| V-memory | **V** | All (See p. 3-41) | All (See p. 3-42) |

| Discrete Bit Flags | Description |
|---|---|
| SP56 | on when the table pointer equals 0 |

**NOTE:** Status flags (SPs) are only valid until:
— another instruction that uses the same flag is executed, or
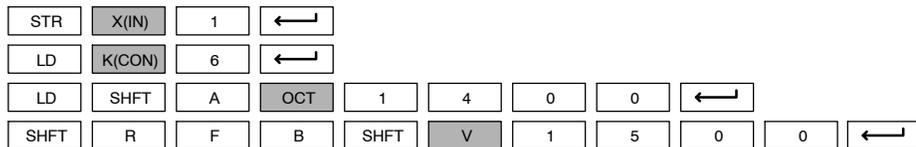— the end of the scan.
The pointer for this instruction can be set to start anywhere in the table. It is not set automatically. You have to load a value into the pointer somewhere in your program.

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.
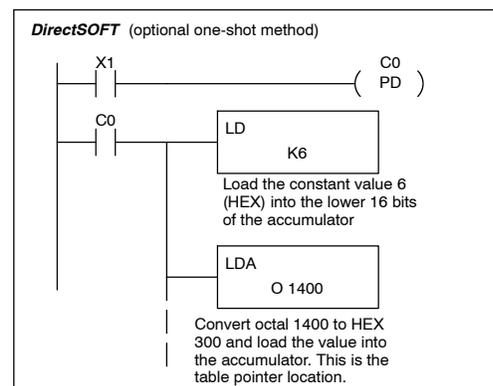
*DirectSOFT*

```
X1        ┌─────────────┐
──┤ ├──────┤ LD          │
          │        K6    │
          └─────────────┘
Load the constant value 6
(HEX) into the lower 16 bits
of the accumulator

          ┌─────────────┐
          │ LDA         │
          │     O 1400   │
          └─────────────┘
Convert octal 1400 to HEX
300 and load the value into
the accumulator. This is the
table pointer location.

          ┌─────────────┐
          │ RFB         │
          │     V1500    │
          └─────────────┘
Copy the specified value from
the table to the specified
destination (V1500)
```

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
|-----|-------|---|---|

| LD | K(CON) | 6 | ↵ |
|----|--------|---|---|

| LD | SHFT | A | OCT | 1 | 4 | 0 | 0 | ↵ |
|----|------|---|-----|---|---|---|---|---|

| SHFT | R | F | B | SHFT | V | 1 | 5 | 0 | 0 | ↵ |
|------|---|---|---|------|---|---|---|---|---|---|

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 | | | | | | | Table Pointer | | | | |
|-------|---|---|---|---|---|---|---|---|---|---|---|
| V1401 | 0 | 5 | 0 | 0 | 1 | | 0 | 0 | 0 | 0 | V1400 |
| V1402 | 9 | 9 | 9 | 9 | 2 | | | | | | |
| V1403 | 3 | 0 | 7 | 4 | 3 | | Destination | | | | |
| V1404 | 8 | 9 | 8 | 9 | 4 | | X | X | X | X | V1500 |
| V1405 | 1 | 0 | 1 | 0 | 5 | | | | | | |
| V1406 | 2 | 0 | 4 | 6 | 6 | | | | | | |
| V1407 | X | X | X | X | | | | | | | |

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)

```
X1                              ( C0  )
──┤ ├──────────────────────────(  PD  )

C0        ┌─────────────┐
──┤ ├──────┤ LD          │
          │        K6    │
          └─────────────┘
Load the constant value 6
(HEX) into the lower 16 bits
of the accumulator

          ┌─────────────┐
          │ LDA         │
          │     O 1400   │
          └─────────────┘
Convert octal 1400 to HEX
300 and load the value into
the accumulator. This is the
table pointer location.
```
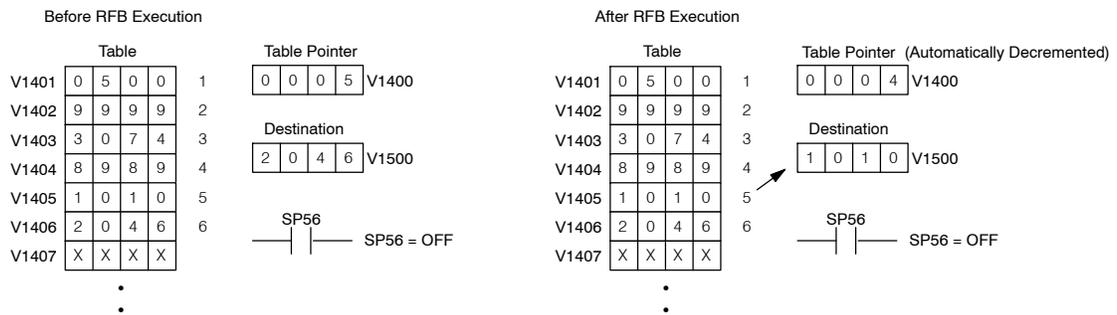
Standard
RLL Instructions

The following diagram shows the scan-by-scan results of the execution for our example program. Notice how the pointer automatically decrements from 6 – 0. Also, notice how SP56 is only on until the end of the scan.
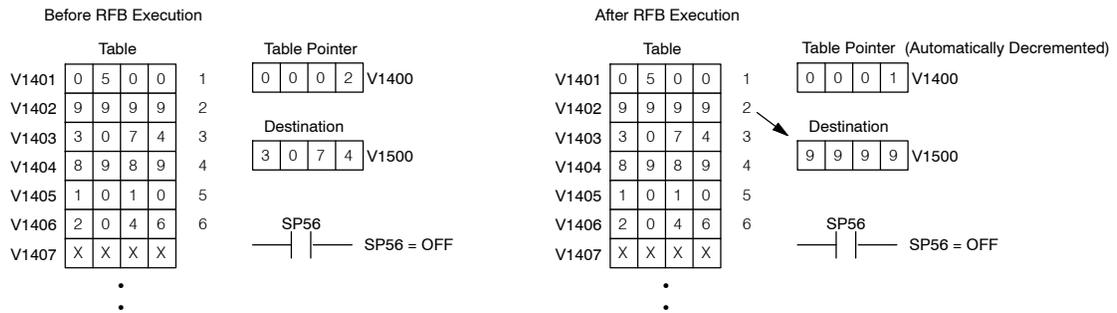
## Example of Execution

### Scan N

**Before RFB Execution**

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 6 V1400

Destination: X X X X V1500

SP56 — SP56 = OFF

**After RFB Execution**

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 (Automatically Decremented): 0 0 0 5 V1400

Destination: 2 0 4 6 V1500

SP56 — SP56 = OFF

### Scan N+1

**Before RFB Execution**

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 5 V1400

Destination: 2 0 4 6 V1500

SP56 — SP56 = OFF

**After RFB Execution**

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 (Automatically Decremented): 0 0 0 4 V1400

Destination: 1 0 1 0 V1500

SP56 — SP56 = OFF

### Scan N+4

**Before RFB Execution**

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 2 V1400

Destination: 3 0 7 4 V1500

SP56 — SP56 = OFF

**After RFB Execution**

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 (Automatically Decremented): 0 0 0 1 V1400

Destination: 9 9 9 9 V1500

SP56 — SP56 = OFF

### Scan N+5

**Before RFB Execution**

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 1 V1400

Destination: 9 9 9 9 V1500

SP56 — SP56 = OFF

**After RFB Execution**

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: 0 5 0 0 V1500

SP56 — SP56 = ON
until end of scan
or next instruction
that uses SP56

Standard
RLL Instructions

**Source to Table (STT)**

| × | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

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.
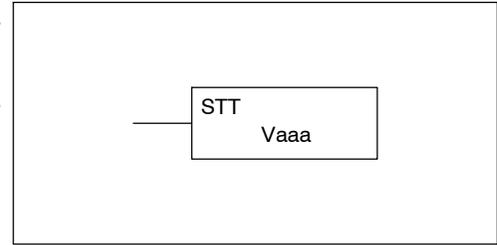
```
        ┌──────────┐
────────┤ STT      │
        │   Vaaa   │
        └──────────┘
```

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.

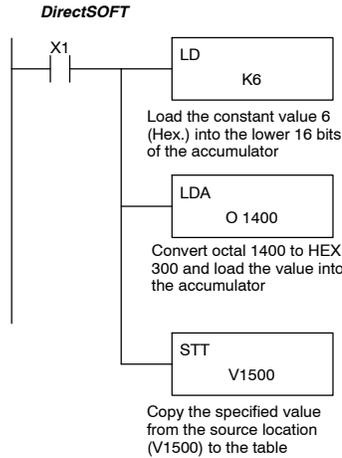| Operand Data Type | | DL440 Range | DL450 Range |
|---|---|---|---|
| | | **aaa** | **aaa** |
| V-memory | V | All (See p. 3-41) | All (See p. 3-42) |

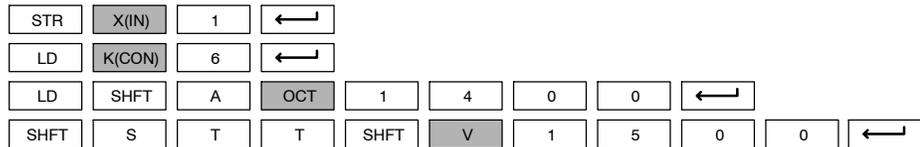| Discrete Bit Flags | Description |
|---|---|
| SP56 | on when the table pointer equals the table length |

**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.

**DirectSOFT**

X1 — LD
K6

Load the constant value 6 (Hex.) into the lower 16 bits of the accumulator

LDA
O 1400

Convert octal 1400 to HEX 300 and load the value into the accumulator

STT
V1500

Copy the specified value from the source location (V1500) to the table

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ← |
| LD | K(CON) | 6 | ← |
| LD | SHFT | A | OCT | 1 | 4 | 0 | 0 | ← |
| SHFT | S | T | T | SHFT | V | 1 | 5 | 0 | 0 | ← |

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.

| | Table | | | | | Table Pointer | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| V1401 | X | X | X | X | 0 6 | 0 0 0 0 | V1400 |
| V1402 | X | X | X | X | 1 | | |
| V1403 | X | X | X | X | 2 | **Data Source** | |
| V1404 | X | X | X | X | 3 | 0 5 0 0 | V1500 |
| V1405 | X | X | X | X | 4 | | |
| V1406 | X | X | X | X | 5 | | |
| V1407 | X | X | X | X | | | |

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.
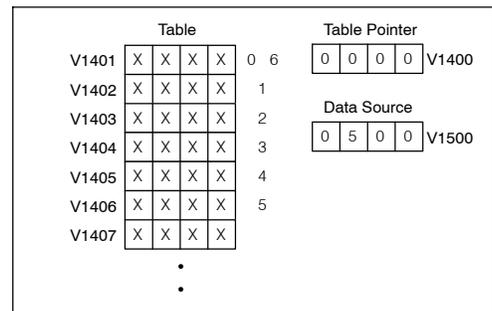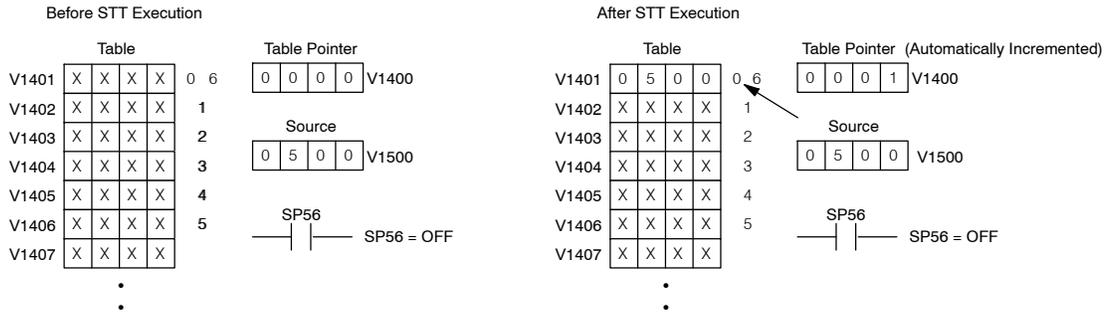
**DirectSOFT** (optional one-shot method)

X1 — ( C0 PD )

C0 — LD
K6

Load the constant value 6 (HEX) into the lower 16 bits of the accumulator

LDA
O 1400

Convert octal 1400 to HEX 300 and load the value into the accumulator. This is the starting table location.
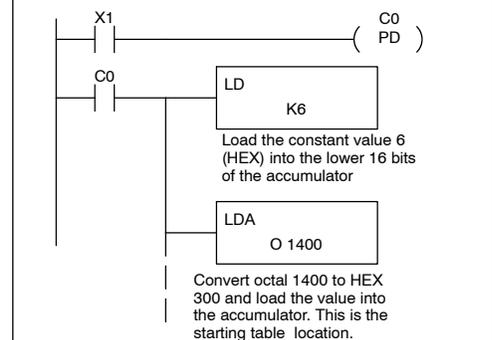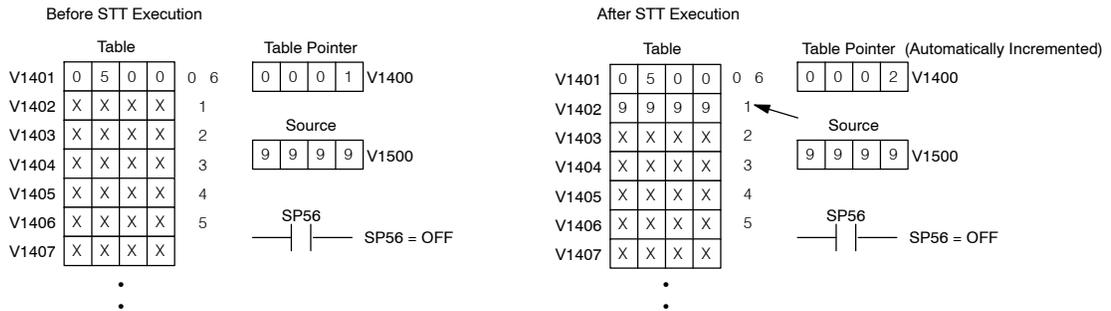
The following diagram shows the scan-by-scan results of the execution for our example program. Notice how the pointer automatically cycles from 0 – 6, and then starts over at 1 instead of 0. Also, notice how SP56 is affected by the execution. Although our example does not show it, we are assuming that 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.

### Example of Execution

**Scan N**

Before STT Execution

| Table | | | | | |
|---|---|---|---|---|---|
| V1401 | X | X | X | X | 0 6 |
| V1402 | X | X | X | X | 1 |
| V1403 | X | X | X | X | 2 |
| V1404 | X | X | X | X | 3 |
| V1405 | X | X | X | X | 4 |
| V1406 | X | X | X | X | 5 |
| V1407 | X | X | X | X | |

Table Pointer: 0 0 0 0 V1400

Source: 0 5 0 0 V1500

SP56 = OFF

After STT Execution

| Table | | | | | |
|---|---|---|---|---|---|
| V1401 | 0 | 5 | 0 | 0 | 0 6 |
| V1402 | X | X | X | X | 1 |
| V1403 | X | X | X | X | 2 |
| V1404 | X | X | X | X | 3 |
| V1405 | X | X | X | X | 4 |
| V1406 | X | X | X | X | 5 |
| V1407 | X | X | X | X | |

Table Pointer (Automatically Incremented): 0 0 0 1 V1400

Source: 0 5 0 0 V1500

SP56 = OFF

**Scan N+1**

Before STT Execution

| Table | | | | | |
|---|---|---|---|---|---|
| V1401 | 0 | 5 | 0 | 0 | 0 6 |
| V1402 | X | X | X | X | 1 |
| V1403 | X | X | X | X | 2 |
| V1404 | X | X | X | X | 3 |
| V1405 | X | X | X | X | 4 |
| V1406 | X | X | X | X | 5 |
| V1407 | X | X | X | X | |

Table Pointer: 0 0 0 1 V1400

Source: 9 9 9 9 V1500

SP56 = OFF

After STT Execution

| Table | | | | | |
|---|---|---|---|---|---|
| V1401 | 0 | 5 | 0 | 0 | 0 6 |
| V1402 | 9 | 9 | 9 | 9 | 1 |
| V1403 | X | X | X | X | 2 |
| V1404 | X | X | X | X | 3 |
| V1405 | X | X | X | X | 4 |
| V1406 | X | X | X | X | 5 |
| V1407 | X | X | X | X | |

Table Pointer (Automatically Incremented): 0 0 0 2 V1400

Source: 9 9 9 9 V1500

SP56 = OFF

**Scan N+5**

Before STT Execution

| Table | | | | | |
|---|---|---|---|---|---|
| V1401 | 0 | 5 | 0 | 0 | 0 6 |
| V1402 | 9 | 9 | 9 | 9 | 1 |
| V1403 | 3 | 0 | 7 | 4 | 2 |
| V1404 | 8 | 9 | 8 | 9 | 3 |
| V1405 | 1 | 0 | 1 | 0 | 4 |
| V1406 | X | X | X | X | 5 |
| V1407 | X | X | X | X | |

Table Pointer: 0 0 0 5 V1400

Source: 2 0 4 6 V1500

SP56 = OFF

After STT Execution

| Table | | | | | |
|---|---|---|---|---|---|
| V1401 | 0 | 5 | 0 | 0 | 0 6 |
| V1402 | 9 | 9 | 9 | 9 | 1 |
| V1403 | 3 | 0 | 7 | 4 | 2 |
| V1404 | 8 | 9 | 8 | 9 | 3 |
| V1405 | 1 | 0 | 1 | 0 | 4 |
| V1406 | 2 | 0 | 4 | 6 | 5 |
| V1407 | X | X | X | X | |

Table Pointer (Automatically Incremented): 0 0 0 6 V1400

Source: 2 0 4 6 V1500

SP56 = ON until end of scan or next instruction that uses SP56

**Scan N+6**

Before STT Execution

| Table | | | | | |
|---|---|---|---|---|---|
| V1401 | 0 | 5 | 0 | 0 | 0 6 |
| V1402 | 9 | 9 | 9 | 9 | 1 |
| V1403 | 3 | 0 | 7 | 4 | 2 |
| V1404 | 8 | 9 | 8 | 9 | 3 |
| V1405 | 1 | 0 | 1 | 0 | 4 |
| V1406 | 2 | 0 | 4 | 6 | 5 |
| V1407 | X | X | X | X | |

Table Pointer: 0 0 0 6 V1400

Source: 1 2 3 4 V1500

SP56 = OFF

After STT Execution

| Table | | | | | |
|---|---|---|---|---|---|
| V1401 | 1 | 2 | 3 | 4 | 0 6 |
| V1402 | 9 | 9 | 9 | 9 | 1 |
| V1403 | 3 | 0 | 7 | 4 | 2 |
| V1404 | 8 | 9 | 8 | 9 | 3 |
| V1405 | 1 | 0 | 1 | 0 | 4 |
| V1406 | 2 | 0 | 4 | 6 | 5 |
| V1407 | X | X | X | X | |

Table Pointer (Resets to 1, not 0): 0 0 0 1 V1400

Source: 1 2 3 4 V1500

SP56 = OFF

**Remove from Table (RFT)**

| × | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

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.
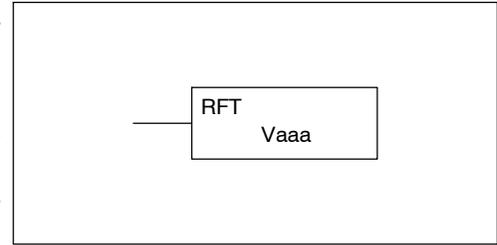
```
        ┌──────────┐
────────┤ RFT      │
        │    Vaaa  │
        └──────────┘
```

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 instruction which specifies destination V–memory location (Vaaa). The value will be moved to this location.

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 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 | | DL440 Range | DL450 Range |
|---|---|---|---|
| | | **aaa** | **aaa** |
| V–memory | V | All (See p. 3–41) | All (See p. 3–42) |

| Discrete Bit Flags | Description |
|---|---|
| SP56 | on when the table counter equals 0 |

**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 have to 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.

*DirectSOFT*

| X1 | LD<br>K6 | Load the constant value 6 (Hex.) into the lower 16 bits of the accumulator |
| --- | --- | --- |
| | LDA<br>O 1400 | Convert octal 1400 to HEX 300 and load the value into the accumulator |
| | RFT<br>V1500 | Copy the specified value from the table to the specified location (V1500) |

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
| --- | --- | --- | --- |
| LD | K(CON) | 6 | ↵ |

| LD | SHFT | A | OCT | 1 | 4 | 0 | 0 | ↵ |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |

| SHFT | R | F | T | SHFT | V | 1 | 5 | 0 | 0 | ↵ |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
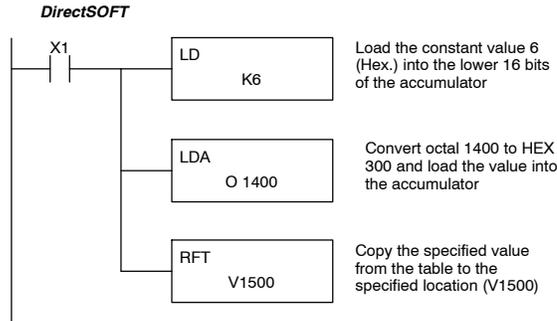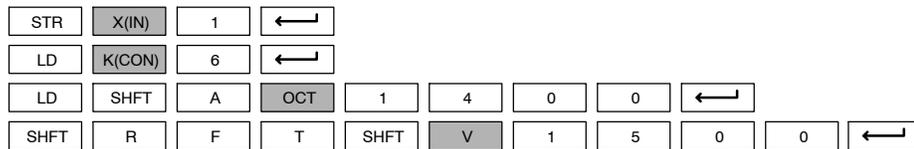
Since the table counter specifies the range of data that will be removed from the table, it is important to understand how the table locations are numbered. If you examine the example table, you'll notice that the data locations are numbered from the top of the table. For example, if the table counter started at 6, then all six of the locations would be affected during the instruction execution.

| 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 Counter: 0 0 0 6 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 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.

*DirectSOFT* (optional one-shot method)

| X1 | ( C0<br>PD ) |
| --- | --- |
| C0 | LD<br>K6 |
| | Load the constant value 6 (HEX) into the lower 16 bits of the accumulator |
| | LDA<br>O 1400 |
| | Convert octal 1400 to HEX 300 and load the value into the accumulator. This is the table pointer location. |
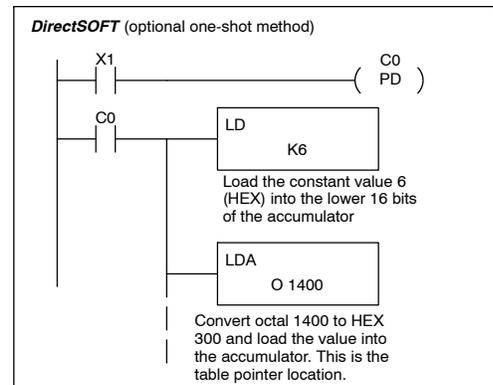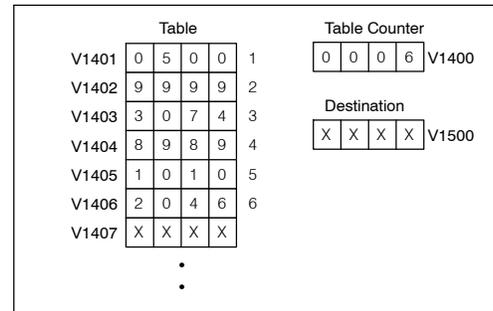
The following diagram shows the scan-by-scan results of the execution for our example program. In our example we're showing the table counter set to 4 initially. (Remember, you can set the table counter to any value that is within the range of the table.) The table counter automatically decrements from 4–0 as the instruction is executed. Notice how the last two table positions, 5 and 6, are not moved up through the table. Also, notice how SP56, which comes on when the table counter is zero, is only on until the end of the scan.

## Example of Execution

### Scan N

**Before RFT Execution**

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 Counter indicates that these 4 positions will be used

Table Counter: `0 0 0 4` V1400

Destination: `X X X X` V1500

SP56 ——| |—— SP56 = OFF

**After RFT Execution**

Table

| | | | | | |
|---|---|---|---|---|---|
| V1401 | 9 | 9 | 9 | 9 | 1 |
| V1402 | 4 | 0 | 7 | 9 | 2 |
| V1403 | 8 | 9 | 8 | 9 | 3 |
| Start here V1404 | 8 | 9 | 8 | 9 | 4 |
| V1405 | 1 | 0 | 1 | 0 | 5 |
| V1406 | 2 | 0 | 4 | 6 | 6 |
| V1407 | X | X | X | X | |

Table Counter (Automatically Decremented): `0 0 0 3` V1400

0 5 0 0

Destination: `0 5 0 0` V1500

SP56 ——| |—— SP56 = OFF

### Scan N+1

**Before RFT Execution**

Table

| | | | | | |
|---|---|---|---|---|---|
| V1401 | 9 | 9 | 9 | 9 | 1 |
| V1402 | 4 | 0 | 7 | 9 | 2 |
| V1403 | 8 | 9 | 8 | 9 | 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 Counter: `0 0 0 3` V1400

Destination: `0 5 0 0` V1500

SP56 ——| |—— SP56 = OFF

**After RFT Execution**

Table

| | | | | | |
|---|---|---|---|---|---|
| V1401 | 4 | 0 | 7 | 9 | 1 |
| V1402 | 8 | 9 | 8 | 9 | 2 |
| Start here V1403 | 8 | 9 | 8 | 9 | 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 Counter (Automatically Decremented): `0 0 0 2` V1400

9 9 9 9

Destination: `9 9 9 9` V1500

SP56 ——| |—— SP56 = OFF

### Scan N+2

**Before RFT Execution**

Table

| | | | | | |
|---|---|---|---|---|---|
| V1401 | 4 | 0 | 7 | 9 | 1 |
| V1402 | 8 | 9 | 8 | 9 | 2 |
| V1403 | 8 | 9 | 8 | 9 | 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 Counter: `0 0 0 2` V1400

Destination: `9 9 9 9` V1500

SP56 ——| |—— SP56 = OFF

**After RFT Execution**

Table

| | | | | | |
|---|---|---|---|---|---|
| V1401 | 8 | 9 | 8 | 9 | 1 |
| Start here V1402 | 8 | 9 | 8 | 9 | 2 |
| V1403 | 8 | 9 | 8 | 9 | 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 Counter (Automatically Decremented): `0 0 0 1` V1400

4 0 7 9

Destination: `4 0 7 9` V1500

SP56 ——| |—— SP56 = OFF

### Scan N+3

**Before RFT Execution**

Table

| | | | | | |
|---|---|---|---|---|---|
| V1401 | 8 | 9 | 8 | 9 | 1 |
| V1402 | 8 | 9 | 8 | 9 | 2 |
| V1403 | 8 | 9 | 8 | 9 | 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 Counter: `0 0 0 1` V1400

Destination: `4 0 7 9` V1500

SP56 ——| |—— SP56 = OFF

**After RFT Execution**

Table

| | | | | | |
|---|---|---|---|---|---|
| Start here V1401 | 8 | 9 | 8 | 9 | 1 |
| V1402 | 8 | 9 | 8 | 9 | 2 |
| V1403 | 8 | 9 | 8 | 9 | 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 Counter (Automatically Decremented): `0 0 0 0` V1400

8 9 8 9

Destination: `8 9 8 9` V1500

SP56 ——| |—— SP56 = ON until end of scan or next instruction that uses SP56

**Add to Top (ATT)**

| × | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | | ✓ |
|---|---|---|
| **DS** | | **HPP** |

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.
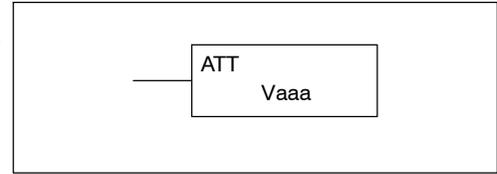
```
        ┌──────────────┐
        │  ATT         │
────────┤     Vaaa     │
        │              │
        └──────────────┘
```

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). The value will be moved from this location.

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 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 | | DL440 Range | DL450 Range |
|---|---|---|---|
| | | **aaa** | **aaa** |
| V-memory | V | All (See p. 3-41) | All (See p. 3-42) |

| Discrete Bit Flags | Description |
|---|---|
| SP56 | on when the table counter is 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 have to 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.

**DirectSOFT**

```
X1
─┤ ├─────────┌─────────────┐
              │ LD          │
              │        K6   │
              └─────────────┘
         Load the constant value 6
         (Hex.) into the lower 16 bits
         of the accumulator

              ┌─────────────┐
              │ LDA         │
              │     O 1400  │
              └─────────────┘
         Convert octal 1400 to HEX
         300 and load the value into
         the accumulator

              ┌─────────────┐
              │ ATT         │
              │     V1500   │
              └─────────────┘
         Copy the specified value
         from V1500 to the table
```

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
|---|---|---|---|

| LD | K(CON) | 6 | ↵ |
|---|---|---|---|

| LD | SHFT | A | OCT | 1 | 4 | 0 | 0 | ↵ |
|---|---|---|---|---|---|---|---|---|

| SHFT | A | T | T | SHFT | V | 1 | 5 | 0 | 0 | ↵ |
|---|---|---|---|---|---|---|---|---|---|---|
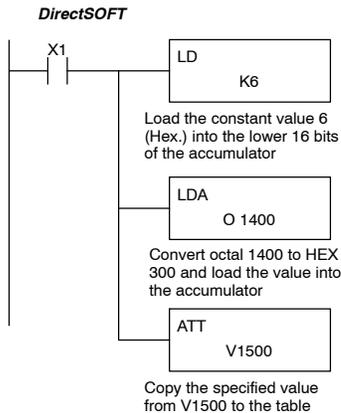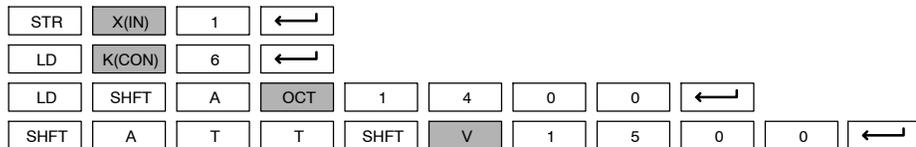
For the ATT instruction, the table counter determines the number of additions that can be made before the instruction will stop executing. So, it is helpful to understand how the system uses this counter to control the execution.

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*

| Table | | | | | |   | Table Counter | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| V1401 | 0 | 5 | 0 | 0 | 1 |   | 0 | 0 | 0 | 2 | V1400 |
| V1402 | 9 | 9 | 9 | 9 | 2 |   | | | | | |
| V1403 | 3 | 0 | 7 | 4 | 3 |   | Data Source | | | | |
| V1404 | 8 | 9 | 8 | 9 | 4 |   | X | X | X | X | V1500 |
| V1405 | 1 | 0 | 1 | 0 | 5 |   | | | | | |
| V1406 | 2 | 0 | 4 | 6 | 6 |   | | | | | |
| V1407 | X | X | X | X | |   | | | | | |

(e.g., 6 – 2 = 4).

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.
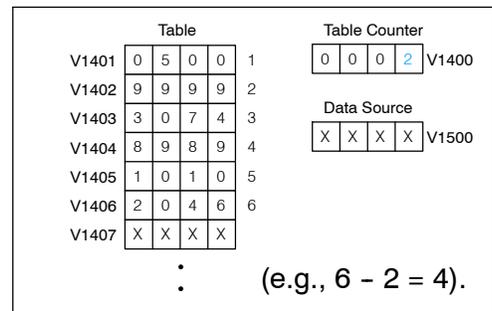
**DirectSOFT** (optional one-shot method)

```
X1                                    C0
─┤ ├─────────────────────────────────( PD )

C0
─┤ ├─────────┌─────────────┐
              │ LD          │
              │        K6   │
              └─────────────┘
         Load the constant value 6
         (HEX) into the lower 16 bits
         of the accumulator

              ┌─────────────┐
              │ LDA         │
              │     O 1400  │
              └─────────────┘
         Convert octal 1400 to HEX
         300 and load the value into
         the accumulator. This is the
         starting table location.
```
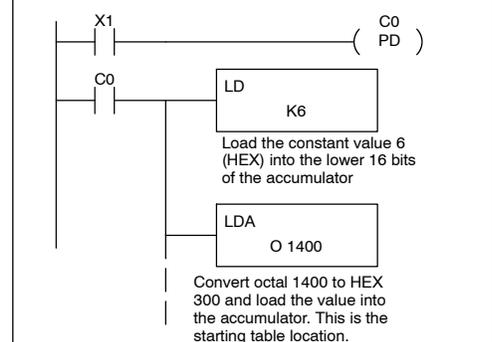
The following diagram shows the scan-by-scan results of the execution for our example program. The table counter is set to 2 initially, and it will automatically increment from 2 – 6 as the instruction is executed. Notice how SP56 comes on when the table counter is 6, which is equal to the table length. 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

**Before ATT Execution**

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 Counter: 0 0 0 2 V1400
Data Source: 1 2 3 4 V1500
SP56 = OFF

**After ATT Execution**

Table

| V1401 | 1 | 2 | 3 | 4 | 1 |
| V1402 | 0 | 5 | 0 | 0 | 2 |
| V1403 | 9 | 9 | 9 | 9 | 3 |
| V1404 | 3 | 0 | 7 | 4 | 4 |
| V1405 | 8 | 9 | 8 | 9 | 5 |
| V1406 | 1 | 0 | 1 | 0 | 6 |
| V1407 | X | X | X | X | |

Table Counter (Automatically Incremented): 0 0 0 3 V1400
Data Source: 1 2 3 4 V1500
SP56 = OFF
Discard Bucket: 2 0 4 6

### Scan N+1

**Before ATT Execution**

Table

| V1401 | 1 | 2 | 3 | 4 | 1 |
| V1402 | 0 | 5 | 0 | 0 | 2 |
| V1403 | 9 | 9 | 9 | 9 | 3 |
| V1404 | 3 | 0 | 7 | 4 | 4 |
| V1405 | 8 | 9 | 8 | 9 | 5 |
| V1406 | 1 | 0 | 1 | 0 | 6 |
| V1407 | X | X | X | X | |

Table Counter: 0 0 0 3 V1400
Data Source: 5 6 7 8 V1500
SP56 = OFF

**After ATT Execution**

Table

| V1401 | 5 | 6 | 7 | 8 | 1 |
| V1402 | 1 | 2 | 3 | 4 | 2 |
| V1403 | 0 | 5 | 0 | 0 | 3 |
| V1404 | 9 | 9 | 9 | 9 | 4 |
| V1405 | 3 | 0 | 7 | 4 | 5 |
| V1406 | 8 | 9 | 8 | 9 | 6 |
| V1407 | X | X | X | X | |

Table Counter (Automatically Incremented): 0 0 0 4 V1400
Data Source: 5 6 7 8 V1500
SP56 = OFF
Discard Bucket: 1 0 1 0

### Scan N+2

**Before ATT Execution**

Table

| V1401 | 5 | 6 | 7 | 8 | 1 |
| V1402 | 1 | 2 | 3 | 4 | 2 |
| V1403 | 0 | 5 | 0 | 0 | 3 |
| V1404 | 9 | 9 | 9 | 9 | 4 |
| V1405 | 3 | 0 | 7 | 4 | 5 |
| V1406 | 8 | 9 | 8 | 9 | 6 |
| V1407 | X | X | X | X | |

Table Counter: 0 0 0 4 V1400
Data Source: 4 3 3 4 V1500
SP56 = OFF

**After ATT Execution**

Table

| V1401 | 4 | 3 | 4 | 3 | 1 |
| V1402 | 5 | 6 | 7 | 8 | 2 |
| V1403 | 1 | 2 | 3 | 4 | 3 |
| V1404 | 0 | 5 | 0 | 0 | 4 |
| V1405 | 9 | 9 | 9 | 9 | 5 |
| V1406 | 3 | 0 | 7 | 4 | 6 |
| V1407 | X | X | X | X | |

Table Counter (Automatically Incremented): 0 0 0 5 V1400
Data Source: 4 3 3 4 V1500
SP56 = OFF
Discard Bucket: 8 9 8 9

### Scan N+3

**Before ATT Execution**

Table

| V1401 | 4 | 3 | 4 | 3 | 1 |
| V1402 | 5 | 6 | 7 | 8 | 2 |
| V1403 | 1 | 2 | 3 | 4 | 3 |
| V1404 | 0 | 5 | 0 | 0 | 4 |
| V1405 | 9 | 9 | 9 | 9 | 5 |
| V1406 | 3 | 0 | 7 | 4 | 6 |
| V1407 | X | X | X | X | |

Table Counter: 0 0 0 5 V1400
Data Source: 7 7 7 7 V1500
SP56 = OFF

**After ATT Execution**

Table

| V1401 | 7 | 7 | 7 | 7 | 1 |
| V1402 | 4 | 3 | 4 | 3 | 2 |
| V1403 | 5 | 6 | 7 | 8 | 3 |
| V1404 | 1 | 2 | 3 | 4 | 4 |
| V1405 | 0 | 5 | 0 | 0 | 5 |
| V1406 | 9 | 9 | 9 | 9 | 6 |
| V1407 | X | X | X | X | |

Table Counter (Automatically Incremented): 0 0 0 6 V1400
Data Source: 7 7 7 7 V1500
SP56 = ON until end of scan or next instruction that uses SP56
Discard Bucket: 3 0 7 4

**Move Memory Cartridge/ Load Label (MOVMC/LDLBL)**

| ✗ | ✓ | ✓ |
|---|---|---|

**430  440  450**

| ✓ | | ✓ |
|---|---|---|

**DS      HPP**

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 Move Memory Cartridge and Load Label functions.

```
        ┌──────────┐
────────┤ MOVMC    │
        │   A aaa  │
        └──────────┘
```

```
        ┌──────────┐
────────┤ LDLBL    │
        │   K aaa  │
        └──────────┘
```
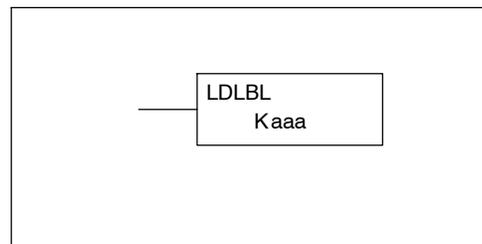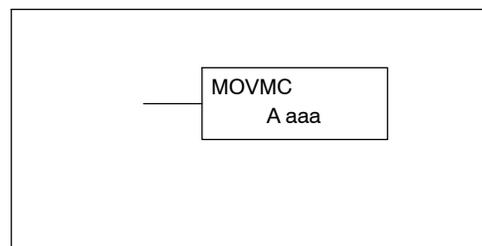
Step 1:— Load the number of words (255 maximum) to be copied into the second level of the accumulator stack. This must be a hex value, 0 to FF.

Step 2:— Load the offset for the data label area (in HEX) in the program ladder memory and the beginning of the V-memory block into the first level of the accumulator 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. The value will be copied from this location. 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 (Aaaa). The value will be copied to this location.

| Operand Data Type | | DL440 Range | DL450 Range |
|---|---|---|---|
| | **A** | **aaa** | **aaa** |
| V-memory | V | All (See p. 3-41) | All (See p. 3-42) |
| Constant | K | 1-FFFF | 1-FFFF |

| Discrete Bit Flags | Description |
|---|---|
| SP53 | on if there is a table pointer error. |

**NOTE:** Status flags are only valid until:
— the end of the scan
— or another instruction that uses the same flag is executed.
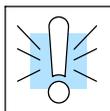
**WARNING: The offset for this usage of the instruction starts at 0, but may be any number that *does not* result in data outside of the source data area being copied into the destination table. When an offset is outside of the source information boundaries, unknown data values will be transferred into the destination table.**

**Copy Data From a Data Label Area to V-memory**

| X | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | | ✓ |
|---|---|---|
| **DS** | | **HPP** |

In the following example, 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 instruction. This value specifies the length of the destination 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 using the Load instruction. This value specifies the offset for the source *and* the destination table, and 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 table starting location and executes the copying of data from the source Data Label Area to V-memory.

*DirectSOFT*

```
X1
─┤ ├────┬──────────────────┐
         │  LD              │   Load the value 4 into the
         │        K4        │   accumulator specifying the
         └──────────────────┘   number of locations to be copied.

         ┌──────────────────┐
         │  LD              │   Load the value 0 into the
         │        K0        │   accumulator specifying the
         └──────────────────┘   offset for source and
                                destination locations

         ┌──────────────────┐
         │  LDLBL           │   Load the value 1 into the
         │        K1        │   accumulator specifying the
         └──────────────────┘   Data Label Area K1 as the
                                starting address of the data
                                to be copied.

         ┌──────────────────┐
         │  MOVMC           │   V1400 is the destination
         │        V1400     │   starting address for the data
         └──────────────────┘   to be copied.


      ─────(  END  )─────

┌──────────────────┐
│  DLBL            │
│        K1        │
└──────────────────┘

   ┌──────────────────┐
   │  NCON            │
   │        K 1234    │
   └──────────────────┘

   ┌──────────────────┐
   │  NCON            │
   │        K 4532    │
   └──────────────────┘

   ┌──────────────────┐
   │  NCON            │
   │        K 6151    │
   └──────────────────┘

   ┌──────────────────┐
   │  NCON            │
   │        K 8845    │
   └──────────────────┘

   ┌──────────────────┐
   │  NCON            │
   │        K 7777    │
   └──────────────────┘
```

Handheld Programmer Keystrokes (for MOVMC portion only)

| STR | X(IN) | 1 | ↵ | | | | | |
|-----|-------|---|---|---|---|---|---|---|
| LD | K(CON) | 4 | ↵ | | | | | |
| LD | K(CON) | 0 | ↵ | | | | | |
| LD | SHFT | L | B | L | SHFT | K(CON) | 1 | ↵ |
| SHFT | M | O | V | M | C | | | |
| SHFT | V | 1 | 4 | 0 | 0 | ↵ | | |

**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.**

The following diagram shows the result of our example. The offset is equal to zero and four words will be copied into the Data Label area.

Example of Execution
Offset = 0, move 4 words

Before MOVMC Execution
Destination

| | | | | |
|---|---|---|---|---|
| V1400 | 0 | 9 | 0 | 0 |
| V1401 | 0 | 5 | 0 | 0 |
| V1402 | 9 | 9 | 9 | 9 |
| V1403 | 3 | 0 | 7 | 4 |
| V1404 | 8 | 9 | 8 | 9 |
| V1405 | 1 | 0 | 1 | 0 |
| V1406 | X | X | X | X |

After MOVMC Execution
Data Label Area
DLBL K1

Start here →
| N | C | O | N |
|---|---|---|---|
| K 1 | 2 | 3 | 4 |
| N | C | O | N |
| K 4 | 5 | 3 | 2 |
| N | C | O | N |
| K 6 | 1 | 5 | 1 |
| N | C | O | N |
| K 8 | 8 | 4 | 5 |
| N | C | O | N |
| K 7 | 7 | 7 | 7 |

Destination

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | V1400 | Start here |
| 4 | 5 | 3 | 2 | V1401 | |
| 6 | 1 | 5 | 1 | V1402 | |
| 8 | 8 | 4 | 5 | V1403 | |
| 8 | 9 | 8 | 9 | V1404 | |
| 1 | 0 | 1 | 0 | V1405 | |
| X | X | X | X | V1406 | |
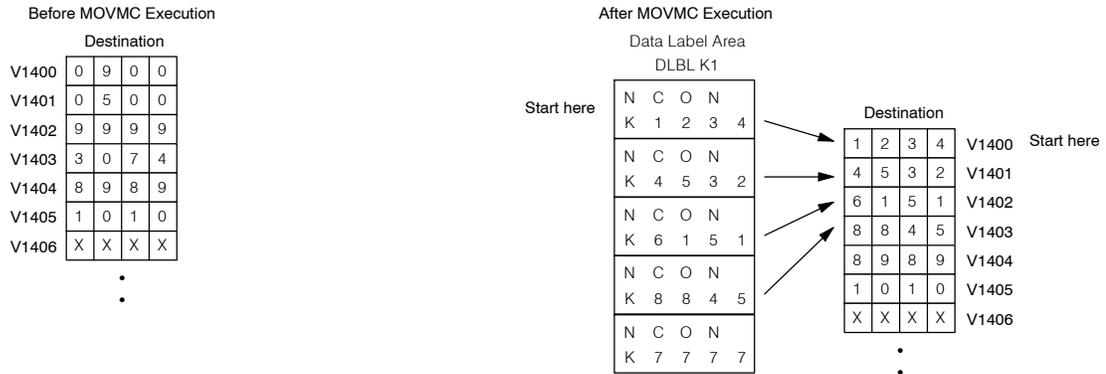
The example is fairly straightforward when an offset of zero is used. However, it is also helpful for you to understand the results that would have been obtained if different offset values (1 and 2) were used. Notice how the offset is used for both the data label (source) *and* the destination table. Also, notice how an improper offset (two in this case) can result in unknown values being copied into the destination table.

Offset = 1, move 4 words

Before MOVMC Execution
Destination

| | | | | |
|---|---|---|---|---|
| V1400 | 0 | 9 | 0 | 0 |
| V1401 | 0 | 5 | 0 | 0 |
| V1402 | 9 | 9 | 9 | 9 |
| V1403 | 3 | 0 | 7 | 4 |
| V1404 | 8 | 9 | 8 | 9 |
| V1405 | 1 | 0 | 1 | 0 |
| V1406 | X | X | X | X |

After MOVMC Execution
Data Label Area
DLBL K1

Offset {
| N | C | O | N |
|---|---|---|---|
| K 1 | 2 | 3 | 4 |

Start here →
| N | C | O | N |
|---|---|---|---|
| K 4 | 5 | 3 | 2 |
| N | C | O | N |
| K 6 | 1 | 5 | 1 |
| N | C | O | N |
| K 8 | 8 | 4 | 5 |
| N | C | O | N |
| K 7 | 7 | 7 | 7 |

Destination

| | | | | | |
|---|---|---|---|---|---|
| 0 | 9 | 0 | 0 | V1400 | |
| 4 | 5 | 3 | 2 | V1401 | Start here |
| 6 | 1 | 5 | 1 | V1402 | |
| 8 | 8 | 4 | 5 | V1403 | |
| 7 | 7 | 7 | 7 | V1404 | |
| 1 | 0 | 1 | 0 | V1405 | |
| X | X | X | X | V1406 | |

Offset = 2, move 4 words

Before MOVMC Execution
Destination

| | | | | |
|---|---|---|---|---|
| V1400 | 0 | 9 | 0 | 0 |
| V1401 | 0 | 5 | 0 | 0 |
| V1402 | 9 | 9 | 9 | 9 |
| V1403 | 3 | 0 | 7 | 4 |
| V1404 | 8 | 9 | 8 | 9 |
| V1405 | 1 | 0 | 1 | 0 |
| V1406 | X | X | X | X |

After MOVMC Execution
Data Label Area
DLBL K1

Offset {
| N | C | O | N |
|---|---|---|---|
| K 1 | 2 | 3 | 4 |
| N | C | O | N |
| K 4 | 5 | 3 | 2 |

Start here →
| N | C | O | N |
|---|---|---|---|
| K 6 | 1 | 5 | 1 |
| N | C | O | N |
| K 8 | 8 | 4 | 5 |
| N | C | O | N |
| K 7 | 7 | 7 | 7 |
| ? | ? | ? | ? | ? |

Since there is no NCON, the CPU does not know where to get the data. Unknown values will be copied into V1405.

Destination

| | | | | | |
|---|---|---|---|---|---|
| 0 | 9 | 0 | 0 | V1400 | |
| 0 | 5 | 0 | 0 | V1401 | |
| 6 | 1 | 5 | 1 | V1402 | Start here |
| 8 | 8 | 4 | 5 | V1403 | |
| 7 | 7 | 7 | 7 | V1404 | |
| ? | ? | ? | ? | V1405 | |
| X | X | X | X | V1406 | |

**Copy Data From V-memory to a Data Label Area**

| × | ✓ | × |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | | ✓ |
|---|---|---|
| **DS** | | **HPP** |

**NOTE:** You must use a RAM cartridge for this example to work.

In this example, data is copied from V-memory to a data label area. When X1 is on, the constant value (K4) is loaded into the accumulator using the Load instruction. This value specifies the length of the destination table and is placed in the second stack location after the next Load and Load Address instructions are executed. The constant value (K0) is loaded into the accumulator using the Load instruction. This value specifies the offset for the source *and* destination table, and is placed in the first stack location after the Load Address instruction is executed. The source address data is being copied from is loaded into the accumulator using the Load Address instruction. The MOVMC instruction specifies the destination starting location and executes the copying of data from V-memory to the data label area.

*DirectSOFT*
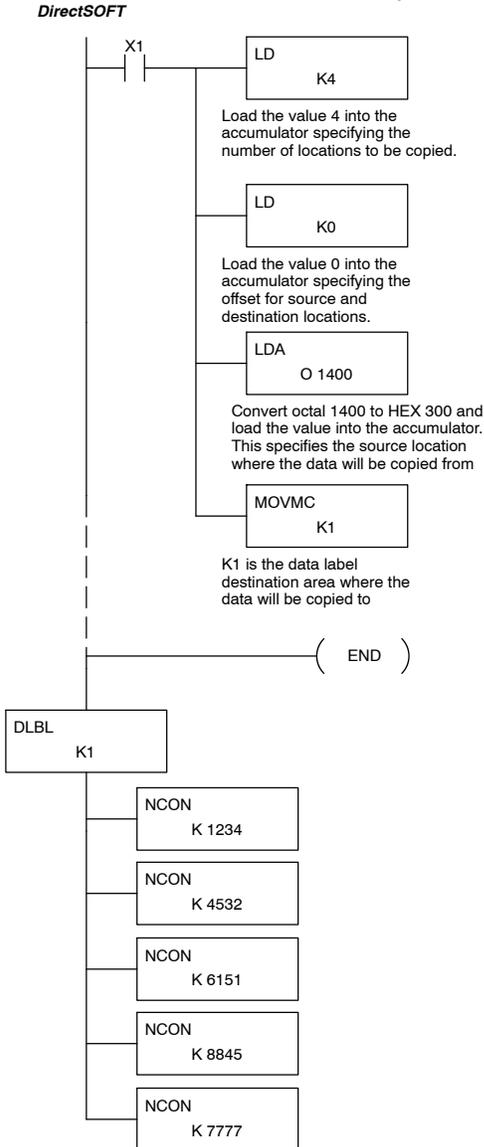
Handheld Programmer Keystrokes (for MOVMC portion only)

| STR | X(IN) | 1 | ↵ | | | | | |
|-----|-------|---|---|---|---|---|---|---|
| LD | K(CON) | 4 | ↵ | | | | | |
| LD | K(CON) | 0 | ↵ | | | | | |
| LD | SHFT | A | OCT | 1 | 4 | 0 | 0 | ↵ |
| SHFT | M | O | V | M | C | SHFT | K(CON) | 1 |

```
      X1
 ──────┤ ├──────┬──── LD
                │     K4
```
Load the value 4 into the accumulator specifying the number of locations to be copied.

```
                ├──── LD
                │     K0
```
Load the value 0 into the accumulator specifying the offset for source and destination locations.

```
                ├──── LDA
                │     O 1400
```
Convert octal 1400 to HEX 300 and load the value into the accumulator. This specifies the source location where the data will be copied from

```
                ├──── MOVMC
                      K1
```
K1 is the data label destination area where the data will be copied to

Note: This instruction works only with the RAM cartridge. It does not work with the EEPROM.

```
 ──────────────────( END )
```

```
 DLBL
 K1
   ├──── NCON
   │     K 1234
   ├──── NCON
   │     K 4532
   ├──── NCON
   │     K 6151
   ├──── NCON
   │     K 8845
   ├──── NCON
         K 7777
```
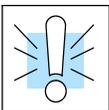
**WARNING: The offset for this usage of the instruction starts at 0. If the offset (or the specified data table range) is large enough to cause data to be copied from V-memory to beyond the end of the DLBL area, then anything after the specified DLBL area will be replaced with invalid instructions.**

The following diagram shows the result of our example. The offset is equal to zero and four words will be copied into the Data Label area.

Example of Execution
Offset = 0, move 4 words



Before MOVMC Execution

Data Label Area
DLBL K1

After MOVMC Execution

Destination

Data Label Area
DLBL K1
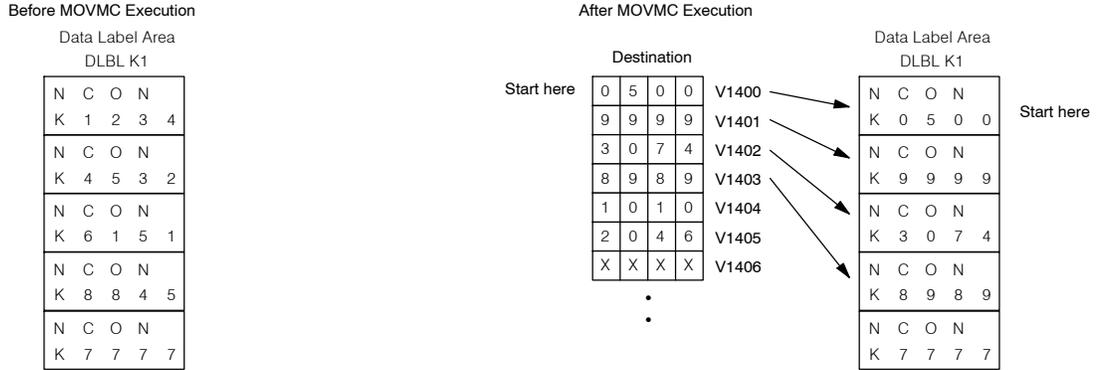
The example is fairly straightforward when an offset of zero is used. However, it is also helpful for you to understand the results that would have been obtained if different offset values (1 and 2) were used. Notice how the offset is used for both the V-memory data table (source) *and* the Data Label area. Also, notice how an improper offset (two in this case) can result in invalid instructions being written over any instructions that follow the Data Label.

Offset = 1, move 4 words



Before MOVMC Execution

Data Label Area
DLBL K1

After MOVMC Execution

Destination

Data Label Area
DLBL K1

Offset = 2, move 4 words



Before MOVMC Execution

Data Label Area
DLBL K1

After MOVMC Execution

Destination

Data Label Area
DLBL K1

Invalid Instruction
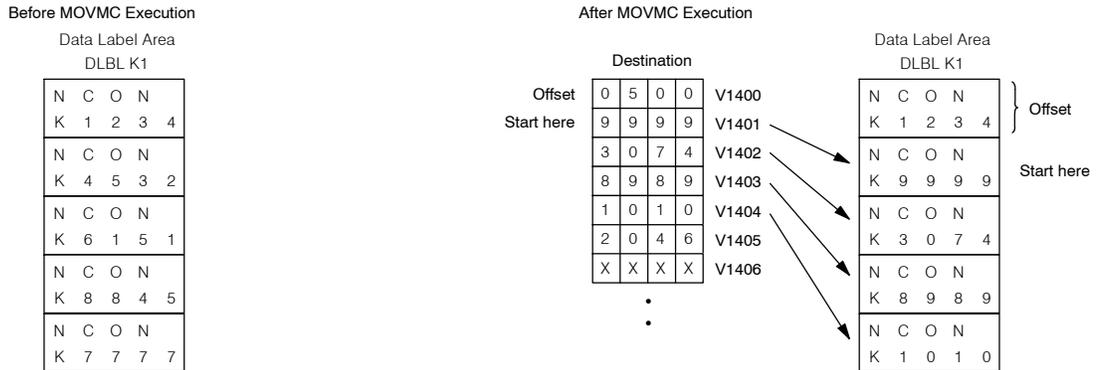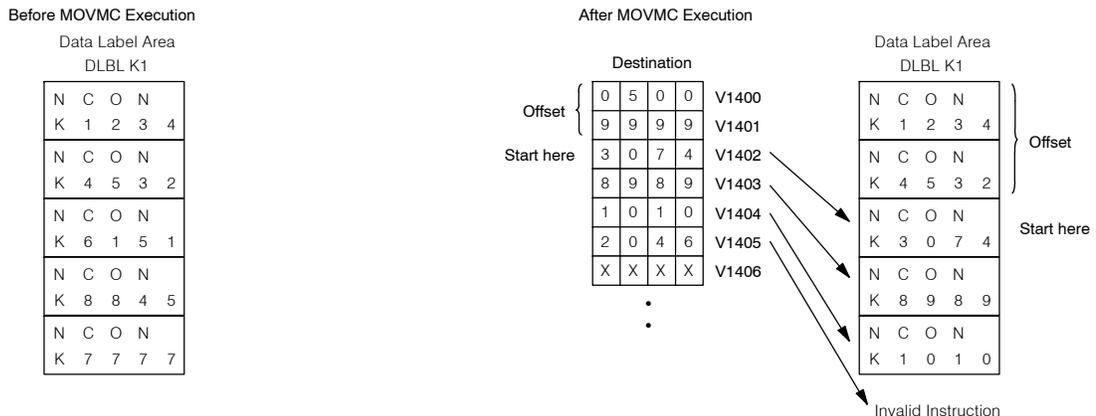
**Set Bit
(SETBIT)**

| × | × | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

**Reset Bit
(RSTBIT)**

| × | × | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

The Set Bit instruction sets a single bit to one within a range of V-memory locations.

```
        ┌─────────────┐
────────┤   SETBIT    │
        │    A aaa    │
        └─────────────┘
```
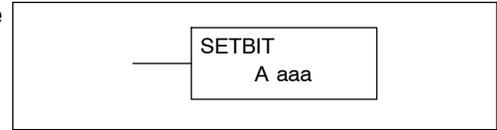
The Reset Bit instruction resets a single bit to zero within a range of V-memory locations.
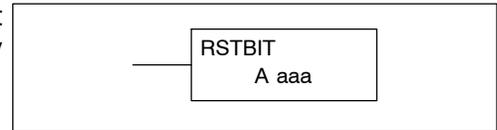
```
        ┌─────────────┐
────────┤   RSTBIT    │
        │    A aaa    │
        └─────────────┘
```

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. Flag 53 will be set if the bit specified is outside the range of the table.

| Operand Data Type | | DL450 Range |
|---|---|---|
| | | **aaa** |
| V-memory | V | All (See p. 3-42) |
| Octal Address | O | 0-7777 |

| Discrete Bit Flags | Description |
|---|---|
| SP53 | on when the bit number which is referenced in the Set Bit or Reset Bit exceeds the range of the table |

**NOTE:** Status flags are only valid until:
— the end of the scan
— or 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".

```
              V3000
MSB                        LSB
┌─┬─┬─┬─┬─┬─┬─┬─┬─┬─┬─┬─┬─┬─┬─┬─┐
└─┴─┴─┴─┴─┴─┴─┴─┴─┴─┴─┴─┴─┴─┴─┴─┘
│←────────── 16 bits ──────────→│

              V3001
MSB                        LSB
┌─┬─┬─┬─┬─┬─┬─┬─┬─┬─┬─┬─┬─┬─┬─┬─┐
└─┴─┴─┴▓┴─┴─┴─┴─┴─┴─┴─┴─┴─┴─┴─┴─┘
 1 1 1 1 1 1 1 1 7 6 5 4 3 2 1 0
 7 6 5 4 3 2 1 0
```
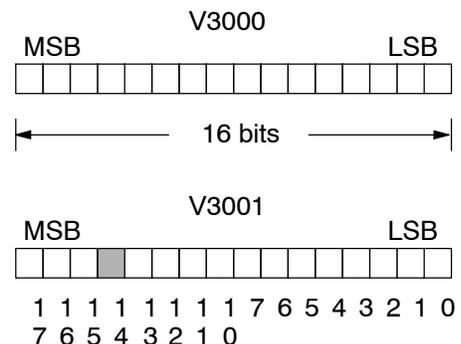
In this ladder example, we will use input X0 to trigger the Set Bit operation. First, we will load the table length (2 words) into the accumulator stack. Next, we load the starting address into the accumulator. Since V3000 is an octal number we have to convert it to hex by using the LDA command. Finally, we use the Set Bit (or Reset Bit) instruction and specify the octal address of the bit (bit 34), referenced from the table beginning.

*DirectSOFT*

| X0 | LD<br>K2 | Load the constant value 2 (Hex.) into the lower 16 bits of the accumulator. |
| | LDA<br>O 3000 | Convert octal 3000 to HEX and load the value into the accumulator. This is the table beginning. |
| | SETBIT<br>O 34 | Set bit 34 (octal) in the table to a "1". |
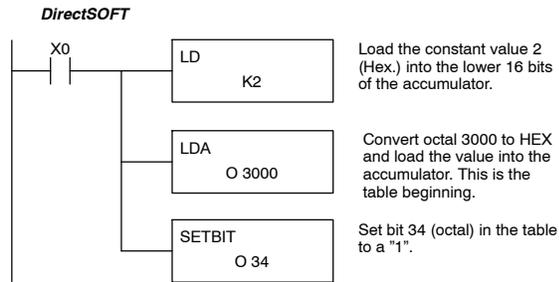
Handheld Programmer Keystrokes

| STR | X(IN) | 0 | ↵ |
| LD | K(CON) | 2 | ↵ |
| LD | SHFT | A | OCT | 3 | 0 | 0 | 0 | ↵ |
| SET | SHFT | B | I | T | OCT | 3 | 4 | ↵ |

**Table Shift Left (TSHFL)**

| × | × | ✓ |
|---|---|---|

**430 440 450**

| ✓ | | ✓ |
|---|---|---|

**DS      HPP**

The Table Shift Left instruction shifts all the bits in a V-memory table to the left, the specified number of bit positions.

```
          ┌──────────┐
          │ TSHFL    │
──────────│  A aaa   │
          └──────────┘
```
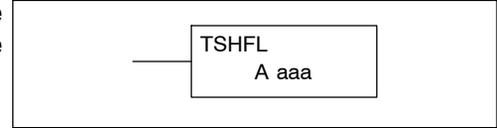
**Table Shift Right (TSHFR)**

| × | × | ✓ |
|---|---|---|

**430 440 450**

| ✓ | | ✓ |
|---|---|---|

**DS      HPP**

The Table Shift Right instruction shifts all the bits in a V-memory table to the right, a specified number of bit positions.

```
          ┌──────────┐
          │ TSHFR    │
──────────│  A aaa   │
          └──────────┘
```
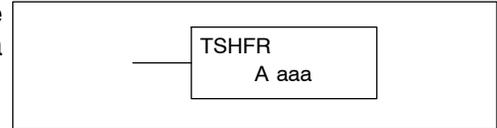
The following description applies to both the Table Shift Left and Table Shift Right instructions. A table is just a range of V-memory locations. The Table Shift Left and Table Shift Right instructions shift bits serially throughout the entire table. Bits are shifted out the end of one word and into 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.
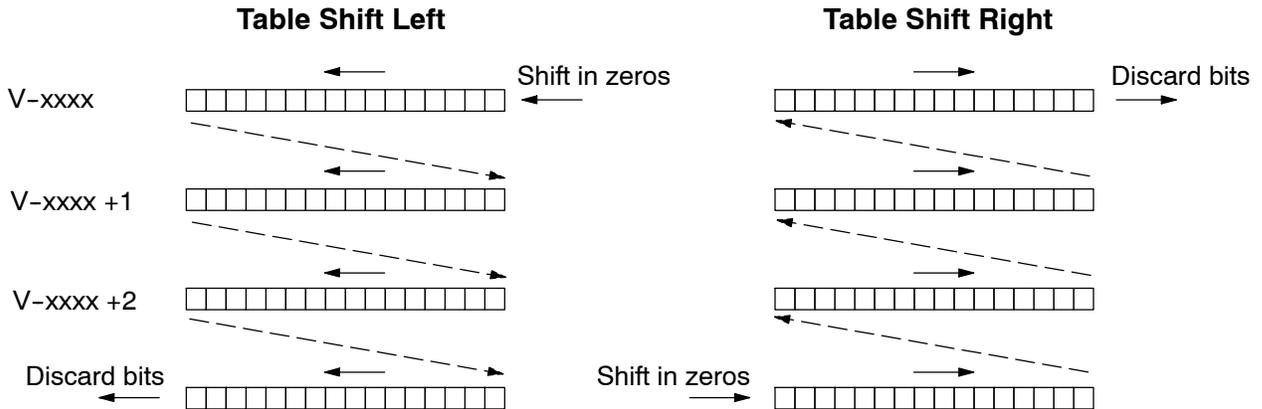
**Table Shift Left**                    **Table Shift Right**



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. Flag 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 | | DL450 Range |
|---|---|---|
| | | aaa |
| V-memory | V | All (See p. 3-42) |

Standard
RLL Instructions

| 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.

V3000

| 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 |
| 1 | 1 | 2 | 2 |
| 3 | 3 | 4 | 4 |
| 5 | 5 | 6 | 6 |

→

V3000

| 6 | 7 | 8 | 1 |
| 1 | 2 | 2 | 5 |
| 3 | 4 | 4 | 1 |
| 5 | 6 | 6 | 3 |
| 0 | 0 | 0 | 5 |

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*

| X0 | LD K5 | Load the constant value 5 (Hex.) into the lower 16 bits of the accumulator. |
|---|---|---|
|  | LDA O 3000 | Convert octal 3000 to HEX and load the value into the accumulator. This is the table beginning. |
|  | TSHFR O 14 | Do a table shift right by 12 bits, which is 14 octal. |

Handheld Programmer Keystrokes

| STR | X(IN) | 0 | ↵ |
| LD | K(CON) | 5 | ↵ |
| LD | SHFT | A | OCT | 3 | 0 | 0 | 0 | ↵ |
| SHFT | T | S | H | F | R | OCT | 1 | 4 | ↵ |
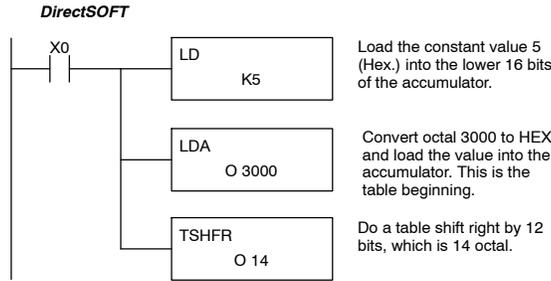
Standard
RLL Instructions
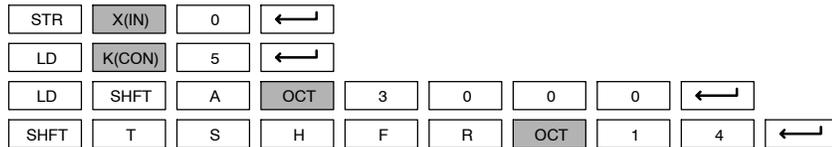
**AND Move (ANDMOV)**

| × | × | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | | ✓ |
|---|---|---|
| **DS** | | **HPP** |

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.

```
        ┌─────────────┐
────────┤  ANDMOV     │
        │  A aaa      │
        └─────────────┘
```
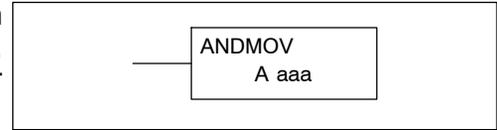
**OR Move (ORMOV)**

| × | × | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | | ✓ |
|---|---|---|
| **DS** | | **HPP** |

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.

```
        ┌─────────────┐
────────┤  LDR        │
        │  A aaa      │
        └─────────────┘
```
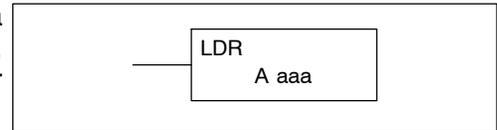
**Exclusive OR Move (XORMOV)**

| × | × | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | | ✓ |
|---|---|---|
| **DS** | | **HPP** |

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.

```
        ┌─────────────┐
────────┤  XORMOV     │
        │  A aaa      │
        └─────────────┘
```
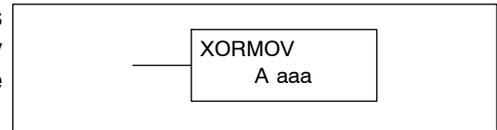
The following description applies to the AND Move, OR Move, and Exclusive OR Move instructions. A table is just a range of V-memory locations. These instructions copy the data of a table to another specified location, preforming a logical operation on each word with the accumulator contents as the new 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.
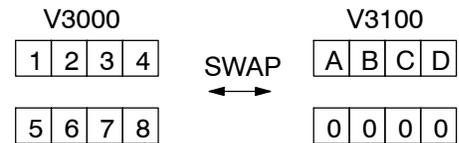
Step 2: — Load the starting V–memory location for the table into the accumulator. This parameter must be a HEX value. You can use the LDA instruction to convert an octal address to hex.

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 | DL450 Range |
|---|---|
| | **aaa** |
| V–memory            V | All (See p. 3–42) |

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.

```
  V3000                              V3100
┌─┬─┬─┬─┐      ANDMOV            ┌─┬─┬─┬─┐
│3│3│3│3│      K6666             │2│2│2│2│
└─┴─┴─┴─┘                        └─┴─┴─┴─┘
┌─┬─┬─┬─┐       ──▶              ┌─┬─┬─┬─┐
│F│F│F│F│                        │6│6│6│6│
└─┴─┴─┴─┘                        └─┴─┴─┴─┘
```

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.
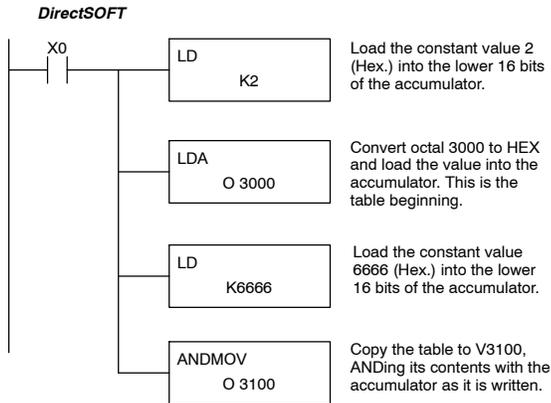
**DirectSOFT**

X0

| LD | |
|----|----|
| | K2 |

Load the constant value 2 (Hex.) into the lower 16 bits of the accumulator.

| LDA | |
|-----|----|
| | O 3000 |

Convert octal 3000 to HEX and load the value into the accumulator. This is the table beginning.

| LD | |
|----|----|
| | K6666 |

Load the constant value 6666 (Hex.) into the lower 16 bits of the accumulator.

| ANDMOV | |
|--------|----|
| | O 3100 |

Copy the table to V3100, ANDing its contents with the accumulator as it is written.

Handheld Programmer Keystrokes

| STR | X(IN) | 0 | ↵ | | | | |
|-----|-------|---|---|---|---|---|---|
| LD | K(CON) | 5 | ↵ | | | | |
| LD | SHFT | A | OCT | 3 | 0 | 0 | 0 | ↵ |
| AND | SHFT | M | O | V | | | |
| OCT | 3 | 1 | 0 | 0 | ↵ | | |
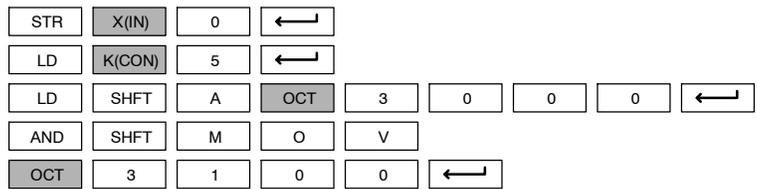
The example to the right shows a table of two words at V3000 and logically ORs it with K8888. The copy of the table at V3100 shows the result of the OR operation for each word.

V3000

| 1 | 1 | 1 | 1 |
|---|---|---|---|

| 1 | 1 | 1 | 1 |
|---|---|---|---|

ORMOV
K8888
→

V3100

| 9 | 9 | 9 | 9 |
|---|---|---|---|

| 9 | 9 | 9 | 9 |
|---|---|---|---|
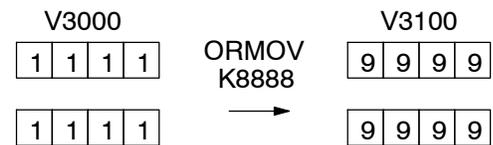
The program to the right performs the ORMOV example above. It assumes that the data in the table at V3000 – V3001 already exists. First we load the table length (two words) into the accumulator. Next we load the starting address of the source table, using the LDA instruction. Then we load the data into the accumulator to be ORed with the table. In the ORMOV command, we specify the table destination, V3100.

**DirectSOFT**

X0

| LD | |
|----|----|
| | K2 |

Load the constant value 2 (Hex.) into the lower 16 bits of the accumulator.

| LDA | |
|-----|----|
| | O 3000 |

Convert octal 3000 to HEX and load the value into the accumulator. This is the table beginning.

| LD | |
|----|----|
| | K8888 |

Load the constant value 8888 (Hex.) into the lower 16 bits of the accumulator.

| ORMOV | |
|-------|----|
| | O 3100 |

Copy the table to V3100, ORing its contents with the accumulator as it is written.

Handheld Programmer Keystrokes

| STR | X(IN) | 0 | ↵ | | | | |
|-----|-------|---|---|---|---|---|---|
| LD | K(CON) | 5 | ↵ | | | | |
| LD | SHFT | A | OCT | 3 | 0 | 0 | 0 | ↵ |
| OR | SHFT | M | O | V | | | |
| OCT | 3 | 1 | 0 | 0 | ↵ | | |
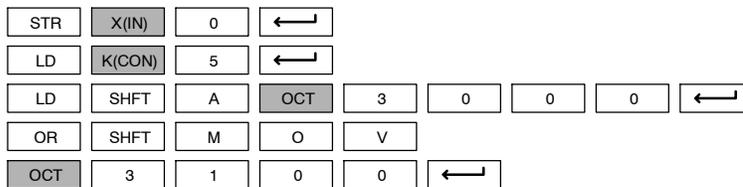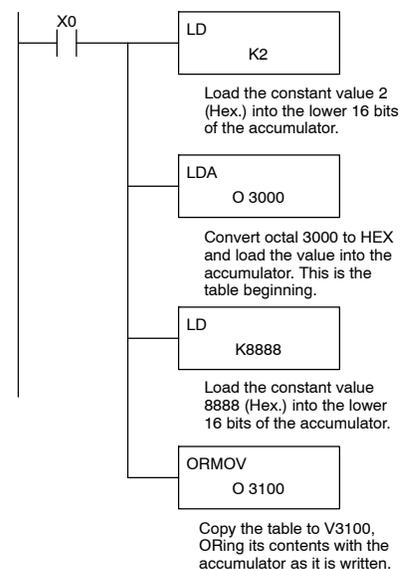
The example to the right shows a table of two words at V3000 and logical XORs it with K3333. The copy of the table at V3100 shows the result of the XOR operation for each word.

V3000

| 1 | 1 | 1 | 1 |
|---|---|---|---|

| 1 | 1 | 1 | 1 |
|---|---|---|---|

XORMOV
K3333
→

V3100

| 2 | 2 | 2 | 2 |
|---|---|---|---|

| 2 | 2 | 2 | 2 |
|---|---|---|---|

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.

Standard
RLL Instructions

## Find Block (FINDB)

| × | × | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

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.

```
        ┌──────────────┐
────────┤ FINDB        │
        │    A aaa     │
        └──────────────┘
```
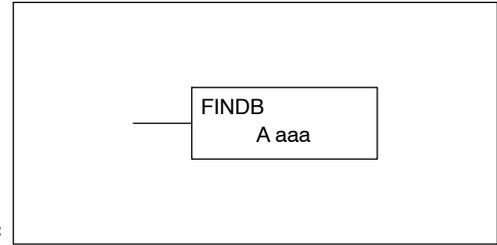
| Operand Data Type | | DL450 Range |
|---|---|---|
| | | **aaa** |
| V-memory | V | All (See p. 3-42) |
| V-memory | P | All (See p. 3-42) |

| Discrete Bit Flags | Description |
|---|---|
| SP53 | on when the Find Block instruction was executed but did not find the block of data in table specified |

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.

Start Addr.

```
        ┌──────────────┐ ┐
        │   Table 1    │ │ Number
        ├──────────────┤ ┘ of words
        │   Table 2    │
        ├──────────────┤
        │   Table 3    │
        ├──────────────┤
        │              │
        ├──────────────┤
        │              │
        ├──────────────┤
        │   Table n    │
        └──────────────┘
```
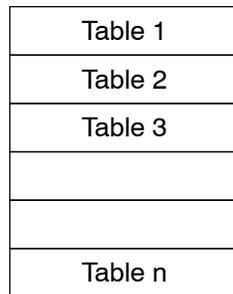
End Addr.

Start Addr.

```
        ┌──────────────┐ ┐
        │              │ │ Number
        │    Block     │ │ of bytes
        │              │ ┘
        └──────────────┘
```
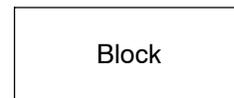
**Swap
(SWAP)**

| × | × | ✔ |
|---|---|---|
| **430** | **440** | **450** |

| ✔ | | ✔ |
|---|---|---|
| **DS** | | **HPP** |

The Swap instruction exchanges the data in two tables of equal length.
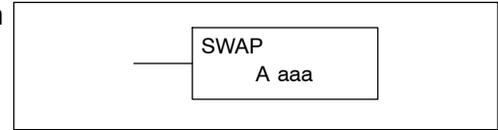
```
        ┌──────────────┐
────────┤ SWAP         │
        │   A aaa      │
        └──────────────┘
```

The following description applies to both the Set Bit and Reset Bit table instructions.

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 | | DL450 Range |
|---|---|---|
| | | **aaa** |
| V–memory | V | All (See p. 3–42) |

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.

| V3000 | | | | | | V3100 | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | SWAP | | A | B | C | D |
| 5 | 6 | 7 | 8 | ←→ | | 0 | 0 | 0 | 0 |

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.

*DirectSOFT*

```
  X0        ┌──────────────┐     Load the constant value 2
──┤↑├───┬───┤ LD           │     (Hex.) into the lower 16 bits
       │   │          K2   │     of the accumulator.
       │   └──────────────┘
       │   ┌──────────────┐     Convert octal 3000 to HEX
       ├───┤ LDA          │     and load the value into the
       │   │      O 3000   │     accumulator. This is the
       │   └──────────────┘     table beginning.
       │   ┌──────────────┐     Swap the contents of the
       └───┤ SWAP         │     table in the previous
           │      O 3100   │     instruction with the one at
           └──────────────┘     V3100.
```

Handheld Programmer Keystrokes

| STR | X(IN) | SHFT | P | D | SHFT | 0 | ↵ |
|---|---|---|---|---|---|---|---|
| LD | K(CON) | 2 | ↵ |
| LD | SHFT | A | OCT | 3 | 0 | 0 | 0 | ↵ |
| SHFT | S | W | A | P | OCT | 3 | 1 | 0 | 0 | ↵ |
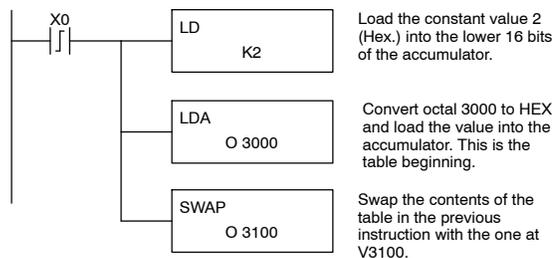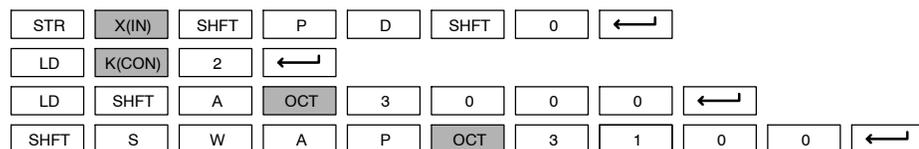
# Clock/Calender Instructions

**Date
(DATE)**

| ✗ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | | ✓ |
|---|---|---|
| **DS** | | **HPP** |

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).

```
          ┌─────────┐
      ────┤ DATE    │
          │   V aaa │
          └─────────┘
```
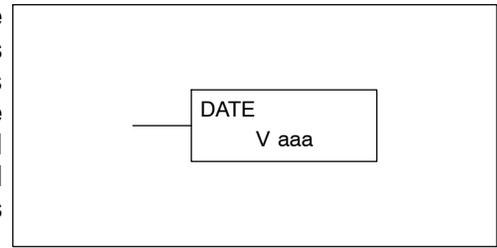
| Date | Range | V Memory Location (BCD) (READ Only) |
|------|-------|--------------------------------------|
| Year | 0-99 | V7774 |
| Month | 1-12 | V7773 |
| Day | 1-31 | V7772 |
| Day of Week | 0-06 | V7771 |

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 | | DL440 Range | DL450 Range |
|-------------------|---|-------------|-------------|
| | **A** | **aaa** | **aaa** |
| V-memory | **V** | All (See p. 3-41) | All (See p. 3-42) |

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.



**DirectSOFT**

Load the constant value (K94010301) into the accumulator

Copy the value in the accumulator to V2000 and V2001

Set the date in the CPU using the value in V2000 and V2001

In this example, the Date instruction uses the value set in V2000 and V2001 to set the date in the appropriate V-memory locations (V7771-V7774)

Format

| V2001 | | | V2000 | | |
|---|---|---|---|---|---|
| 9 | 4 | 0 | 1 | 0 | 3 | 0 | 1 |

Year  Month  Day  Day of Week
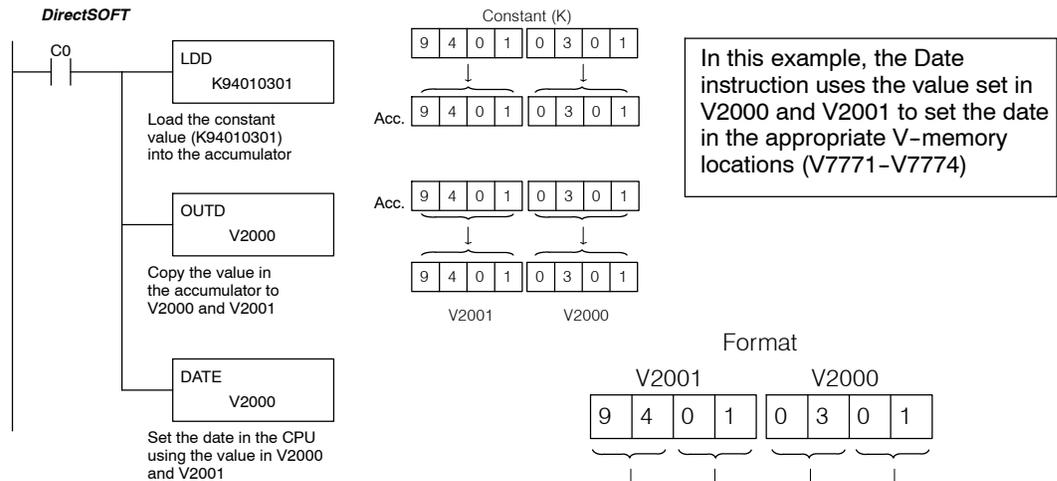
Handheld Programmer Keystrokes

| STR | C(CR) | 0 | ↵ |
| LD | SHFT | D | SHFT | K(CON) | 9 | 4 | 0 | 1 |
| 0 | 3 | 0 | 1 | ↵ |
| OUT | SHFT | D | SHFT | V | 2 | 0 | 0 | 0 | ↵ |
| SHFT | D | A | T | E | SHFT | V | 2 | 0 | 0 | 0 | ↵ |

Standard
RLL Instructions

**Time
(TIME)**

| | | |
|---|---|---|
| ✕ | ✓ | ✓ |
| **430** | **440** | **450** |

| | |
|---|---|
| ✓ | ✓ |
| **DS** | **HPP** |

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 V–memory locations V7747 and V7766–V7770.

```
        ┌─────────────┐
────────┤ TIME        │
        │  V aaa       │
        └─────────────┘
```
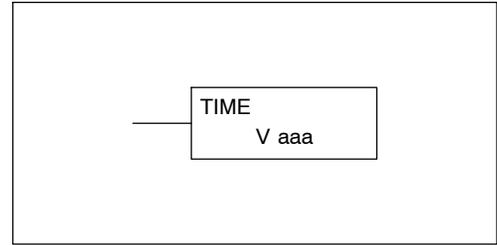
| Date | Range | V Memory Location (BCD) (READ Only) |
|---|---|---|
| 1/100 seconds (10ms) | 0–99 | V7747 |
| Seconds | 0–59 | V7766 |
| Minutes | 0–59 | V7767 |
| Hour | 0–23 | V7770 |

| Operand Data Type | | DL440 Range | DL440 Range |
|---|---|---|---|
| | **A** | **aaa** | **aaa** |
| V-memory | V | All (See p. 3–41) | All (See p. 3–41) |

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.

*DirectSOFT*

```
 C0          ┌─────────────┐
──┤├─────────┤ LDD         │
             │   K73000     │
             └─────────────┘
             Load the constant
             value (K73000) into
             the accumulator

             ┌─────────────┐
             │ OUTD        │
             │   V2000      │
             └─────────────┘
             Copy the value in the
             accumulator to V2000
             and V2001

             ┌─────────────┐
             │ TIME        │
             │   V2000      │
             └─────────────┘
             Set the time in the CPU
             using the value in V2000
             and V2001
```

Constant (K)

| 0 | 0 | 0 | 7 | 3 | 0 | 0 | 0 |

Acc. | 0 | 0 | 0 | 7 | 3 | 0 | 0 | 0 |

Acc. | 0 | 0 | 0 | 7 | 3 | 0 | 0 | 0 |

| 0 | 0 | 0 | 7 | 3 | 0 | 0 | 0 |

V2001    V2000
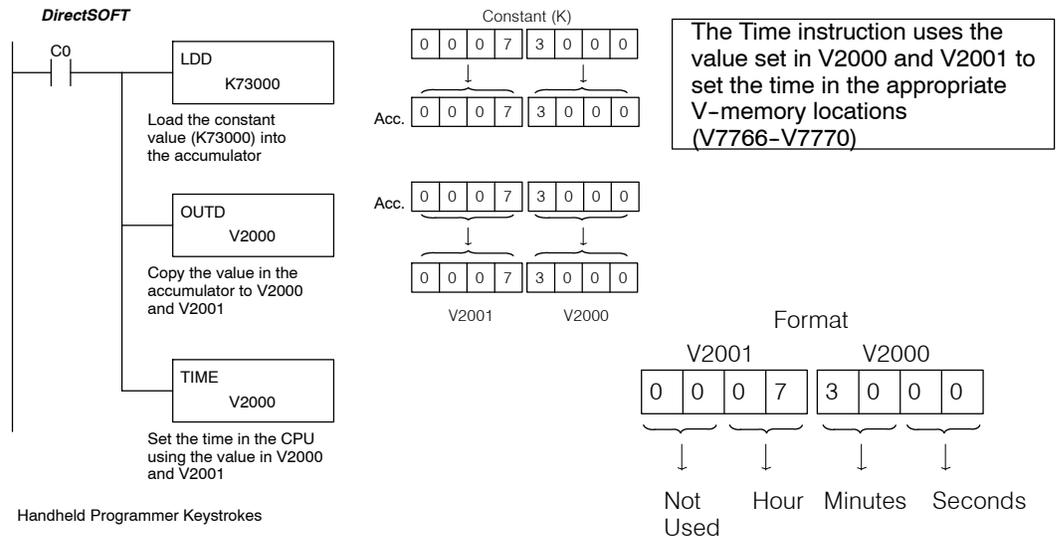
The Time instruction uses the value set in V2000 and V2001 to set the time in the appropriate V–memory locations (V7766–V7770)

Format

V2001          V2000

| 0 | 0 | 0 | 7 | 3 | 0 | 0 | 0 |

Not Used | Hour | Minutes | Seconds

Handheld Programmer Keystrokes

| STR | C(CR) | 0 | ↵ |

| LD | SHFT | D | SHFT | K(CON) | 7 | 3 | 0 | 0 | 0 | ↵ |

| OUT | SHFT | D | SHFT | V | 2 | 0 | 0 | 0 | ↵ |

| SHFT | T | I | M | E | SHFT | V | 2 | 0 | 0 | 0 | ↵ |

# CPU Control Instructions

**No Operation (NOP)**

| × | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | | ✓ |
|---|---|---|
| **DS** | | **HPP** |

The No Operation is an empty (not programmed) memory location. These instructions are just place-holders in the program. So, you will not need to program them, because they automatically appear after the end of the program.

—( NOP )

*DirectSOFT*

——————( NOP )

Handheld Programmer Keystrokes

| SHFT | N | O | P | ↵ |
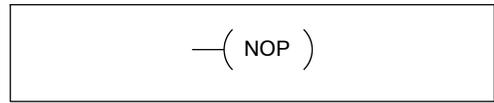
**End (END)**

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | | ✓ |
|---|---|---|
| **DS** | | **HPP** |

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.

—( END )

*DirectSOFT*

——————( END )

Handheld Programmer Keystrokes

| END | ↵ |
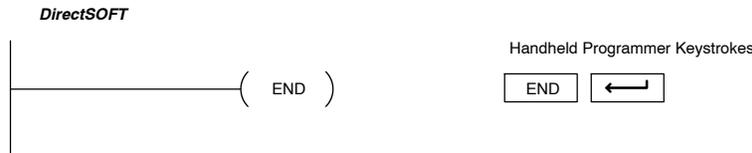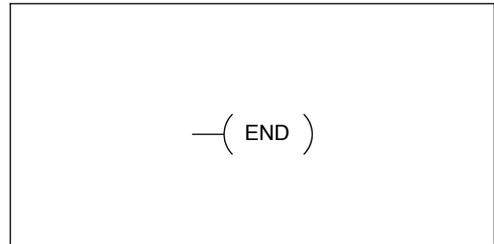
**Stop (STOP)**

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | | ✓ |
|---|---|---|
| **DS** | | **HPP** |

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 a shutdown condition such as a I/O module failure.
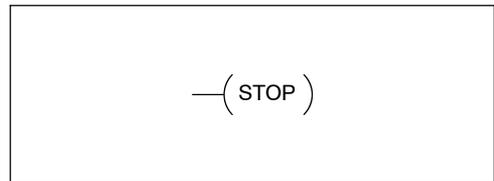
—( STOP )

In the following example, when SP45 comes on indicating a I/O module failure, the CPU will stop operation and switch to the program mode.

*DirectSOFT*

SP45
——| |——————( STOP )

SP45 will turn on
if there is an I/O
module falure

Handheld Programmer Keystrokes

| STR | SPCL | 4 | 5 | ↵ |
| SHFT | S | T | O | P | ↵ |

Standard
RLL Instructions
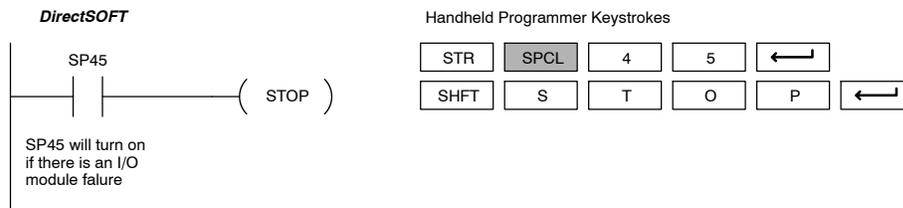
**Break
(BREAK)**

| ✕ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

The Break instruction changes the operational mode of the CPU from Run to the Test Program mode. This instruction is typically used to aid in debugging an application program. The Break instruction allows V–memory and image register data to be retained where it would be normally cleared with the Stop instruction or a normal Run to Program transition.

—(BREAK)
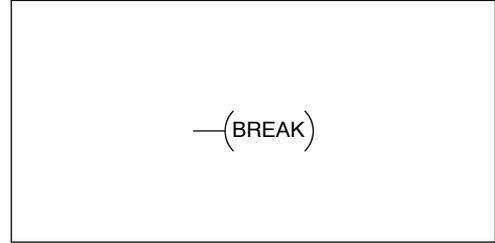
In the following example when C10 turns on, the CPU will stop operation and switch to the Test program mode.

*DirectSOFT*

```
     C10
——| |——————————( BREAK )
```

Handheld Programmer Keystrokes

| STR | C(CR) | 1 | 0 | ↵ |
|-----|-------|---|---|---|

| SHFT | B | R | E | A | K | ↵ |
|------|---|---|---|---|---|---|
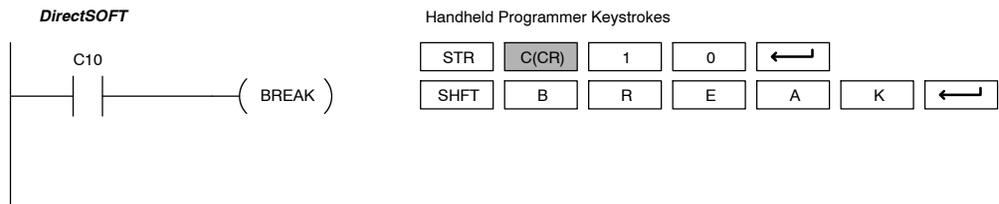
**Reset Watch Dog
Timer
(RSTWT)**

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

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.
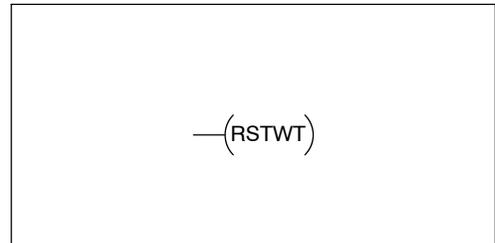
—(RSTWT)

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*

```
——————————————( RSTWT )
```

Handheld Programmer Keystrokes

| SHFT | R | S | T | W | T | ↵ |
|------|---|---|---|---|---|---|
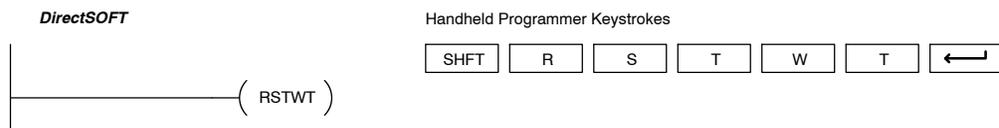
# Program Control Instructions

**Goto/Label
(GOTO/LBL)**

| ✕ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

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 128 Goto instructions and 64 LBL instructions can be used in the program.

```
          K aaa
    ─( GOTO )
```

```
    LBL        K aaa
```
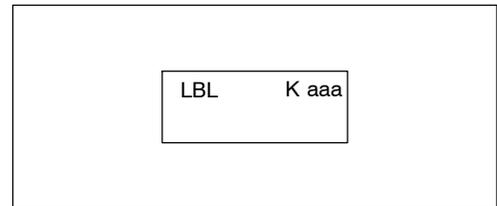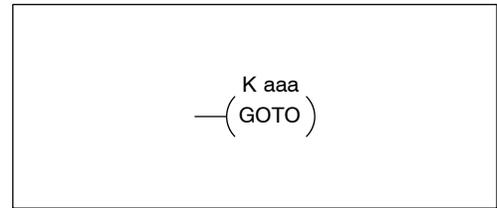
| Operand Data Type | | DL440 Range | DL450 Range |
|---|---|---|---|
| | | **aaa** | **aaa** |
| Constant | K | 1–FFFF | 1–FFFF |

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.

*DirectSOFT*

```
        C7                    K5
    ────┤ ├──────────────( GOTO )

        X1                    C2
    ────┤ ├──────────────( OUT )

        .
        .
        .

    ┌─────────────────┐
    │ LBL        K5   │
    └─────────────────┘

        X5                    Y2
    ────┤ ├──────────────( OUT )
```

Handheld Programmer Keystrokes

| STR | C(CR) | 7 | ↵ |

| SHFT | G | O | T | O | SHFT | K(CON) | 5 | ↵ |

| STR | X(IN) | 1 | ↵ |

| OUT | C(CR) | 2 | ↵ |

.
.

| SHFT | L | B | L | SHFT | K(CON) | 5 | ↵ |

| STR | X(IN) | 5 | ↵ |

| OUT | Y(OUT) | 2 | ↵ |
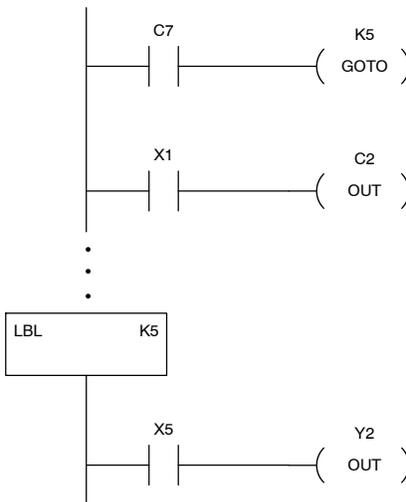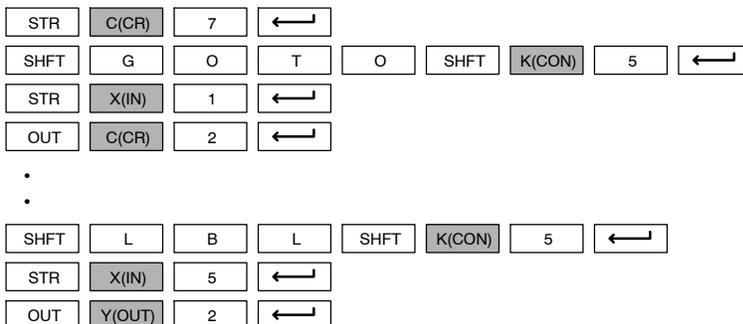
## For/Next
## (FOR/NEXT)

| ✕ | ✓ | ✓ |
|---|---|---|

**430 440 450**

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

The For and Next instructions are used to execute a section of ladder logic between the For and Next instructions 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.

```
        A aaa
   ──( FOR )
```

```
   ──( NEXT )
```
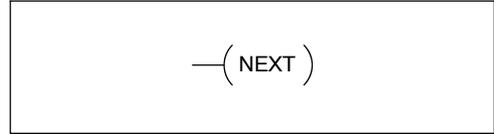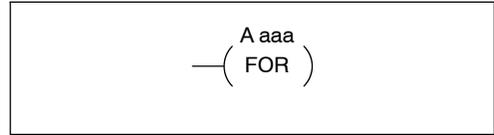
For / Next instructions cannot be nested. Up to 64 For / Next loops may be used in a program. If the maximum number of For / Next loops is exceeded, error E413 will occur. The normal I/O update and CPU housekeeping is suspended while executing the For / Next loop. The program scan can increase significantly, 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 | | DL440 Range | DL450 Range |
|---|---|---|---|
| | **A** | **aaa** | **aaa** |
| V-memory | V | All (See p. 3–41) | All (See p. 3–42) |
| Constant | K | 1–9999 | 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 larger the Watch Dog Timer setting. For more information on the Watch Dog Timer, refer to the RSTWT instruction.

*DirectSOFT*

```
        X1                    K3
     ──┤ ├──────────────────( FOR )
                                              1   2   3

                            ──( RSTWT )

        X20                   Y5
     ──┤↑├──────────────────( OUTI )

                            ──( NEXT )
```

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ← |
| SHFT | F | O | R | SHFT | K(CON) | 3 | ← |
| SHFT | R | S | T | W | T | ← |
| STR | SHFT | I | SHFT | X(IN) | 2 | 0 | ← |
| Y(OUT) | SHFT | I | SHFT | Y(OUT) | 5 | ← |
| SHFT | N | E | X | T | ← |
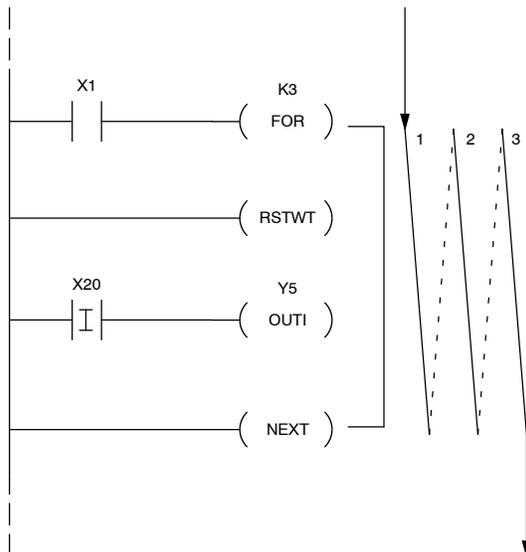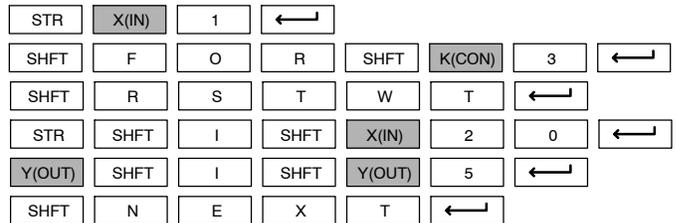
**Goto Subroutine
(GTS)**

| × | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

The Goto Subroutine instruction allows a section of ladder logic to be placed outside the main body of the program execute only when needed. There can be a maximum of 192 (DL440) and an unlimited amount for DL450 GTS instructions.

```
                        K aaa
                   —( GTS )
```
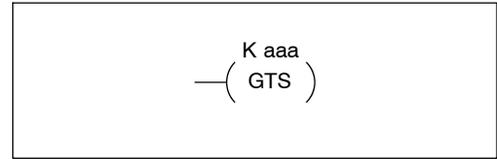
Upon completion of executing the subroutine, program execution returns to the main program immediately after the GTS instruction. GTS instructions can be nested up to 8 levels. An error E412 will occur if the maximum limits are exceeded. Typically this will be used in an application where a block of program logic may be slow to execute and is not required to execute every scan.

| Operand Data Type | | DL440 Range | DL450 Range |
|---|---|---|---|
| | | aaa | aaa |
| Constant | K | 1-FFFF | 1-FFFF |

**Subroutine
(SBR)**

| × | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

The subroutine label and all associated logic is placed after the End statement in the program. There can be a maximum of 64 (DL440) and 256 (DL450) SBR instructions used in a 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.

```
   SBR      K aaa
```
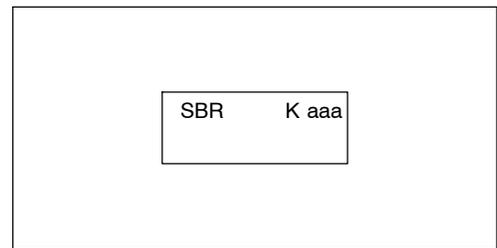
By placing code in a subroutine it is only scanned and executed when needed since it resides after the End instruction. Code which is not scanned does not impact the overall scan time of the program.

| Operand Data Type | | DL440 Range | DL450 Range |
|---|---|---|---|
| | | aaa | aaa |
| Constant | K | 1-FFFF | 1-FFFF |

**Subroutine Return
(RT)**

| × | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

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 which must be the last instruction in the subroutine and is a stand alone instruction (no input contact on the rung).

```
              —( RT )
```
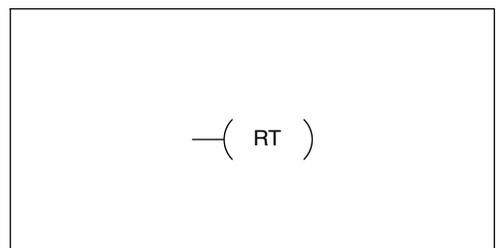
**Subroutine Return
Conditional
(RTC)**

| × | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

The Subroutine Return Conditional instruction is a optional instruction used with a input contact to implement a conditional return from the subroutine. The Subroutine Return (RT) is still required for termination of the Subroutine.

```
              —( RTC )
```

Standard
RLL Instructions
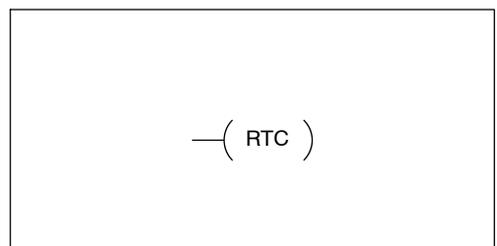
In the following example, when X1 is on, Subroutine K3 will be called. The CPU will jump to the Subroutine Label K3 and the ladder logic in the subroutine will be executed. If X35 is on the CPU will return to the main program at the RTC instruction. If X35 is not on Y0–Y17 will be reset to off and then the CPU will return to the main body of the program.
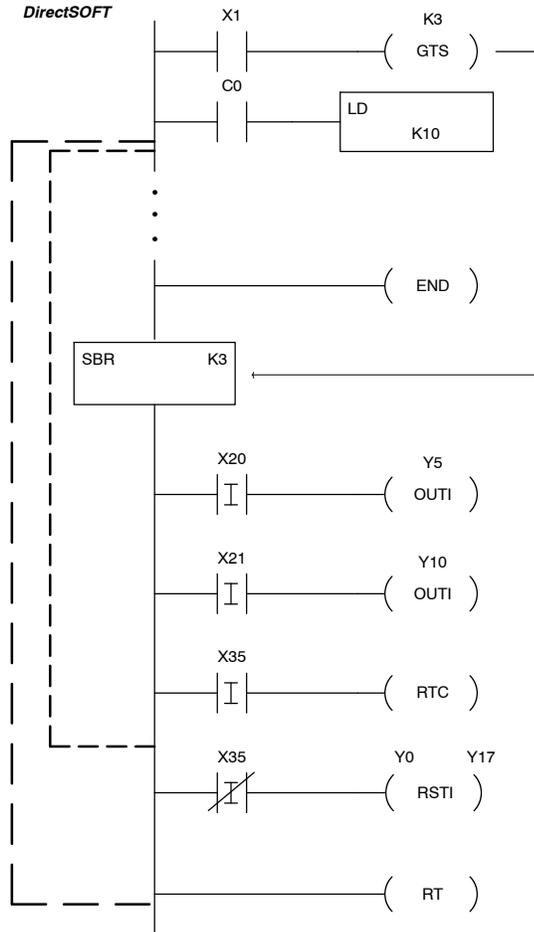


Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ← |
| SHFT | G | T | S | SHFT | K(CON) | 3 | ← |

...

| END | ← |
| SHFT | S | B | R | SHFT | K(CON) | 3 | ← |
| STR | SHFT | I | SHFT | X(IN) | 2 | 0 | ← |
| OUT | SHFT | I | SHFT | Y(OUT) | 5 | ← |
| STR | SHFT | I | SHFT | X(IN) | 2 | 1 | ← |
| OUT | SHFT | I | SHFT | Y(OUT) | 1 | 0 | ← |
| STR | SHFT | I | SHFT | X(IN) | 3 | 5 | ← |
| SHFT | R | T | C | ← |
| STR | NOT | SHFT | I | SHFT | X(IN) | 3 | 5 | ← |
| RST | SHFT | I | SHFT | Y(OUT) | 0 | Y(OUT) | 1 | 7 | ← |
| SHFT | R | T | ← |

**Master Line Set
(MLS)**

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | | ✓ |
|---|---|---|
| **DS** | | **HPP** |
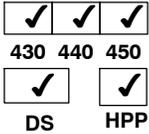
The Master Line Set instruction allows the program to control sections of ladder logic by forming a new power rail controlled by the main left power rail. The main left rail is always master line 0. When a MLS K1 instruction is used, a new power rail is created at level 1. Master Line Sets and Master Line Resets can be used to nest power rails up to seven levels deep.

```
        K aaa
    —( MLS )
```
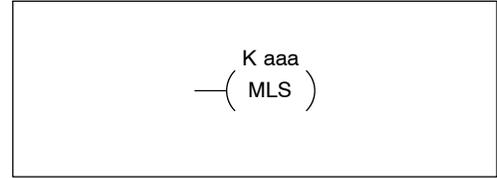
| Operand Data Type | | DL430 Range | DL440 Range | DL450 Range |
|---|---|---|---|---|
| | | aaa | aaa | aaa |
| Constant | K | 1–7 | 1–7 | 1–7 |

**Master Line Reset
(MLR)**

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | | ✓ |
|---|---|---|
| **DS** | | **HPP** |
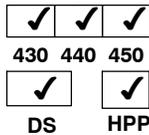
The Master Line Reset instruction marks the end of control for the corresponding MLS instruction. The MLR reference is one less than the corresponding MLS.

```
        K aaa
    —( MLR )
```
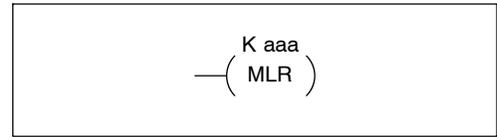
| Operand Data Type | | DL430 Range | DL440 Range | DL450 Range |
|---|---|---|---|---|
| | | aaa | aaa | aaa |
| Constant | K | 0–6 | 0–6 | 1–6 |

**Understanding
Master Control
Relays**

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | | ✓ |
|---|---|---|
| **DS** | | **HPP** |

**MLS/MLR Example**
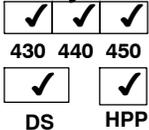
The Master Line Set (MLS) and Master Line Reset (MLR) instructions allow you to quickly control the power flow for sections of the RLL program. This provides program control flexibility. The following example shows how the MLS and MLR instructions operate by creating a sub power rail for control logic.

In the following MLS/MLR example, logic between the first MLS K1 (A) and MLR K0 (B) will only have power flow present at the power rail if input X0 is on. (Note, if X0 is off the logic will still be scanned, but there is no power flow.) The logic between the MLS K2 (C) and MLR K1 (D) will only have power flow present at the power rail if input X10 and X0 are on. The last rung is not controlled by either of the MLS coils and always has power flow present at the beginning of the rung.

Remember, the MLS / MLR instructions control the power flow between the power rails. It does not control the execution of the instructions. The instructions are still executed, but since there is no power flow, the logic cannot turn on the output coils. Consider the following case for our example.

1.  X0 is off, which means that there is no power flow at the second rung.
2.  You use a programming device to turn on C0.

Since there is no power flow at the second rung (STR X1, OUT C0), then you would expect that C0 would remain on, since you turned it on with the programming device. However, the MLS instruction does not mean that the instructions within the zone of control are not executed. They *are* in fact executed, but with no power flow. So in our case, C0 would be turned off when the rung was executed. This is because the CPU sees that there is no power flow present at the rung. When it executes this rung, it will turn off C0.

**DirectSOFT**

| | |
|---|---|
| X0 — K1 MLS | When contact X0 is on, logic under the first MLS will be executed. |
| X1 — Y7 OUT | |
| X2 — K2 MLS | When contact X0 and X2 are ON, logic under the second MLS will be executed. |
| X3 — Y10 OUT | |
| — K1 MLR | |
| — K0 MLR | The MLR instructions note the end of the Master Control area. |
| X10 — Y11 OUT | |

**DirectSOFT**

Handheld Programmer Keystrokes

| X0 — K1 MLS | A |
| X1 — C0 OUT | |
| X2 — C1 OUT | |
| X3 — Y0 OUT | |
| X10 — K2 MLS | C |
| X5 — Y1 OUT | |
| X4 — Y2 OUT | |
| — K1 MLR | D |
| X5 — C2 OUT | |
| X6 — Y3 OUT | |
| — K0 MLR | B |
| X7 — Y22 OUT | |

| | | | |
|---|---|---|---|
| STR | X(IN) | 0 | ↵ |
| MLS | K(CON) | 1 | ↵ |
| STR | X(IN) | 1 | ↵ |
| OUT | C(CR) | 0 | ↵ |
| STR | X(IN) | 2 | ↵ |
| OUT | C(CR) | 1 | ↵ |
| STR | X(IN) | 3 | ↵ |
| OUT | Y(OUT) | 0 | ↵ |
| STR | X(IN) | 1 | 0 | ↵ |
| MSL | K(CON) | 2 | ↵ |
| STR | X(IN) | 5 | ↵ |
| OUT | Y(OUT) | 1 | ↵ |
| STR | X(IN) | 4 | ↵ |
| OUT | Y(OUT) | 2 | ↵ |
| MLR | K(CON) | 1 | ↵ |
| STR | X(IN) | 5 | ↵ |
| OUT | C(CR) | 2 | ↵ |
| STR | X(IN) | 6 | ↵ |
| OUT | Y(OUT) | 3 | ↵ |
| MLR | K(CON) | 0 | ↵ |
| STR | X(IN) | 7 | ↵ |
| OUT | Y(OUT) | 2 | 2 | ↵ |

Standard
RLL Instructions

DL405 User Manual, 4th Edition, Rev. A

# Interrupt Instructions

**Interrupt
(INT)**

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | | ✓ |
|---|---|---|
| **DS** | | **HPP** |

The Interrupt instruction allows a section of ladder logic to be placed outside the main body of the program and executed when needed. Interrupts can be called from the program or an interrupt module can be installed in slot 0 to provide 8 interrupt inputs.

```
INT        O aaa
```
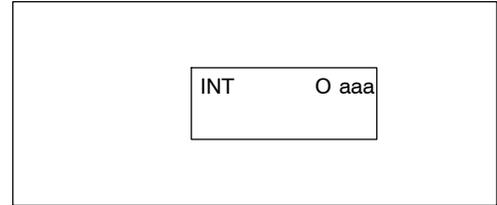
One interrupt module can be installed in a DL430 system (X0–X7) and two interrupt modules can be installed in a DL440 or DL450 System (X0–X7, and X20–X27). Remember, the interrupt modules consume 16 points.

The two software interrupts use interrupt #16 and #17 which means the hardware interrupts #16 and #17 and the software interrupt cannot be used together.

Typically, interrupts will be used in an application where a fast response to an input is needed or a program section needs to 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 the interrupt routine is called from the interrupt module or software interrupt, the CPU will complete execution of the instruction it is currently processing in ladder logic, then execute the designated interrupt routine. There are two software interrupts and INT 16 and INT 17. Once the interrupt is serviced, the program execution will continue from where it was before the interrupt occurred.

The software interrupts are setup by programming the interrupt times in V736 and V737. The valid range is 3–999ms. The value must be a BCD value. The interrupt will not execute if the value is out of range.

See the example program of a software interrupt.

| Operand Data Type | | DL430 Range | DL440 Range | DL450 Range |
|---|---|---|---|---|
| | | **aaa** | **aaa** | **aaa** |
| Constant | O | 0–7 | 0–17 | 0–17 |

| Software | | DL430 | | DL440/450 | |
|---|---|---|---|---|---|
| Interrupt Input | Interrupt Routine | Interrupt Input | Interrupt Routine | Interrupt Input | Interrupt Routine |
| -- | -- | X0 | INT 0 | X0 | INT 0 |
| -- | -- | X1 | INT 1 | X1 | INT 1 |
| -- | -- | X2 | INT 2 | X2 | INT 2 |
| -- | -- | X3 | INT 3 | X3 | INT 3 |
| -- | -- | X4 | INT 4 | X4 | INT 4 |
| -- | -- | X5 | INT 5 | X5 | INT 5 |
| -- | -- | X6 | INT 6 | X6 | INT 6 |
| -- | -- | X7 | INT 7 | X7 | INT 7 |
| -- | -- | -- | -- | X20 | INT 10 |
| -- | -- | -- | -- | X21 | INT 11 |
| -- | -- | -- | -- | X22 | INT 12 |
| -- | -- | -- | -- | X23 | INT 13 |
| -- | -- | -- | -- | X24 | INT 14 |
| -- | -- | -- | -- | X25 | INT 15 |
| V736 sets interrupt time | INT 16 | -- | -- | X26(cannot be used along with s/w interrupt) | INT 16 |
| V737 sets interrupt time | INT 17 | -- | -- | X27 (cannot be used along with s/w interrupt) | INT 17 |

2nd Module (brace spanning X20–X27 / INT 10–17 rows)

Standard
RLL Instructions

**Interrupt Return (IRT)**

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |
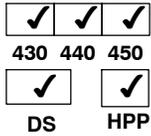
When an Interrupt Return is executed in the interrupt routine the CPU will return to the point in the main body of the program from which it was called. The Interrupt Return is programmed as the last instruction in an interrupt routine and is a stand alone instruction (no input contact on the rung).

—( IRT )
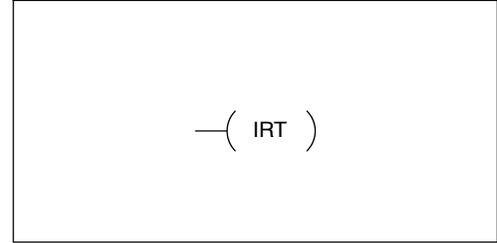
**Interrupt Return Conditional (IRTC)**

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |
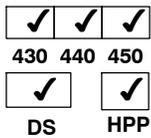
The Interrupt Return Conditional instruction is a optional instruction used with an input contact to implement a conditional return from the interrupt routine. The Interrupt Return is still required to terminate the interrupt routine

—( IRTC )
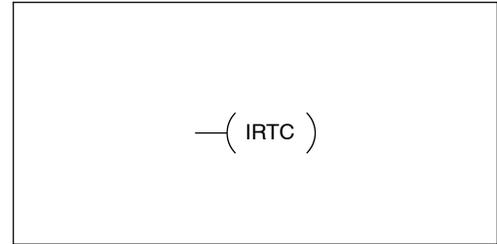
**Enable Interrupts (ENI)**
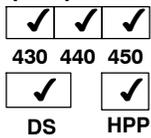
| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

The Enable Interrupt instruction is programmed in the main body of the application program (before the End instruction) to enable hardware or software interrupts. Once the coil has been energized interrupts will be enabled until the interrupt is disabled by the Disable Interrupt instruction.

—( ENI )
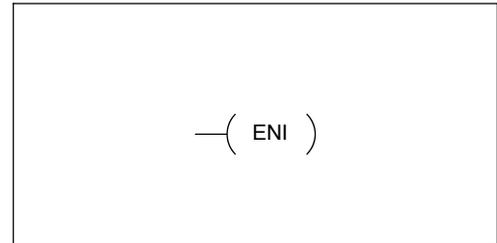
**Disable Interrupts (DISI)**
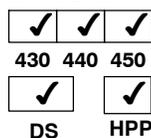
| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

The Disable Interrupt instruction is programmed in the main body of the application program (before the End instruction) to disable both hardware or software interrupts. Once the coil has been energized interrupts will be disabled until the interrupt is enabled by the Enable Interrupt instruction.

—( DISI )
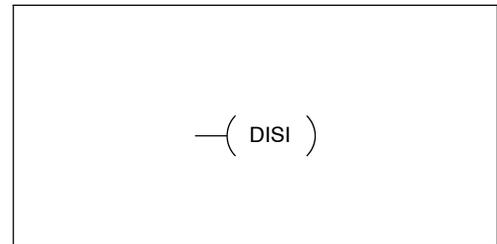
**Interrupt Example for Interrupt Module**

In the following example, when X40 is on, the interrupts will be enabled. When X40 is off the interrupts will be disabled. When a interrupt signal X1 is received the CPU will jump to the interrupt label INT O 1. The application ladder logic in the interrupt routine will be performed. If X35 is on the CPU will return to the main program with the IRTC instruction. If X35 is not on Y0-Y17 will be reset to off and then the CPU will return to the main body of the program.

*DirectSOFT*

Handheld Programmer Keystrokes

| STR | X(IN) | 4 | 0 | ↵ |
| SHFT | E | N | I | ↵ |
| STR | NOT | X(IN) | 1 | 7 | ↵ |
| SHFT | D | I | S | I | ↵ |

. .

| END | ↵ |
| SHFT | I | N | T | OCT | 1 | ↵ |
| STR | SHFT | I | SHFT | X(IN) | 2 | 0 | ↵ |
| OUT | SHFT | I | SHFT | Y(OUT) | 5 | ↵ |
| STR | SHFT | I | SHFT | X(IN) | 2 | 1 | ↵ |
| OUT | SHFT | I | SHFT | Y(OUT) | 1 | 0 | ↵ |
| STR | SHFT | I | SHFT | X(IN) | 3 | 5 | ↵ |
| SHFT | I | R | T | C | ↵ |
| STR | NOT | SHFT | I | SHFT | X(IN) | 3 | 5 | ↵ |
| RST | SHFT | I | SHFT | Y(OUT) | 0 | Y(OUT) | 1 | 7 | ↵ |
| SHFT | I | R | T | ↵ |
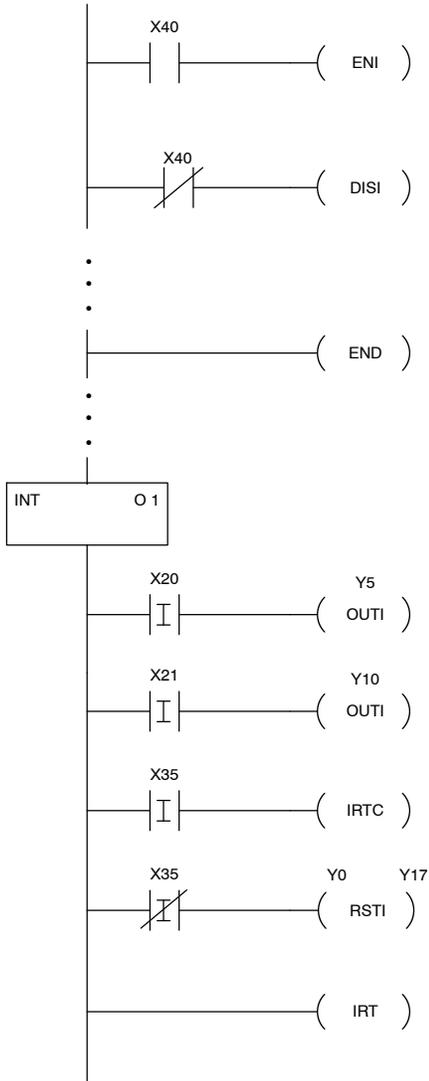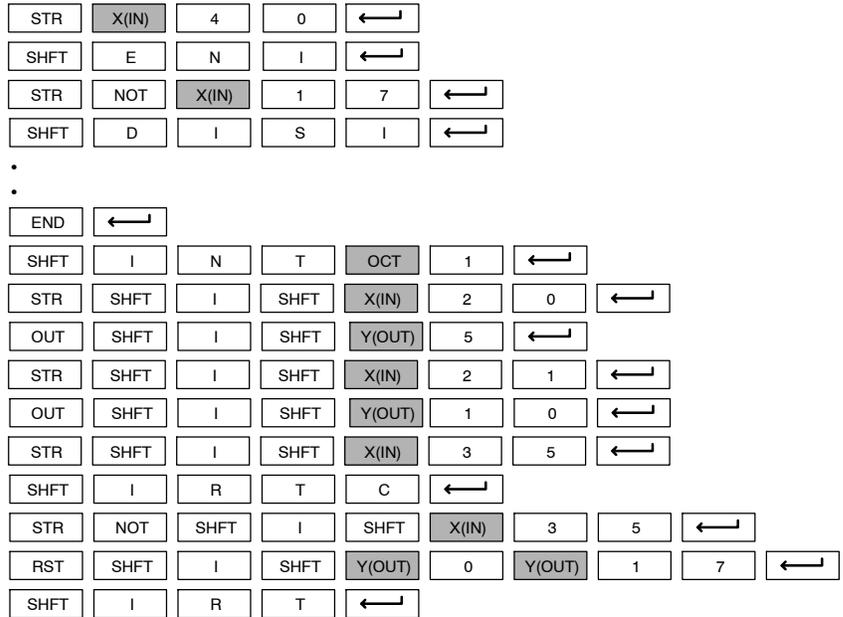
**Interrupt Example for Software Interrupt**

In the following example, when X1 is on, the value 10 is copied to V737. This value sets the software interrupt to 10 ms. When X20 turns on, the interrupt will be enabled. When X20 turns off, the interrupts will be disabled. Every 10 ms the CPU will jump to the interrupt label INT O 17. The application ladder logic in the interrupt routine will be performed. If X35 is on the CPU will return to the main program with the IRTC instruction. If X35 is not on Y0–Y17 will be reset to off and then the CPU will return to the main body of the program when the IRT instruction is executed. The software interrupt is limited to the range 3 – 999 milliseconds. Entering a 0, 1, or 2 will yield an interrupt time of 3 milliseconds.

*DirectSOFT*

Handheld Programmer Keystrokes

Load the constant value (K10) into the lower 16 bits of the accumulator

Copy the value in the lower 16 bits of the accumulator to V737

*Standard RLL Instructions*

# Intelligent I/O Instructions

**Read from Intelligent Module (RD)**

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |
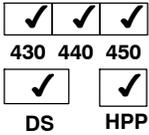
The Read from Intelligent Module instruction reads a block of data (1–128 bytes maximum) from an intelligent I/O module into the CPU's V-memory. It loads the function parameters into the first and second level of the accumulator stack, and the accumulator by three additional instructions.

```
┌──────────┐
│ RD       │
│    V aaa │
└──────────┘
```
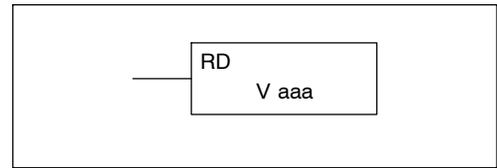
Listed below are the steps to program the Read from Intelligent module function.

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.
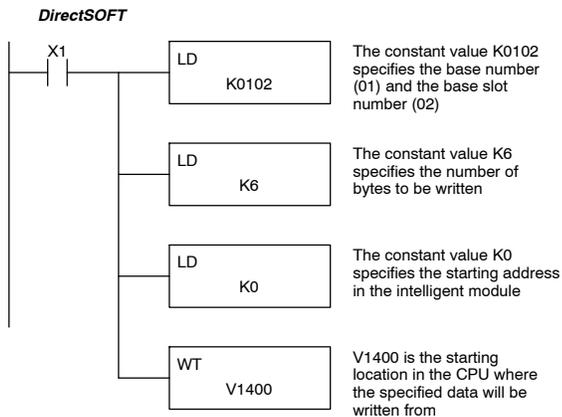
| Operand Data Type | | DL430 Range | DL440 Range | DL450 Range |
|---|---|---|---|---|
| | | **aaa** | **aaa** | **aaa** |
| V–memory | V | All (See p. 3–40) | All (See p. 3–41) | All (See p. 3–42) |

| 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.

*DirectSOFT*

| LD | The constant value K0102 specifies the base number (01) and the base slot number (02) |
| K0102 | |

| LD | The constant value K6 specifies the number of bytes to be read |
| K6 | |

| LD | The constant value K0 specifies the starting address in the intelligent module |
| K0 | |

| RD | V1400 is the starting location in the CPU where the specified data will be stored |
| V1400 | |

**CPU**

| | | | | |
|---|---|---|---|---|
| V1400 | 3 | 4 | 1 | 2 |
| V1401 | 7 | 8 | 5 | 6 |
| V1402 | 0 | 1 | 9 | 0 |
| V1403 | X | X | X | X |
| V1404 | X | X | X | X |

**Intelligent Module**

Data

| 12 | Address 0 |
| 34 | Address 1 |
| 56 | Address 2 |
| 78 | Address 3 |
| 90 | Address 4 |
| 01 | Address 5 |

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
| LD | K(CON) | 0 | 1 | 0 | 2 | ↵ |
| LD | K(CON) | 6 | ↵ |
| LD | K(CON) | 0 | ↵ | SHFT | R | D | SHFT |
| V | 1 | 4 | 0 | 0 | ↵ |

Standard RLL Instructions

**Write to Intelligent Module (WT)**

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | | ✓ |
|---|---|---|
| **DS** | | **HPP** |

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 necessary to program the Read from Intelligent module function.

```
        ┌──────────┐
────────┤ WT       │
        │   V aaa  │
        └──────────┘
```

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 | | DL430 Range | DL440 Range | DL450 Range |
|---|---|---|---|---|
| | | **aaa** | **aaa** | **aaa** |
| V-memory | V | All (See p. 3–40) | All (See p. 3–41) | All (See p. 3–42) |

| 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 information from Vmemory locations V1400–V1402.



*DirectSOFT*

| | |
|---|---|
| LD K0102 | The constant value K0102 specifies the base number (01) and the base slot number (02) |
| LD K6 | The constant value K6 specifies the number of bytes to be written |
| LD K0 | The constant value K0 specifies the starting address in the intelligent module |
| WT V1400 | V1400 is the starting location in the CPU where the specified data will be written from |

# Network Instructions

**Read from Network (RX)**

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | ✓ |
|---|---|
| **DS** | **HPP** |

The Read from Network instruction is used by the master device (CPU with a DCM) on a network to read a block of data from another 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 necessary to program the Read from Intelligent module function.

```
        ┌──────────────┐
────────┤ RX           │
        │   A aaa      │
        └──────────────┘
```
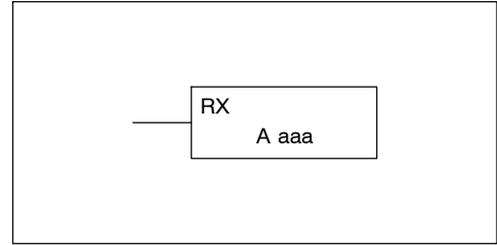
Step 1: — Load the slave address (1-90 BCD) into the first byte and the slot number of the master DCM (0-7) into the second byte of the second level of the accumulator stack. (Note: address 0 is only valid for peer stations.)

Step 2: — Load the number of bytes (2-128 BCD, multiple of 2) to be transferred into the first level of the accumulator stack.

Step 3: — Load the address where you want to store the data in the master station. The address must be specified in HEX.

Step 4: — Insert the RX instruction and specify the starting V-memory location (Aaaa) in the slave CPU where the data will be obtained.

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.

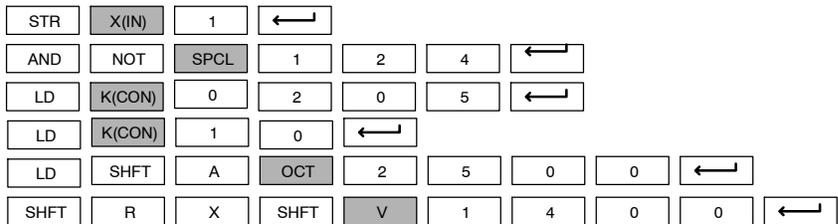| Operand Data Type | | DL430 Range | DL440 Range | DL450 Range |
|---|---|---|---|---|
| | **A** | **aaa** | **aaa** | **aaa** |
| V-memory | V | All (See p. 3-40) | All (See p. 3-41) | All (See p. 3-42) |
| Inputs | X | 0-477 | 0-477 | 0-1777 |
| Outputs | Y | 0-477 | 0-477 | 0-1777 |
| Control Relays | C | 0-737 | 0-1777 | 0-3777 |
| Stage | S | 0-577 | 0-1777 | 0-1777 |
| Timer | T | 0-177 | 0-377 | 0-377 |
| Counter | CT | 0-177 | 0-177 | 0-377 |
| Special Relay | SP | 0-137, 320-617 | 0-137 320-717 | 0-137 320-717 |
| Global I/O | GX | 0-777 | 0-1777 | 0-2777 |
| Program Memory | L$ | 0-3583 | 0-7679 (7.5K program mem.) 0-15871 (15.5K program mem.) | 0-7679 (7.5K program mem.) 0-15871 (15.5K program mem.) |
| Scratchpad | Z | 0-FFFF | 0-FFFF | 0-FFFF |

**NOTE:** If you are using a CoProcessor, Share Data Network, or DCM module, please refer to their manuals for information on how to transfer data over the *Direct*NET network.
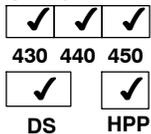
In the following example, when X1 is on and the module busy relay SP124 (see special relays) is not on, the RX instruction will access a DCM operating as a master in slot 2. Ten consecutive bytes of data (V1400 – V1404) will be read from a CPU at station address 5 and copied into V-memory locations V2500–V2504 in the CPU with the master DCM.

**DirectSOFT**

```
    X1      SP124           LD
  --| |------|/|-----------  K 0205
```

The constant value K0205 specifies the slot number (2) and the slave address (5)

```
                            LD
                            K 10
```

The constant value K10 specifies the number of bytes to be read

```
                            LDA
                            O 2500
```

Octal address 2500 is converted to 540 HEX and loaded into the accumulator. V2500 is the starting location for the Master CPU where the specified data will be stored

```
                            RX
                            V1400
```

V1400 is the starting location in the DL405 slave CPU where the specified data will be obtained.

**Master CPU** / **Slave CPU**

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| V2500 | 3 | 4 | 5 | 7 | ← | 3 | 4 | 5 | 7 | V1400 |
| V2501 | 8 | 5 | 3 | 4 | ← | 8 | 5 | 3 | 4 | V1401 |
| V2502 | 1 | 9 | 3 | 6 | ← | 1 | 9 | 3 | 6 | V1402 |
| V2503 | 9 | 5 | 7 | 1 | ← | 9 | 5 | 7 | 1 | V1403 |
| V2504 | 1 | 4 | 2 | 3 | ← | 1 | 4 | 2 | 3 | V1404 |
| V2505 | X | X | X | X | | X | X | X | X | V1405 |

Handheld Programmer Keystrokes

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| STR | X(IN) | 1 | ↵ | | | | | |
| AND | NOT | SPCL | 1 | 2 | 4 | ↵ | | |
| LD | K(CON) | 0 | 2 | 0 | 5 | ↵ | | |
| LD | K(CON) | 1 | 0 | ↵ | | | | |
| LD | SHFT | A | OCT | 2 | 5 | 0 | 0 | ↵ |
| SHFT | R | X | SHFT | V | 1 | 4 | 0 | 0 | ↵ |

## Write to Network (WX)

| ✓ | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | | ✓ |
|---|---|---|
| **DS** | | **HPP** |

The Write to Network instruction is used to write a block of data from the master device (CPU with a DCM) to 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 Write to Network function.
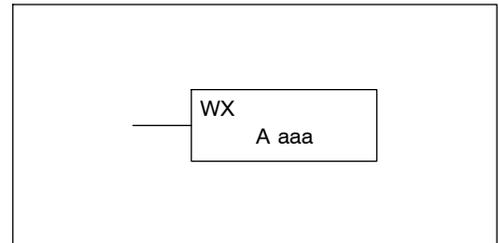
```
          WX
          A aaa
```

Step 1: — Load the slave address (1–90 BCD) into the first byte and the slot number of the master DCM (0–7) into the second byte of the second level of the accumulator stack. (Note: address 0 is only valid for peer stations.)

Step 2: — Load the number of bytes (2–128 BCD, multiple of 2) to be transferred into the first level of the accumulator stack.

Step 3: — Load the address where you want to obtain the data in the master station. The address must be specified in HEX.

Step 4: — Insert the WX instruction and specify the starting V–memory location (Aaaa) in the slave CPU where the data will be stored.
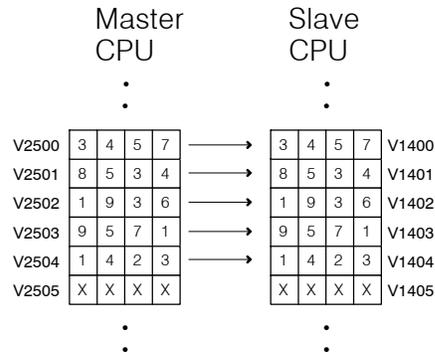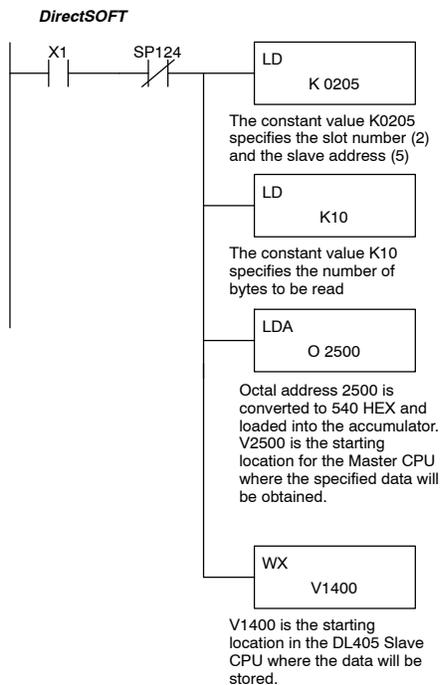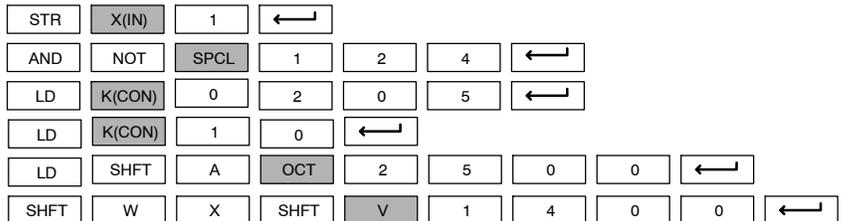
Helpful Hint: — For parameters that require HEX values, the LDA instruction can be used to convert an octal address to the HEX equivalent and load the value into the accumulator.

| Operand Data Type | | DL430 Range | DL440 Range | DL450 Range |
|---|---|---|---|---|
| | **A** | **aaa** | **aaa** | **aaa** |
| V-memory | V | All (See p. 3-40) | All (See p. 3-41) | All (See p. 3-42) |
| Inputs | X | 0-477 | 0-477 | 0-1777 |
| Outputs | Y | 0-477 | 0-477 | 0-1777 |
| Control Relays | C | 0-737 | 0-1777 | 0-3777 |
| Stage | S | 0-577 | 0-1777 | 0-1777 |
| Timer | T | 0-177 | 0-377 | 0-377 |
| Counter | CT | 0-177 | 0-177 | 0-377 |
| Special Relay | SP | 0-137, 320-617 | 0-137 320-717 | 0-137 320-717 |
| Global I/O | GX | 0-777 | 0-1777 | 0-2777 |
| Program Memory | $ | 0-3585 | 0-7679 (7.5K program mem.) 0-15873 (15.5K program mem.) | 0-7679 (7.5K program mem.) 0-15873 (15.5K program mem.) |
| Scratchpad | Z | 0-FFFF | 0-FFFF | 0-FFFF |

**NOTE:** If you are using a CoProcessor, Share Data Network, or DCM module, please refer to their manuals for information on how to transfer data over the DirectNET network.

In the following example when X1 is on and the module busy relay SP124 (see special relays) is not on, the WX instruction will access a DCM operating as a master in slot 2. 10 consecutive bytes of data is read from the CPU at station address 5 and copied to V-memory locations V1400-V1404 in the slave CPU.



*DirectSOFT*

LD
K 0205

The constant value K0205 specifies the slot number (2) and the slave address (5)

LD
K10

The constant value K10 specifies the number of bytes to be read

LDA
O 2500

Octal address 2500 is converted to 540 HEX and loaded into the accumulator. V2500 is the starting location for the Master CPU where the specified data will be obtained.

WX
V1400

V1400 is the starting location in the DL405 Slave CPU where the data will be stored.

Master CPU    Slave CPU

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| V2500 | 3 | 4 | 5 | 7 | → | 3 | 4 | 5 | 7 | V1400 |
| V2501 | 8 | 5 | 3 | 4 | → | 8 | 5 | 3 | 4 | V1401 |
| V2502 | 1 | 9 | 3 | 6 | → | 1 | 9 | 3 | 6 | V1402 |
| V2503 | 9 | 5 | 7 | 1 | → | 9 | 5 | 7 | 1 | V1403 |
| V2504 | 1 | 4 | 2 | 3 | → | 1 | 4 | 2 | 3 | V1404 |
| V2505 | X | X | X | X | | X | X | X | X | V1405 |

Handheld Programmer Keystrokes

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| STR | X(IN) | 1 | ← | | | | | |
| AND | NOT | SPCL | 1 | 2 | 4 | ← | | |
| LD | K(CON) | 0 | 2 | 0 | 5 | ← | | |
| LD | K(CON) | 1 | 0 | ← | | | | |
| LD | SHFT | A | OCT | 2 | 5 | 0 | 0 | ← |
| SHFT | W | X | SHFT | V | 1 | 4 | 0 | 0 | ← |

# Message Instructions

**System Errors and Fault Messages**

The DL405 CPUs provide error logging capabilities. There are certain predefined system error messages and codes, but you can also use the Fault instruction to create your own specific messages. The CPU logs the error, the date, and the time the error occurred. There are two separate tables that store this information.

- **System Error Table** – both the DL430 and DL440 have several predefined system error codes. The DL430 can hold one error at a time. The DL440 can show up to 32 errors in an error table. When an error occurs, the error is loaded into the first available location. *Therefore, the most recent error may not appear in the top row of the table.* If the table is full when an error occurs, the oldest error is pushed (erased) from the table and the new error is inserted in the row.

- **Fault Message Table** – the DL430 and DL440 also allow you to build your own error codes and messages. These are called Fault Messages. With the DL430, you can only build numeric error codes. With the DL440, you can build error codes, or up to 16 messages that can contain up to 23-character alphanumeric characters. In either case, you can have up to 16 messages or codes shown in the table. When a message is triggered, it is put in the first available table location. *Therefore, the most recent error may not appear in the top row of the table.* If the table is full when an error occurs, the oldest error is pushed (erased) from the table and the new error is inserted in the row.

The following diagram shows an example of a the Fault Message table as shown in *Direct*SOFT. You cannot view the entire table at one time with the handheld programmer. Instead, the messages automatically appear on the handheld programmer display as they occur. The message will remain on the display as long as the Fault instruction is being executed. You can also use an Auxiliary function (5C) to view the messages one at a time. (More on the handheld programmer display later.)

DL440 Error Msg. Example

Most recent message appears here, not at the top of the table.

Next message will show up in this row, which is now the oldest message.
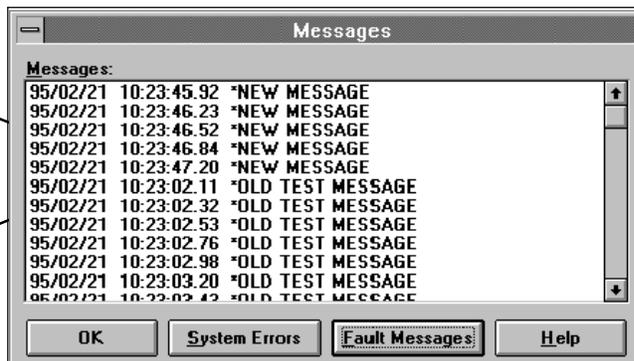


There are several instructions that can be used in combination to create these error codes and messages.

- FAULT – Fault
- DLBL – Data Label
- ACON – ASCII Constant
- NCON – Numeric Constant

The next few pages provide details on these instructions. Also, at the end of this section, there are two examples that show how the instructions are used together.

## Fault (FAULT)

| × | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | × |
|---|---|
| **DS** | **HPP** |

In a DL440 or DL450, the Fault instruction is used to display a message or numeric error code on the handheld programmer, *Direct*SOFT screen, or DV-1000 Operator Interface display. The message has a maximum of 23 characters and can be either V-memory data, numerical constant data or ASCII text.

```
      ┌──────────┐
──────┤ FAULT    │
      │   A aaa  │
      └──────────┘
```
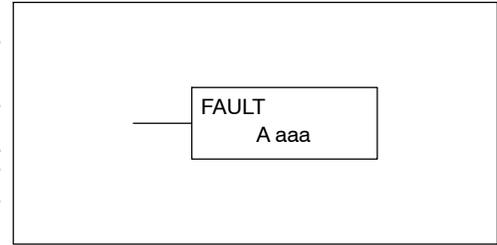
To display a value in a V-memory location, specify the V-memory location in the instruction.

To display ASCII or numeric data, you must use the DLBL (Data Label) instruction, and ACON (ASCII constant) or NCON (Numeric constant) instructions, in conjunction with the Fault instruction. In this case, you should specify the constant (K) value in the Fault instruction for the corresponding data label area that contains the ACON and/or NCON instructions.

| Operand Data Type | | DL440 Range | DL450 Range |
|---|---|---|---|
| | **A** | **aaa** | **aaa** |
| V-memory | V | All (See p. 3-41) | All (See p. 3-42) |
| Constant | K | 1-FFFF | 1-FFFF |

## Fault (FAULT)

| ✓ | × | × |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | × |
|---|---|
| **DS** | **HPP** |

In a DL430, the Fault instruction is used to display a numeric error code on the handheld programmer, *Direct*SOFT screen, or DV-1000 Operator Interface display.

```
      ┌──────────┐
──────┤ FAULT    │
      │   A aaa  │
      └──────────┘
```
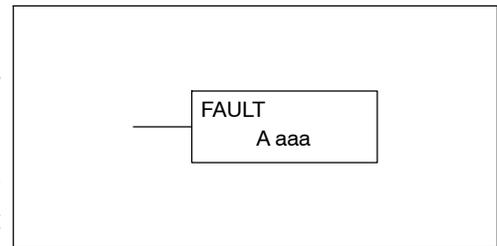
The error code may be obtained from a V-memory location, or may be constructed by specifying a constant in the Fault instruction.

To display the value in a V-memory location, specify the V-memory location in the instruction. To display a numeric constant, specify the constant (K) value in the instruction.

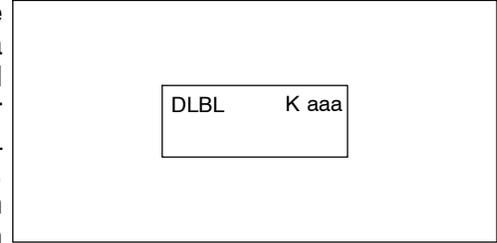| Operand Data Type | | DL430 Range |
|---|---|---|
| | **A** | **aaa** |
| V-memory | V | All (See p. 3-40) |
| Constant | K | 1-FFFF |

**NOTE:** The DL430 *does not* support the necessary instructions to build alphanumeric error messages. You can only build error codes by obtaining the code from a V-memory location, or by specifying a constant in the Fault instruction.

**Data Label
(DLBL)**

| × | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | × |
|---|---|
| **DS** | **HPP** |

The Data Label instruction marks the beginning of an ASCII/numeric data area and is typically used with ACON and NCON instructions. 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. Examples are shown later in this section.
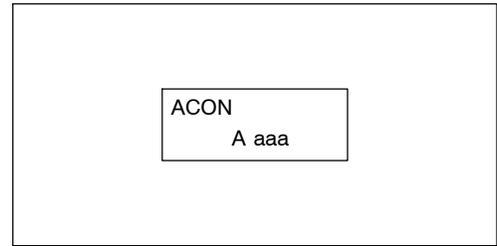
```
DLBL        K aaa
```

| Operand Data Type | | DL430 Range | DL440 Range | DL450 Range |
|---|---|---|---|---|
| | | aaa | aaa | aaa |
| Constant | K | -- | 1-FFFF | 1-FFFF |

**ASCII Constant
(ACON)**

| × | ✓ | ✓ |
|---|---|---|
| **430** | **440** | **450** |

| ✓ | × |
|---|---|
| **DS** | **HPP** |

The ASCII Constant instruction is used with the DLBL instruction to store ASCII text for use with other instructions. The instruction is utilized differently between the handheld programmer and our *Direct*SOFT programming software.
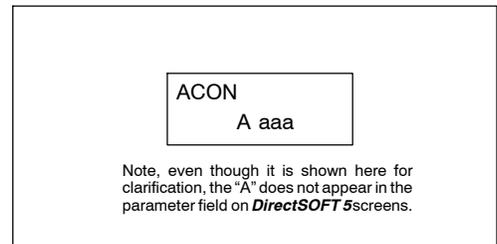
**Handheld Programmer:** With a handheld programmer, two ASCII characters can be stored in an ACON instruction. If only one character is stored in an ACON, a leading space will be printed in the Fault message.

```
ACON
    A aaa
```

| Operand Data Type | | DL430 Range | DL440 Range | DL450 Range |
|---|---|---|---|---|
| | | aaa | aaa | aaa |
| ASCII | A | -- | 0-9 A-Z | 0-9 A-Z |

***Direct*SOFT Programming Software:**
If you're using *Direct*SOFT, you can store up to 40 characters in an ACON. Also, you have a much wider range of characters that are supported. (See Appendix G for a complete listing of ASCII characters available.)

```
ACON
    A aaa
```

Note, even though it is shown here for clarification, the "A" does not appear in the parameter field on *DirectSOFT 5* screens.

| Operand Data Type | | DL430 Range | DL440 Range | DL450 Range |
|---|---|---|---|---|
| | | aaa | aaa | aaa |
| ASCII | A | -- | (See *DirectSOFT Manual*) | (See *DirectSOFT Manual*) |

At first glance you may wonder why the instructions work differently in the two different programming tools. In reality, the instructions *do not* work differently. In *Direct*SOFT, the 40-characters are actually broken down into multiple ACONs that contain 2 characters each when it is downloaded to the CPU. So, if you create the program with the software, and you examine the program mnemonics with a handheld (or even with *Direct*SOFT), you would see multiple ACONs that contain 2 characters each. Examples are shown on the following pages.

Standard
RLL Instructions

**Numeric Constant (NCON)**

| × | ✓ | ✓ |
|---|---|---|

**430 440 450**
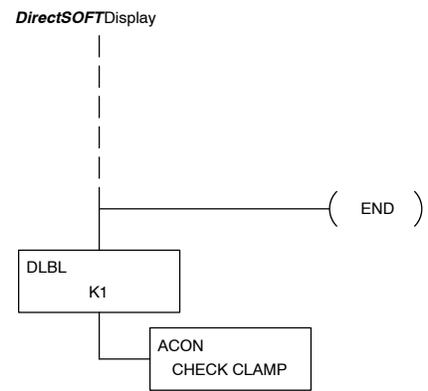
| ✓ | × |
|---|---|

**DS HPP**

The Numeric Constant instruction is used with the DLBL instruction to store the HEX ASCII equivalent of numeric data for use with other instructions. Two digits can be stored in an NCON instruction.

```
NCON
    K aaa
```
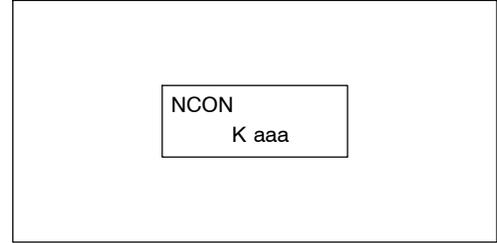
| Operand Data Type | | DL440 Range | DL450 Range |
|---|---|---|---|
| | | aaa | aaa |
| Constant | K | 0–FFFF | 0–FFFF |

**Using the Instructions to Build the Messages**

The Fault instruction actually copies the messages into the appropriate Error table. However, it is important to understand how the DLBL, ACON, and NCON instructions are combined to *build* the message. The DLBL is placed after the END instruction, which signifies the end of the main program. The ACON and NCON instructions are placed within the DLBL area. The diagram shows an example.

*DirectSOFT*Display

```
                              ( END )

DLBL
    K1

        ACON
            CHECK CLAMP
```

**History (HISTRY)**

| × | × | ✓ |
|---|---|---|

**430 440 450**

| ✓ | × |
|---|---|

**DS HPP**

The History instruction stores event history information in memory of the PLC.
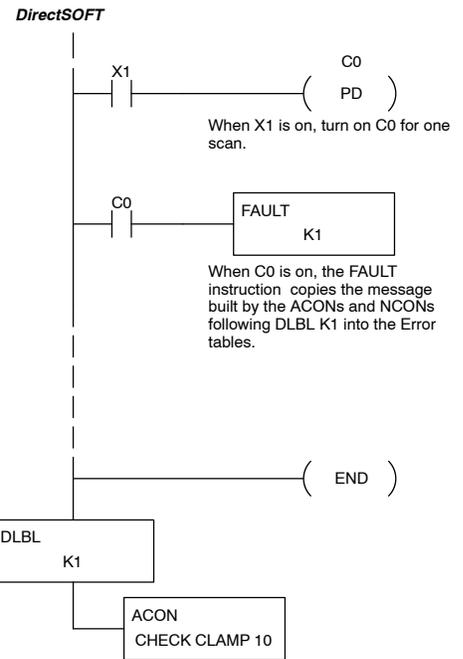
```
HISTRY
    A aaa
```
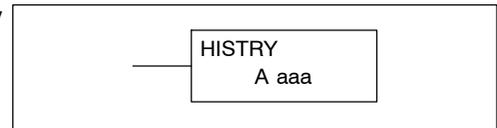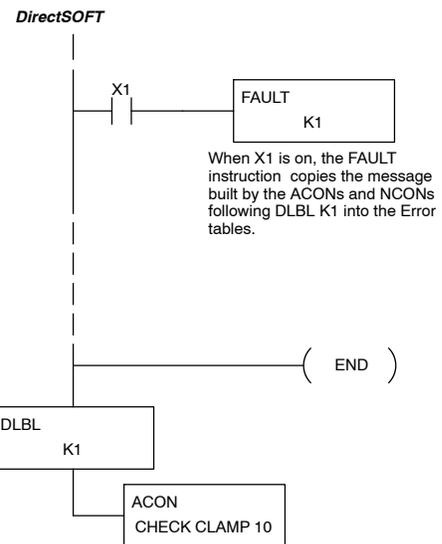
**Important Information about FAULT Execution**

It is important to understand how the Error Message and Error Code tables work when the Fault instruction is executed. Each time the instruction is executed, the code or message is inserted into the table. Since the CPU scan is extremely fast, then it is easily possible to completely fill up a table with a single message. In many cases this is not desirable, since it's nice to maintain a history of the errors. (That's why there are 32 positions in the Error Code table and 16 positions in the Error Message table.)

For example, let's say you are examining a limit switch (X1). When the switch is closed, you want an error message (CHECK CLAMP 10) to be logged into the Error Message table. In the real world, the limit switch may be closed for several seconds (or minutes, or hours...). During this time, the CPU will execute the Fault instruction a number of times, so the entire Error Message table will be full of a single message.

How do you solve this problem? Simple. Instead of using the limit switch to trigger the Fault instruction, use the limit switch to trigger a control relay used as a one-shot (PD coil instruction). Then, use the control relay as the input to the Fault instruction. Since the Fault is now triggered by the PD, which is only on for one scan, the message will only be copied into the table one time. This keeps the history of older messages intact.

*DirectSOFT*

X1 ———| |——————————————( C0 PD )

When X1 is on, turn on C0 for one scan.

C0 ———| |————— FAULT K1

When C0 is on, the FAULT instruction copies the message built by the ACONs and NCONs following DLBL K1 into the Error tables.

————————————( END )

DLBL K1

ACON CHECK CLAMP 10

**NOTE:** This method *does not* work correctly with a handheld programmer. The message will never appear on the display. You *must* use the input contact (X1 in this example) to trigger the Fault instruction directly. The diagram shown here provides an example.

*DirectSOFT*

X1 ———| |————— FAULT K1

When X1 is on, the FAULT instruction copies the message built by the ACONs and NCONs following DLBL K1 into the Error tables.

————————————( END )

DLBL K1

ACON CHECK CLAMP 10

Obviously, you could combine other instructions with the PD instruction usage to develop logic that would work for both the handheld programmer *and* **Direct**SOFT. For example, if it's possible in your application, you can have *two* input contacts that can trigger the FAULT instruction.
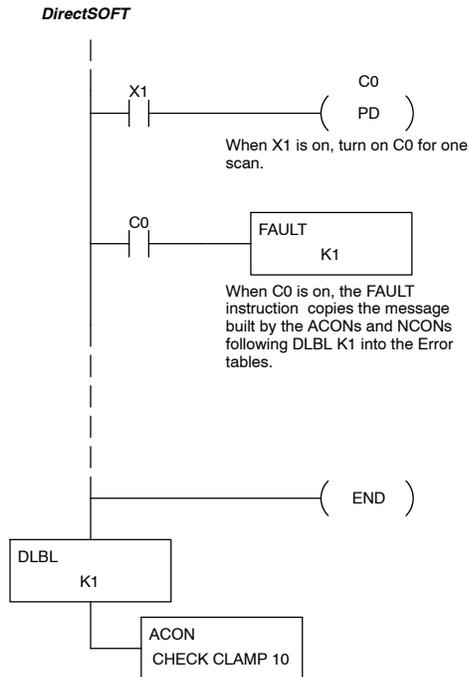
Consider the example shown to the right. If X1 comes on, then the message will only be copied into the table one time. You could see the message with **Direct**SOFT, but you would not automatically see it on the handheld programmer display.

If X2 comes on, then the message would probably fill the entire table. Also, it would automatically appear on the handheld programmer.
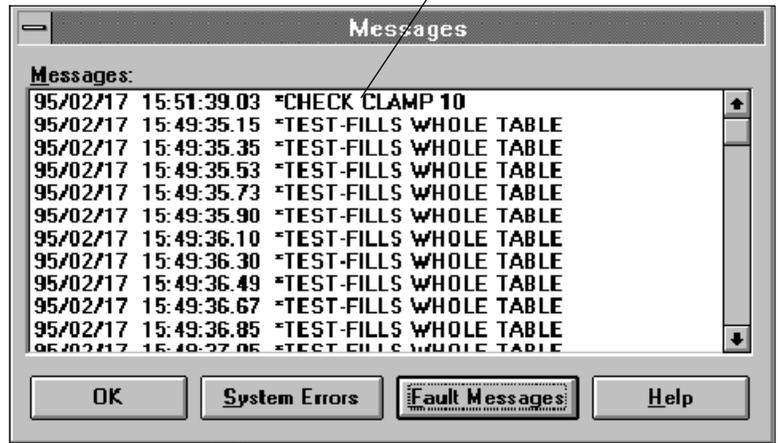
*DirectSOFT*Display

```
        X1                          C0
    ----| |--------------------(   PD   )
                        When X1 is on, turn on C0 for one
                        scan.

        C0              +-----------------+
    ----| |-------------|  FAULT          |
                        |          K1     |
        X2              +-----------------+
    ----| |--------------   When C0 is on, the FAULT
                        instruction  copies the message
                        built by the ACONs and NCONs
                        following DLBL K1 into the Error
                        tables.


    ------------------------------(  END  )

   +-----------------+
   | DLBL            |
   |        K1       |
   +-----------------+
            +-----------------+
            | ACON            |
            | CHECK CLAMP 10  |
            +-----------------+
```
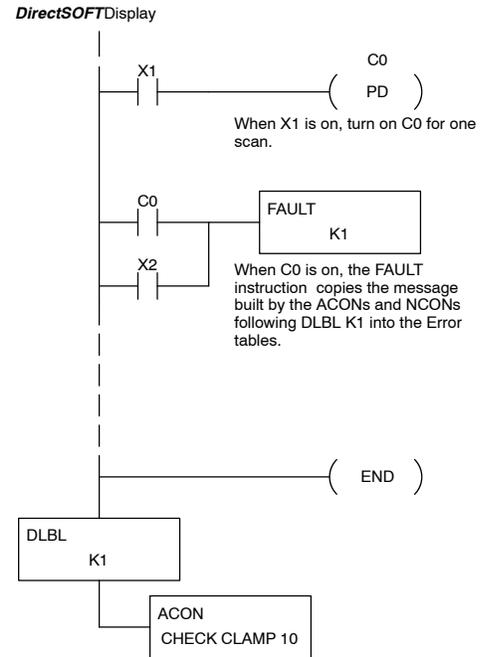
### *Direct*SOFT Example

Since you can enter more characters in an ACON instruction in **Direct**SOFT, it is very easy to build the entire message in one ACON instruction.
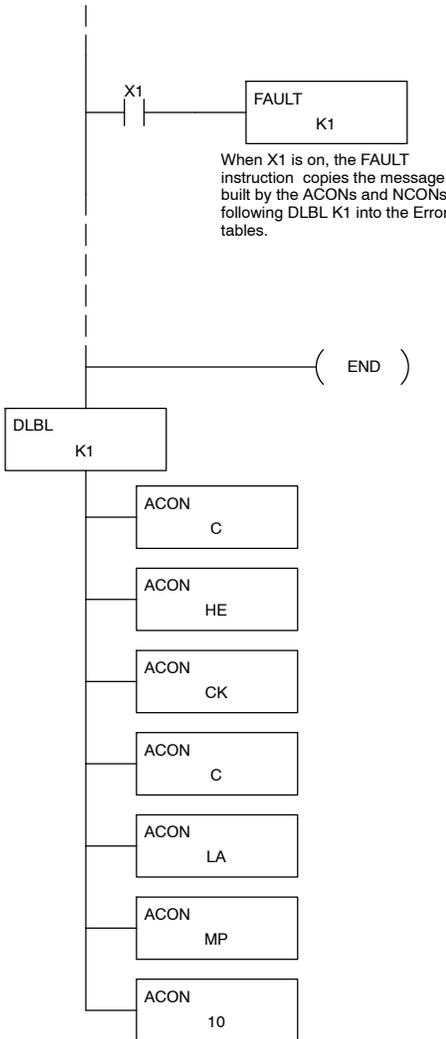
*DirectSOFT*

```
        X1                          C0
    ----| |--------------------(   PD   )
                        When X1 is on, turn on C0 for one
                        scan.

        C0              +-----------------+
    ----| |-------------|  FAULT          |
                        |          K1     |
                        +-----------------+
                        When C0 is on, the FAULT
                        instruction  copies the message
                        built by the ACONs and NCONs
                        following DLBL K1 into the Error
                        tables.


    ------------------------------(  END  )

   +-----------------+
   | DLBL            |
   |        K1       |
   +-----------------+
            +-----------------+
            | ACON            |
            | CHECK CLAMP 10  |
            +-----------------+
```

PD C0 triggers one entry into the table

```
+--------------------------------------------------+
| -              Messages                          |
+--------------------------------------------------+
| Messages:                                        |
| 95/02/17  15:51:39.03  *CHECK CLAMP 10        [↑]|
| 95/02/17  15:49:35.15  *TEST-FILLS WHOLE TABLE   |
| 95/02/17  15:49:35.35  *TEST-FILLS WHOLE TABLE   |
| 95/02/17  15:49:35.53  *TEST-FILLS WHOLE TABLE   |
| 95/02/17  15:49:35.73  *TEST-FILLS WHOLE TABLE   |
| 95/02/17  15:49:35.90  *TEST-FILLS WHOLE TABLE   |
| 95/02/17  15:49:36.10  *TEST-FILLS WHOLE TABLE   |
| 95/02/17  15:49:36.30  *TEST-FILLS WHOLE TABLE   |
| 95/02/17  15:49:36.49  *TEST-FILLS WHOLE TABLE   |
| 95/02/17  15:49:36.67  *TEST-FILLS WHOLE TABLE   |
| 95/02/17  15:49:36.85  *TEST-FILLS WHOLE TABLE   |
| 95/02/17  15:49:37.05  *TEST-FILLS WHOLE TABLE[↓]|
+--------------------------------------------------+
|  [ OK ]   [ System Errors ]  [Fault Messages] [ Help ] |
+--------------------------------------------------+
```
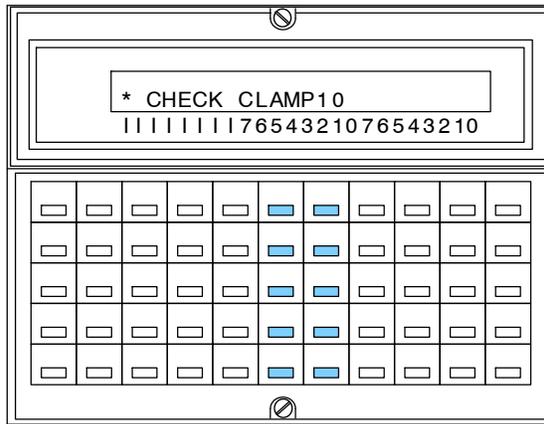
Standard
RLL Instructions

**Handheld Programmer Example**

Since you can only enter two characters per ACON with the handheld programmer, the program appears to be longer to get the same results. You'll also notice that we've organized the ACON contents slightly differently. For example, we've only included the character "C" in the first ACON. We have to do it this way in order to get the message to appear correctly. (Remember, a single character entered in an ACON with a handheld will have a blank space preceding it. So if your messages do not contain an even number of characters, you may have to play around with them to get the spacing just right.) Also, we have to use X1 to directly trigger the FAULT instruction. Otherwise, the message will never automatically appear on the handheld programmer display.

Message appears on handheld display



X1 — FAULT K1

When X1 is on, the FAULT instruction copies the message built by the ACONs and NCONs following DLBL K1 into the Error tables.

( END )

DLBL K1

ACON C

ACON HE

ACON CK

ACON C

ACON LA

ACON MP

ACON 10

```
*  CHECK  CLAMP10
I I I I I I I I 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0
```

Asterisk (*) is automatically provided by the handheld operating system. It does not have to be entered with an ACON.

Handheld Programmer Keystrokes

| STR | X(IN) | 1 | ↵ |
| SHFT | F | A | U | L | T | SHFT | K(CON) | 1 | ↵ |

.
.

| END | ↵ |
| SHFT | D | L | B | L | SHFT | K(CON) | 1 | ↵ |
| SHFT | A | C | O | N | ASC | SHFT | C | ↵ |
| SHFT | A | C | O | N | ASC | SHFT | H | E | ↵ |
| SHFT | A | C | O | N | ASC | SHFT | C | K | ↵ |
| SHFT | A | C | O | N | ASC | SHFT | C | ↵ |
| SHFT | A | C | O | N | ASC | SHFT | L | A | ↵ |
| SHFT | A | C | O | N | ASC | SHFT | M | P | ↵ |
| SHFT | A | C | O | N | ASC | 1 | 0 | ↵ |
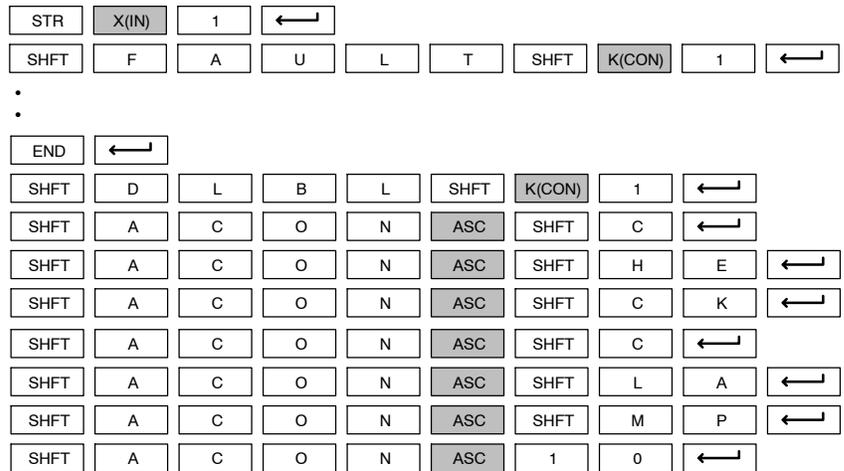
**Clearing the Messages**

You use different methods to clear the System Errors vs. the Fault Messages.
- System Errors — initialize the CPU's scratchpad memory
- Fault Message Table — clear the CPU program memory

**WARNING: If you initialize the scratchpad, you will also remove any retentive memory ranges that you may have changed.**

**Print Message (PRINT)**

| | | |
|---|---|---|
| ✗ | ✗ | ✔ |
| **430** | **440** | **450** |

| | |
|---|---|
| ✔ | ✗ |
| **DS** | **HPP** |
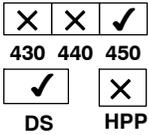
The Print Message instruction prints the embedded text or text/data variable message to the specified communications port (1, 2, or 3 on the DL450 CPU), which must have the communications port configured.

```
PRINT       A aaa
"Hello, this is a PLC message".
```
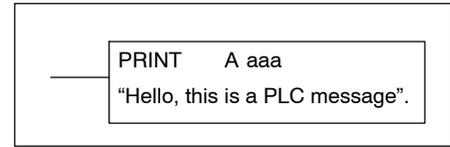
| Data Type | | DL450 Range |
|---|---|---|
| | A | aaa |
| Constant | K | 1, 2, or 3 |

You may recall from the CPU specifications in Chapter 3 that the DL450's ports are capable of several protocols. To configure a port 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 PLC on the menu bar, then **Setup > Setup Secondary Comm Port**, the dialog below will appear.

- **Port:** From the port number list box at the top, choose "Port 1".
- **Protocol:** Click the check box to the left of "Non-sequence".
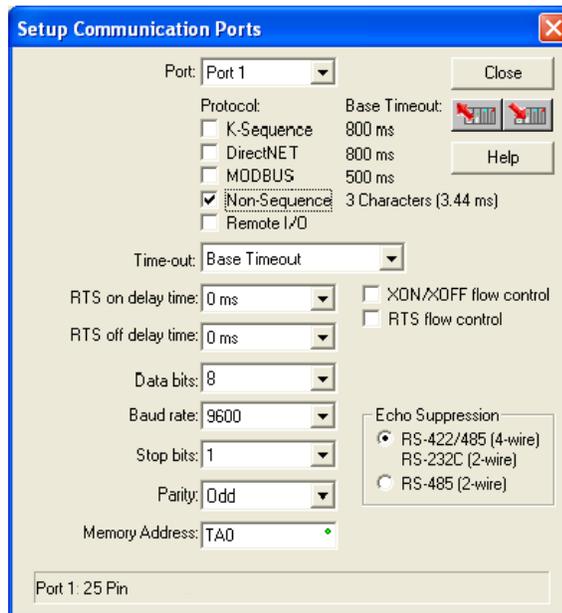


- **RTS Flow Control:** Choose the appropriate Request-to-Send option for your printer.
- **Data bits:** Choose the data bits
- **Baud Rate:** Choose the baud rate that matches your printer.
- **Stop Bits, Parity:** Choose number of stop bits and parity setting to match your printer.
- **Memory Address:** 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. Select "Use for printing only".

Click the button indicated to send the Port 1 configuration to the CPU, and click **Close**. See Chapter 3 for port wiring information, in order to connect your printer to the DL450.

Standard RLL Instructions

Ports 1 and 2 on the DL450 has standard RS232 levels, and should work with most printer serial input connections. Port 3 has RS422 levels, and will not work with most printers. You could use port 3 for long cables, converting the signals to RS232 at the other end to drive the printer.

**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:

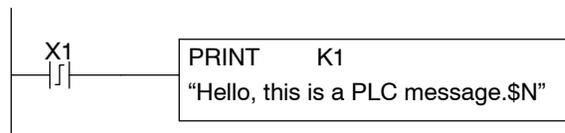| # | Character code | Description |
|---|---|---|
| 1 | $$ | Dollar sign ($) |
| 2 | $" | Double quotation (") |
| 3 | $L or $l | Line feed (LF) |
| 4 | $N or $n | Carriage return line feed (CRLF) |
| 5 | $P or $p | Form feed |
| 6 | $R or $r | Carriage return  (CR) |
| 7 | $T or $t | Tab |

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. Note that *Direct*SOFT does not give error indications for syntax errors in the PRINT instruction. Therefore, it is important to test your PRINT instruction data during the application development.

The following example prints the message to port 1. 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.

```
    X1              PRINT      K1
────┤↓├────        ┌─────────────────────────────────┐
                   │ "Hello, this is a PLC message.$N"│
                   └─────────────────────────────────┘
```

Print the message to Port 1 when X1 makes an off-to-on transition.
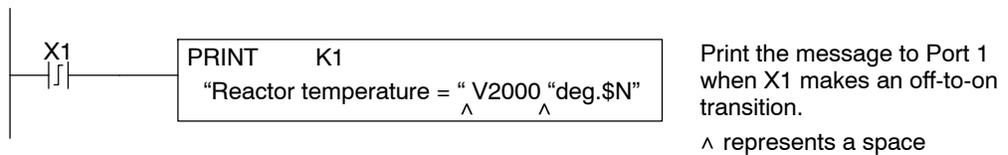
**V-memory element** – this is used for printing V-memory contents in the integer format or real format. Use V-memory number or V-memory number with ":" and data type. The data types are shown in the table below. The Character code must be in capital letters.

| # | 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. You must include spaces between the text string and the variable.

```
X1        PRINT     K1
─┤↑├─      "Reactor temperature = " V2000 "deg.$N"
                                  ^          ^
```

Print the message to Port 1 when X1 makes an off-to-on transition.

∧ represents a space

**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 C00 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 SP112 through SP117 indicate the status of the DL450 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.