Errata Sheet

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

Product Family:	DL305	**Date:**	07/28/2021
Manual Number	D3-350-M		
Revision and Date	2nd Edition, Rev. D; March 2010		

07.2021

Handheld Programmer D3-HP has been discontinued. Please consider Productivity, BRX, or CLICK series PLCs as an alternative platform.

Handheld Programmer D3-HPP was discontinued 01/2018.

08.2018

Changes to Chapter 5. Standard RLL Instructions; Timer, Counter and Shift Register; Accumulating Timer (TMRA) and Accumulating Fast Timer (TMRAF)

Page 5-38. In the first paragraph, second sentence on this page, the maximum value for the TMRAF instruction reads 99999.99 (total of seven 9’s). This is incorrect - it should be 999999.99 (total of eight 9’s) .

05.2018

Changes to Chapter 3. CPU Specifications and Operations

Page 3-6. Using Battery Backup; Enabling the Battery Backup

The ladder example shown to enable the backup battery is incorrect. Use the following example instead. Note: This example assumes that the original content of V7633 was a 0 (zero),

Changes to Chapter 5. Standard RLL Instructions; Accumulator Logical Instructions; Compare Real Number (CMPR)

Page 5-76. In the ladder example, the contact reference is incorrect. It should be SP62 turning on output C1, not SP60.

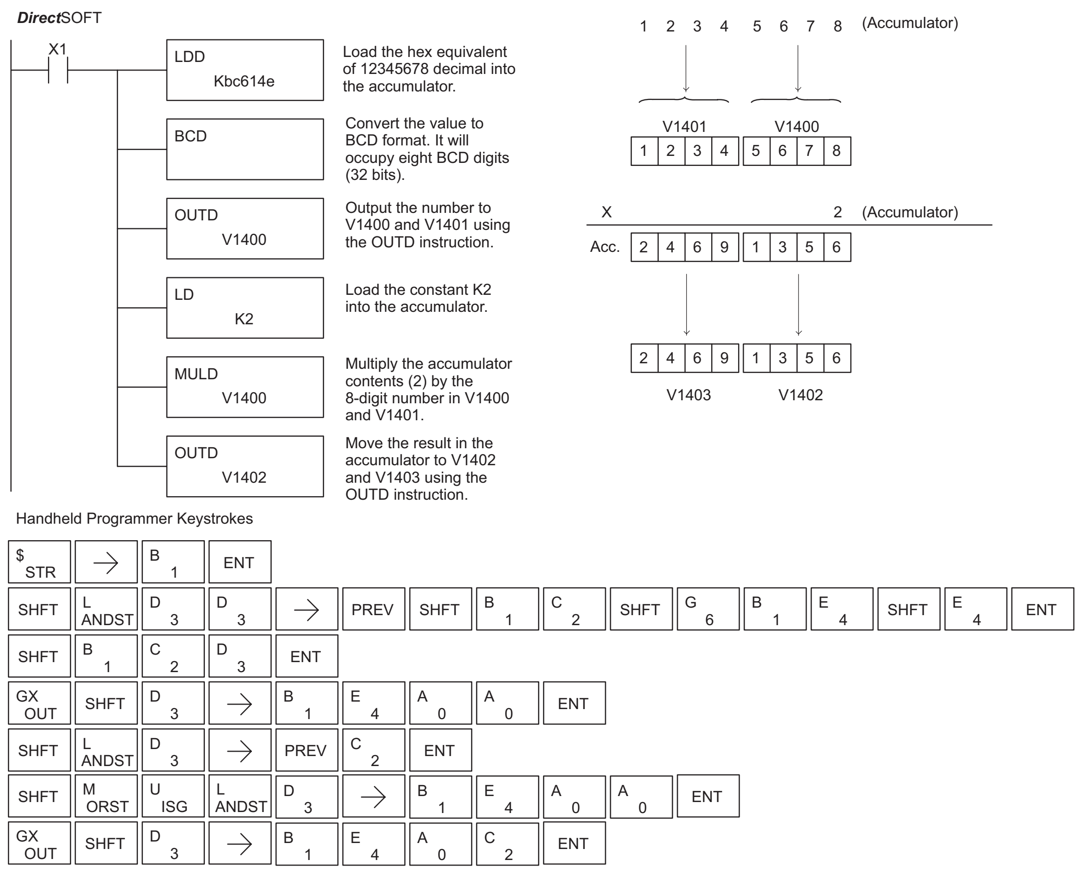
Errata Sheet

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

05.2018, cont'd

Changes to Chapter 5. Standard RLL Instructions; Math Instructions; Multiply Double (MULD)

Page 5-84. The ladder example shown is incorrect. Replace it with the following example.



Changes to Chapter 5. Standard RLL Instructions; Network Instructions

Please change Step 1 on pages 5-137 and 5-139 (RX & WX Commands) to read as follows:

Step 1: - Load the slave address (0-90 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. When using Port 2 of the CPU, the formatting should be Kf1xx where xx is the slave address (0-90 BCD).

Errata Sheet

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

05.2018, cont'd

Changes to Chapter 8. PID Loop Operation

For more recent and complete information on PID loop operation, refer to Chapter 8 in the DL06 user manual (p/n D0-06USER-M).

Changes to Chapter 9. Maintenance and Troubleshooting

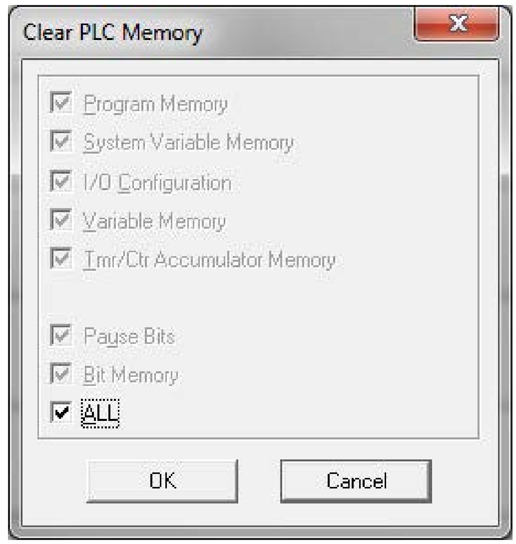
Page 9-25. Add the following to the end of this chapter (right after Regular Forcing with Direct Access):

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".



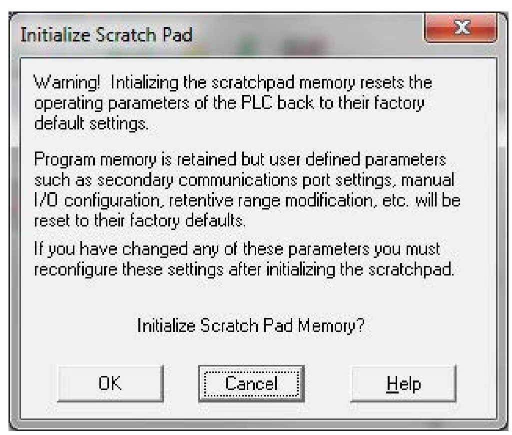
Step 2 – While connected with DirectSoft, go the PLC menu and then to the "Setup" submenu and select "Initialize Scratch Pad". Press "Ok".

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.

<u>**06.14.2012**</u>

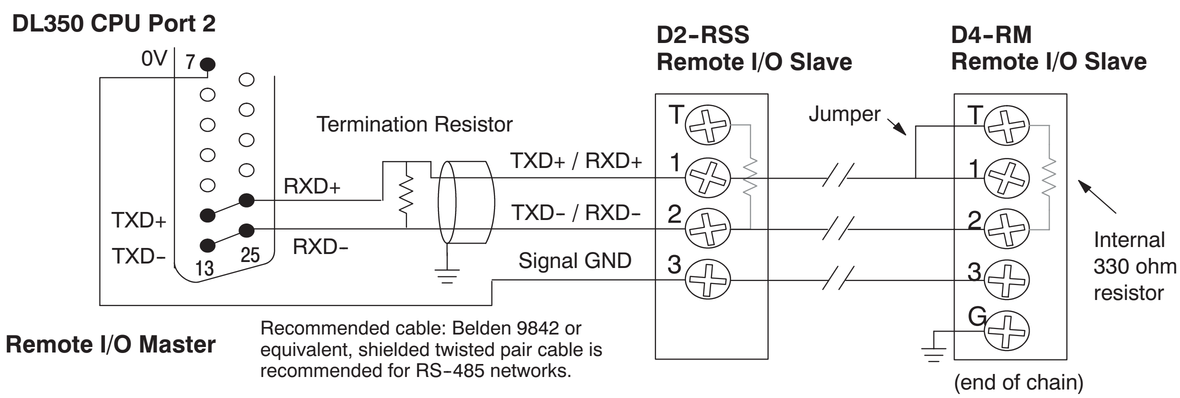
Changes to Chapter 4. System Design and Configuration

Page 4-15. I/O Configurations with a 10 Slot Local CPU Base

The four drawings on this page to the left of the bases showing jumper switch SW2 are mislabeled. They should say "700" and "100 EXP", not "700 EXP" and "100" as shown.

Page 4-18. CPU Specifications; Remote I/O Expansion; Configuring the CPU's Remote I/O Channel

Replace the remote I/O wiring diagram shown with this one.



Standard RLL Instructions

5

In This Chapter. . . .

Handheld Programmer D3-HP & D3-HPP have been retired as of 03/2021 & 01/2018 respectively. Please consider Productivity, BRX, or CLICK series PLC systems as upgrades.

Introduction

The DL350 CPU offers a wide variety of instructions to perform many different types of operations. This chapter shows you how to use these individual instructions. There are two ways to quickly find the instruction you need.

- If you know the instruction category (Boolean, Comparative Boolean, etc.) 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 name, use the following table to find the page that discusses the instruction.

Instruction		Page
ACON	ASCII Constant	5-143
ADD	Adds BCD	5-77
ADDB	Add Binary	5-90
ADDD	Add Double	5-78
ADDR	Add Real Number	5-79
AND	And for contacts or boxes	5-12, 5-29, 5-64
AND STR	And Store	5-14
ANDB	And Bit-of-Word	5-13
ANDD	And Double	5-65
ANDE	And if Equal	5-26
ANDF	And Formatted	5-66
ANDI	And Immediate	5-32
ANDN	And Not	5-12, 5-29
ANDNB	And Not Bit-of-Word	5-13
ANDNE	And if Not Equal	5-26
ANDNI	And Not Immediate	5-32
ANDND	And Negative Differential	5-21
ANDPD	And Positive Differential	5-21
ATH	ASCII to Hex	5-109
BCD	Binary Coded Decimal	5-104
BCDCPL	Tens Compliment	5-106
BIN	Binary	5-103
BCALL	Block Call (Stage)	7-27
BEND	Block End (Stage)	7-27
BLK	Block (Stage)	7-27
BTOR	Binary to Real	5-107
CMP	Compare	5-73
CMPD	Compare Double	5-74
CMPF	Compare Formatted	5-75
CMPR	Compare Real Number	5-76
CNT	Counter	5-40
CV	Converge Stage	7-25
CVJMP	Converge Jump (Stage)	7-25

Instruction		Page
DATE	Date	5-120
DEC	Decrement	5-89
DECB	Decrement Binary	5-95
DECO	Decode	5-102
DISI	Disable Interrupts	5-133
DIV	Divide	5-86
DIVB	Divide Binary	5-93
DIVD	Divide Double	5-87
DIVR	Divide Real Number	5-88
DLBL	Data Label	5-143
DRUM	Timed Drum	6-12
EDRUM	Event Drum	6-14
ENCO	Encode	5-101
END	End	5-122
ENI	Enable Interrupts	5-133
FAULT	Fault	5-141
FOR	For/Next	5-125
GOTO	Goto/Label	5-124
GRAY	Gray Code	5-113
GTS	Goto Subroutine	5-127
HTA	HEX to ASCII	5-110
INC	Increment	5-89
INCB	Increment Binary	5-94
INT	Interrupt	5-132
INV	Invert	5-105
IRT	Interrupt Return	5-133
IRTC	Interrupt Return Conditional	5-133
ISG	Initial Stage	7-24
JMP	Jump	7-24
LBL	Label (Goto/Lbl)	5-124
LD	Load	5-52
LDA	Load Address	5-55
LDD	Load Double	5-53
LDF	Load Formatted	5-54
LDR	Load Real number	5-58
LDX	Load Indexed	5-56
LDLBL	Load Label	5-117
LDSX	Load Indexed from Constant	5-57

Instruction		Page
MDRUMD	Masked Event Drum Discrete	6–18
MDRUMW	Masked Event Drum Word	6–20
MLR	Master Line Reset	5–130
MLS	Master Line Set	5–130
MOV	Move	5–116
MOVMC	Move Memory Cartridge	5–117
MUL	Multiply	5–83
MULB	Multiply Binary	5–92
MULD	Multiply Double	5–84
MULR	Multiply Real Number	5–85
NCON	Numeric Constant	5–143
NEXT	Next (For/Next)	5–125
NJMP	Not Jump (Stage)	7–24
NOP	No Operation	5–122
NOT	Not	5–17
OR	Or	5–10, 5–28, 5–67
OR OUT	Or Out	5–17
OR OUTI	Or Out Immediate	5–33
OR STR	Or Store	5–14
ORB	Or Bit-of-word	5–11
ORD	Or Double	5–68
ORE	Or if Equal	5–25
ORF	Or Formatted	5–69
ORI	Or Immediate	5–31
ORN	Or Not	5–10, 5–28
ORNB	Or Not Bit-of-Word	5–11
ORND	Or Negative Differential	5–20
ORNE	Or if Not Equal	5–25
ORNI	Or Not Immediate	5–31
ORPD	Or Positve Differential	5–20
OUT	Out	5–15, 5–59
OUTB	Out Bit-of-Word	5–16
OUTD	Out Double	5–60
OUTF	Out Formatted	5–61
OUTI	Out immediate	5–33
OUTX	Indexed	5–62
PD	Positve Differential	5–18
POP	Pop	5–63
PRINT	Print	5–145
RD	Read from Intelligent Module	5–135
ROTL	Rotate Left	5–99
ROTR	Rotate Right	5–100
RST	Reset	5–22
RSTB	Reset Bit-of-Word	5–23
RSTI	Reset Immediate	5–34
RSTWT	Reset Watch Dog Timer	5–123
RT	Subroutine return	5–127
RTC	Subroutine Return Conditional	5–127
RTOB	Real to Binary	5–108

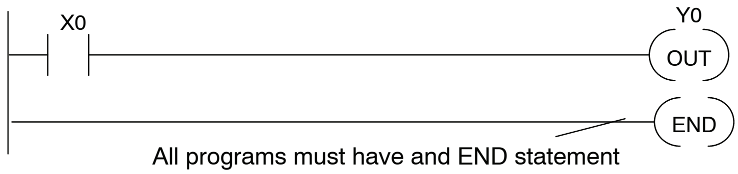
Instruction		Page
RX	Read From Network	5–137
SBR	Subroutine (Goto Subroutine)	5–127
SEG	Segment	5–112
SET	Set	5–22
SETB	Set Bit-of-Word	5–23
SETI	Set Immediate	5–34
SFLDGT	Shuffle Digits	5–114
SG	Stage	7–23
SGCNT	Stage Counter	5–42
SHFL	Shift Left	5–97
SHFR	Shift Right	5–98
SR	Shift Register	5–46
STOP	Stop	5–123
STR	Store	5–8, 5–27
STRB	Store Bit-of-word	5–9
STRE	Store if Equal	5–24
STRI	Store Immediate	5–30
STRN	Store Not	5–8, 5–27
STRNB	Store Not Bit-of-Word	5–9
STRNE	Store if not Equal	5–24
STRNI	Store Not Immediate	5–30
STRND	Store Negative Differential	5–19
STRPD	Store Positive Differential	5–19
SUB	Subtract	5–80
SUBB	Subtract Binary	5–91
SUBD	Subtract Double	5–81
SUBR	Subtract Real Number	5–82
SUM	Sum	5–96
TIME	Time of CPU	5–121
TMR	Timer	5–36
TMRA	Accumualting Timer	5–38
TMRAF	Accumualting Fast Timer	5–38
TMRF	Fast Timer	5–36
UDC	Up Down Couonter	5–44
WT	Write to Intelligent Module	5–136
WX	Write to Network	5–139
XOR	Exclusive Or	5–70
XORD	Exclusive Or Double	5–71
XORF	Exclusive Or Formatted	5–72

Using Boolean Instructions

Do you question why many PLC manufacturers quote the scan time for a 1K boolean program? It is because 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. Since the ***Direct***SOFT package allows the use of graphic symbols to build the program, you don't absolutely *have* to know the mnemonics of the instructions. However, it may helpful at some point, especially if you ever have to troubleshoot the program with a Handheld Programmer.

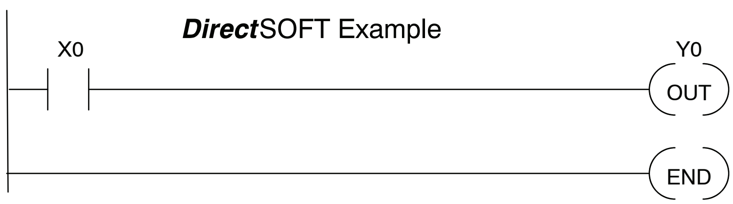
END Statement

All programs require an END statement as the last instruction. This tells the CPU it 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. The instruction set at the end of this chapter discussed this in detail.



Simple Rungs

You will use a contact to start rungs that contain both contacts and coils. The boolean instruction, Store or, STR instruction performs this function. 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.

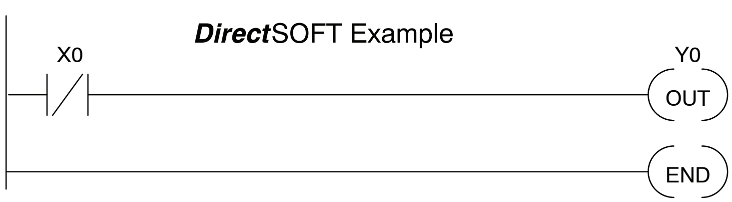


Handheld Mnemonics

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.

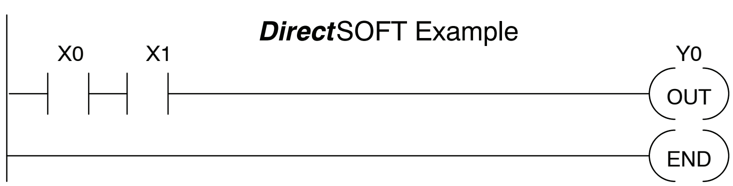


Handheld Mnemonics

STRN X0
OUT Y0
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 are STR X0, AND X1, followed by OUT Y0.

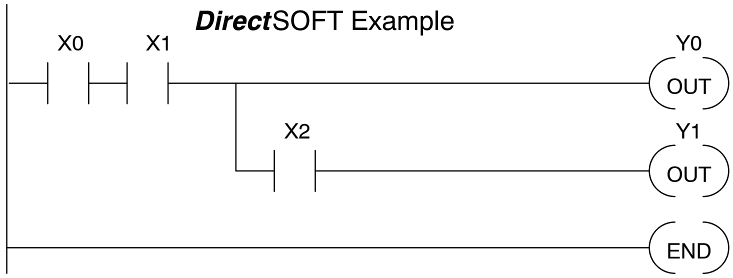


Handheld Mnemonics

STR X0
AND X1
OUT Y0
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.

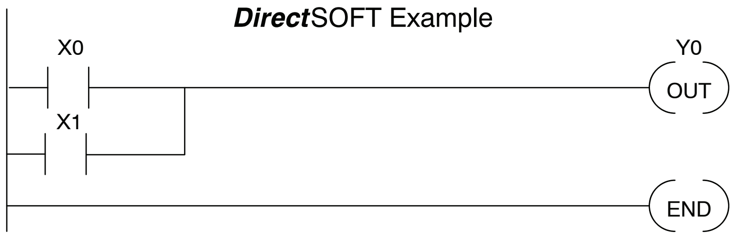


Handheld Mnemonics

```
STR X0
AND X1
OUT Y0
AND X2
OUT Y1
END
```

Parallel Elements

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

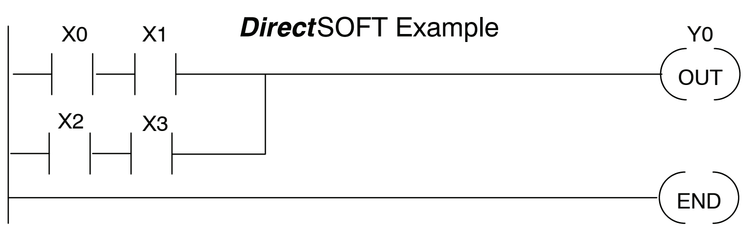


Handheld Mnemonics

```
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.

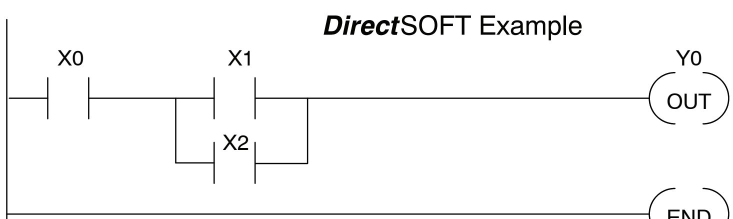


Handheld Mnemonics

```
STR X0
AND X1
STR X2
AND X3
ORSTR
OUT Y0
END
```

Joining Parallel Branches in Series

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

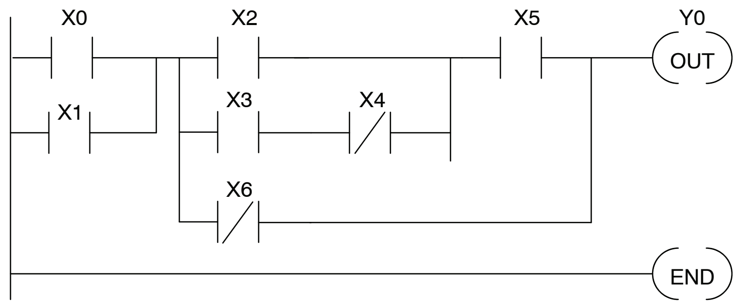


Handheld Mnemonics

```
STR X0
STR X1
OR X2
ANDSTR
OUT Y0
END
```

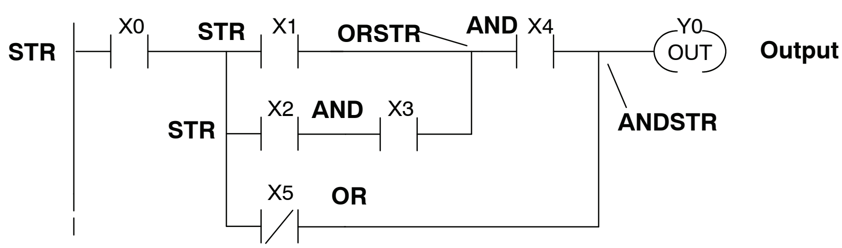
Combination Networks

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.



Boolean Stack

There are limits to how many elements you can include in a rung. This is because the DL350 CPU uses 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 stack.



STR X0

1	STR X0
2	
3	
4	
5	
6	
7	
8	

STR X1

1	STR X1
2	STR X0
3	
4	
5	
6	
7	
8	

STR X2

1	STR X2
2	STR X1
3	STR X0
4	
5	
6	
7	
8	

AND X3

1	X2 AND X3
2	STR X1
3	STR X0
4	
5	
6	
7	
8	

ORSTR

1	X1 OR (X2 AND X3)
2	STR X0
3	
•	
8	

AND X4

1	X4 AND [X1 OR (X2 AND X3)]
2	STR X0
3	
•	
8	

ORNOT X5

1	NOT X5 OR X4 AND [X1 OR (X2 AND X3)]
2	STR X0
3	
•	
8	

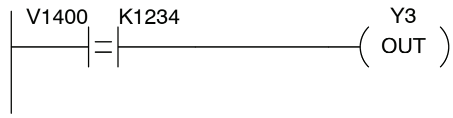
ANDSTR

1	X0 AND (NOT X5 OR X4) AND [X1 OR (X2 AND X3)]
2	
3	
:	
8	

Comparative Boolean

The DL350 CPU provides Comparative Boolean instructions that allow you to quickly and easily compare two numbers. The Comparative Boolean provides evaluation of two 4-digit values using boolean contacts. The valid evaluations are: equal to, not equal to, equal to or greater than, and less than.

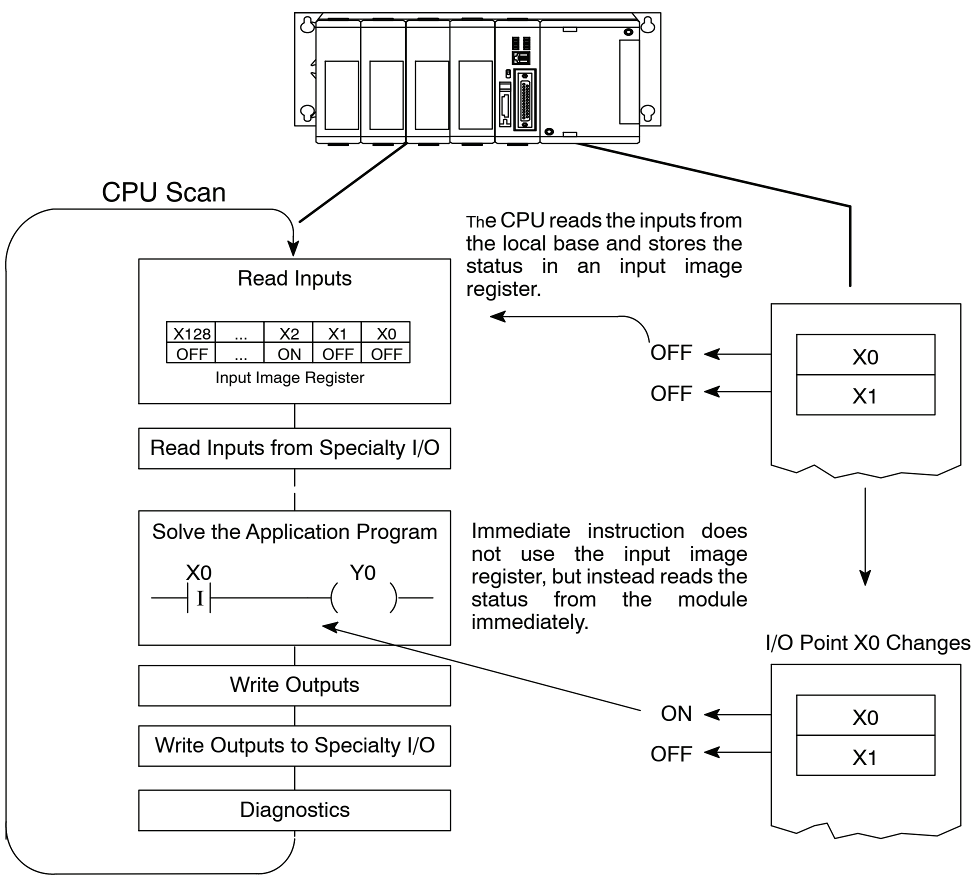
In the following example when the value in Vmemory location V1400 is equal to the constant 1234, Y3 will energize.



Immediate Boolean

The DL350 CPU usually can complete an operation cycle in milliseconds. However, in some applications you may not be able to wait a few milliseconds until the next I/O update occurs. The DL350 CPU offers 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. This is normally performed 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 performed until the read inputs or the write outputs portion of the CPU cycle.

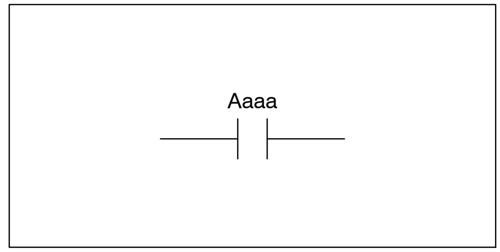
NOTE: Even though the immediate input instruction reads the most current status from the module, it only uses the results to solve that one instruction. It does not use the new status to update the image register. Therefore, any regular instructions that follow will still use the image register values.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

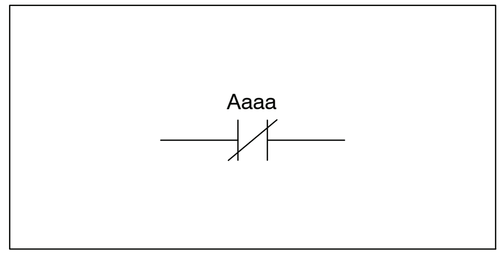
Store (STR)

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



Store Not (STRN)

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



Operand Data Type		DL350 Range
	A	aaa
Inputs	X	0–777
Outputs	Y	0–777
Control Relays	C	0–1777
Stage	S	0–1777
Timer	T	0–377
Counter	CT	0–177
Special Relay	SP	0–0777

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

*Direct*SOFT

Handheld Programmer Keystrokes

STR	→	1	ENT
OUT	→	2	ENT

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

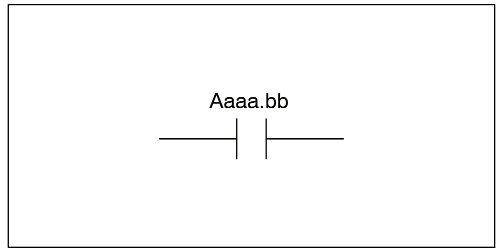
*Direct*SOFT

Handheld Programmer Keystrokes

STRN	→	1	ENT
OUT	→	2	ENT

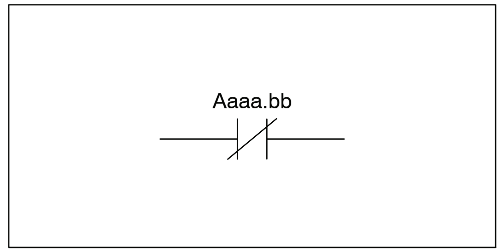
Store Bit-of-Word (STRB)

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



Store Not Bit-of-Word (STRNB)

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



Operand Data Type		DL350 Range	
	A	aaa	bb
V–memory	B	All (See p.3–29)	BCD, 0 to 15
Pointer	PB	All (See p 3–29)	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

Handheld Programmer Keystrokes

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

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

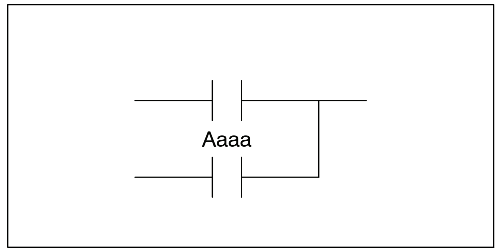
DirectSOFT

Handheld Programmer Keystrokes

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

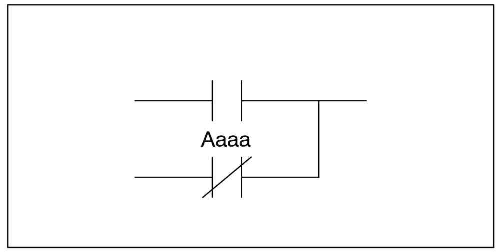
Or (OR)

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.



Or Not (ORN)

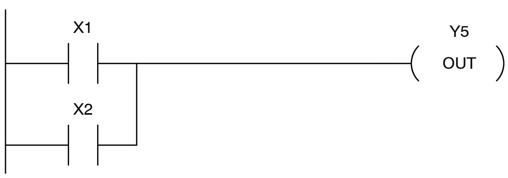
The Or Not instruction logically ors a normally closed contact in parallel with another contact in a rung. The status of the contact will be opposite the state of the associated image register point or memory location.



Operand Data Type		DL350 Range
	A	aaa
Inputs	X	0–777
Outputs	Y	0–777
Control Relays	C	0–1777
Stage	S	0–1777
Timer	T	0–377
Counter	CT	0–177
Special Relay	SP	0–777

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

*Direct*SOFT



Handheld Programmer Keystrokes

STR	→	1	ENT
OR	→	2	ENT
OUT	→	5	ENT

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

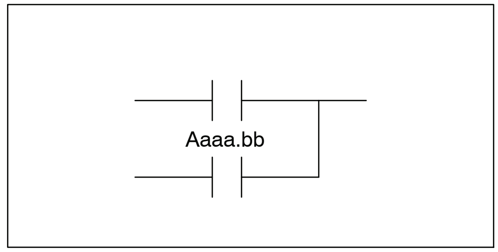
*Direct*SOFT

Handheld Programmer Keystrokes

STR	→	1	ENT
ORN	→	2	ENT
OUT	→	5	ENT

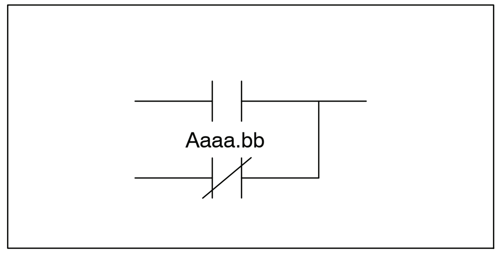
Or Bit-of-Word (ORB)

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.



Or Not Bit-of-Word (ORNB)

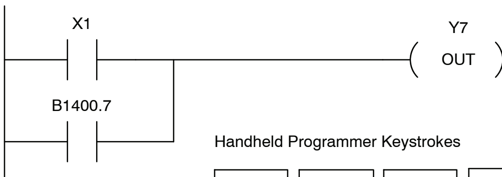
The Or Not Bit-of-Word instruction logically ors a normally closed contact in parallel with another contact in a rung. Status of the contact will be opposite the state of the bit referenced in the associated memory location.



Operand Data Type		DL350 Range	
	A	aaa	bb
V–memory	B	All (See p. 3–29)	BCD, 0 to 15
Pointer	PB	All (See p.3–29)	BCD

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

*Direct*SOFT

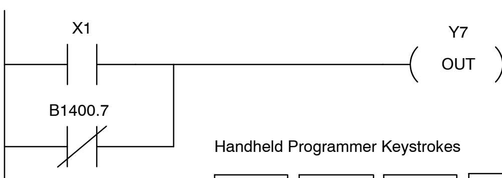


Handheld Programmer Keystrokes

STR	→	1	ENT					
OR	SHFT	B	→	V	1	4	0	0
→	K	7	ENT					
OUT	→	7	ENT					

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

*Direct*SOFT

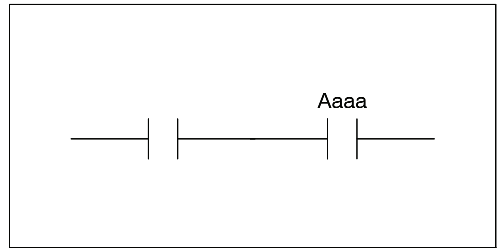


Handheld Programmer Keystrokes

STR	→	1	ENT					
ORN	SHFT	B	→	V	1	4	0	0
→	K	7	ENT					
OUT	→	5	ENT					

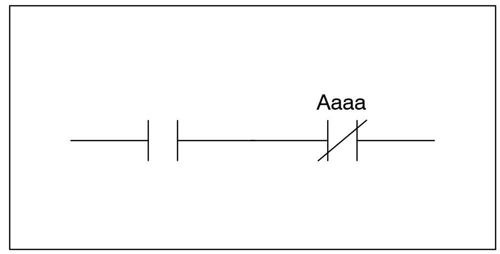
And (AND)

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



And Not (ANDN)

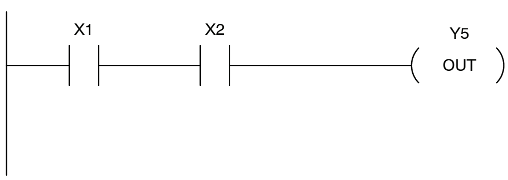
The And Not instruction logically ands a normally closed contact in series with another contact in a rung. The status of the contact will be opposite the state of the associated image register point or memory location.



Operand Data Type		DL350 Range
	A	aaa
Inputs	X	0–777
Outputs	Y	0–777
Control Relays	C	0–1777
Stage	S	0–1777
Timer	T	0–377
Counter	CT	0–177
Special Relay	SP	0–777

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

***Direct*SOFT**

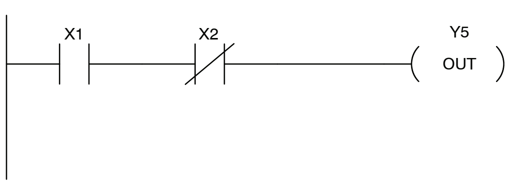


Handheld Programmer Keystrokes

STR	→	1	ENT
AND	→	2	ENT
OUT	→	5	ENT

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

***Direct*SOFT**

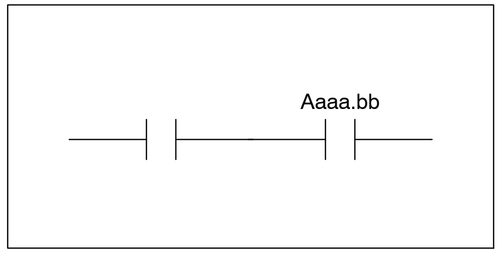


Handheld Programmer Keystrokes

STR	→	1	ENT
ANDN	→	2	ENT
OUT	→	5	ENT

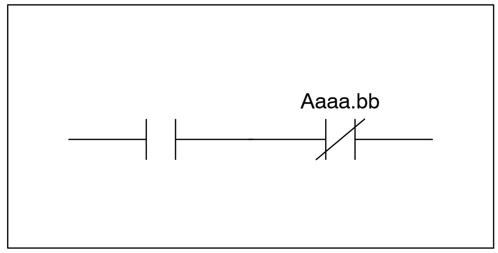
And Bit-of-Word (ANDB)

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



And Not Bit-of-Word (ANDNB)

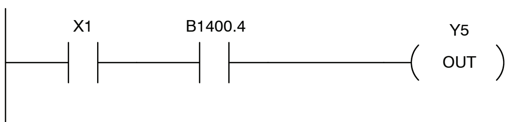
The And Not Bit-of-Word instruction logically ands a normally closed contact in series with another contact in a rung. The status of the contact will be opposite the state of the bit referenced in the associated memory location.



Operand Data Type		DL350 Range	
	A	aaa	bb
V–memory	B	All (See p. 3–29)	BCD, 0 to 15
Pointer	PB	All (See p. 3–29)	BCD

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

*Direct*SOFT

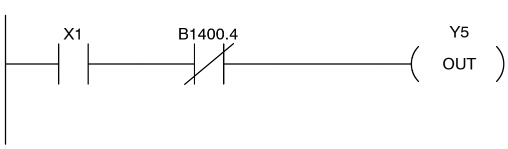


Handheld Programmer Keystrokes

STR	→	1	ENT					
AND	SHFT	B	→	V	1	4	0	0
→	K	4	ENT					
OUT	→	5	ENT					

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.

*Direct*SOFT

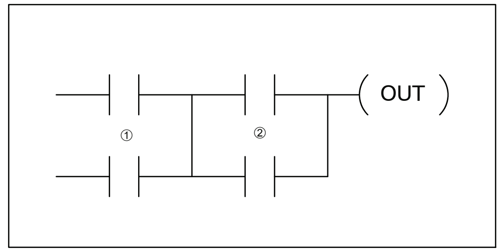


Handheld Programmer Keystrokes

STR	→	1	ENT					
ANDN	SHFT	B	→	V	1	4	0	0
→	K	4	ENT					
OUT	→	5	ENT					

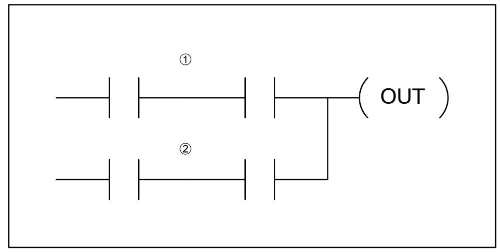
And Store (AND STR)

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



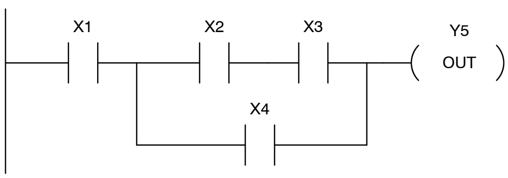
Or Store (OR STR)

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.

***Direct*SOFT**

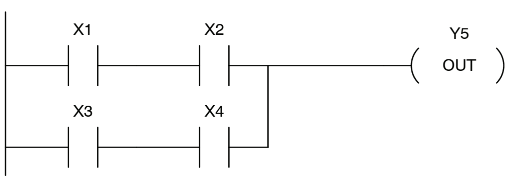


Handheld Programmer Keystrokes

STR	→	1	ENT
STR	→	2	ENT
AND	→	3	ENT
OR	→	4	ENT
ANDST	ENT		
OUT	→	5	ENT

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

***Direct*SOFT**

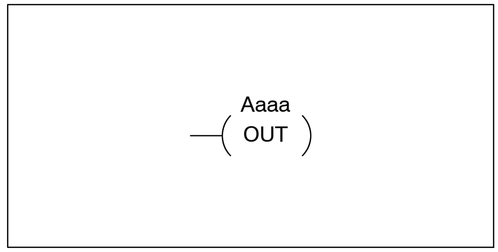


Handheld Programmer Keystrokes

STR	→	1	ENT
AND	→	2	ENT
STR	→	3	ENT
AND	→	4	ENT
ORST	ENT		
OUT	→	5	ENT

Out
(OUT)

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.



Operand Data Type		DL350 Range
	A	aaa
Inputs	X	0–777
Outputs	Y	0–777
Control Relays	C	0–1777

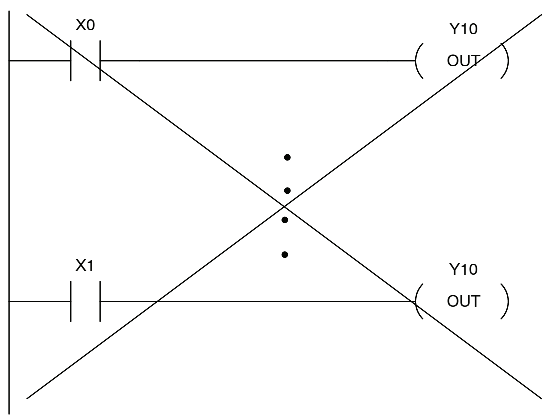
In the following Out example, when input X1 is on, output Y2 and Y5 will energize.

***Direct*SOFT**

Handheld Programmer Keystrokes

STR	→	1	ENT
OUT	→	2	ENT
OUT	→	5	ENT

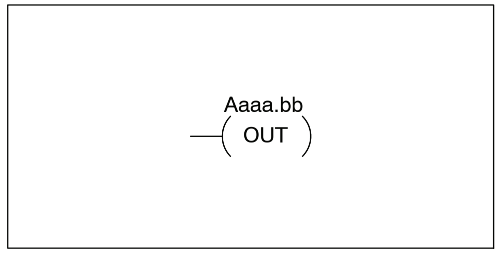
In the following Out example the program contains two Out instructions using the same location (Y10). The physical output of Y10 is ultimately controlled by the last rung of logic referencing Y10. X1 will override the Y10 output being controlled by X0. To avoid this situation, multiple outputs using the same location should not be used in programming. If you need to have an output controlled by multiple inputs see the OROUT instruction on page 5–17.



Standard RLL Instructions

Out Bit-of-Word (OUTB)

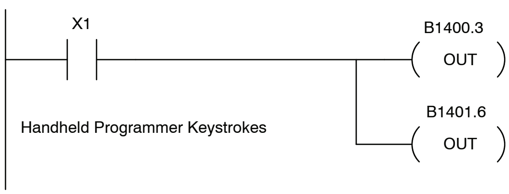
The Out Bit-of-Word instruction reflects the status of the rung (on/off) and outputs the discrete (on/off) state to the specified bit in the referenced memory location. Multiple Out Bit-of-Word instructions referencing the same bit of the same word generally should not be used since only the last Out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



Operand Data Type		DL350 Range	
	A	aaa	bb
V–memory	B	All (See p. 3–29)	BCD, 0 to 15
Pointer	PB	All (See p. 3–29)	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.

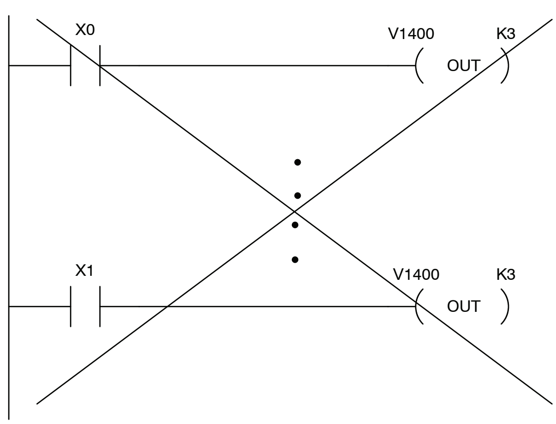
DirectSOFT



Handheld Programmer Keystrokes

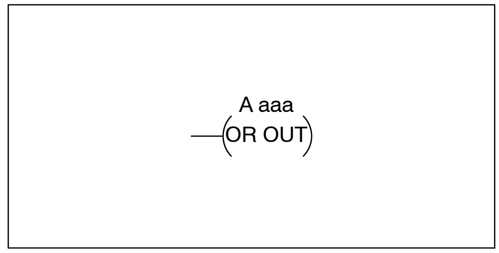
STR	→	1	ENT					
OUT	SHFT	B	→	V	1	4	0	0
→	K	3	ENT					
OUT	SHFT	B	→	V	1	4	0	0
→	K	4	ENT					

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.



Or Out (OR OUT)

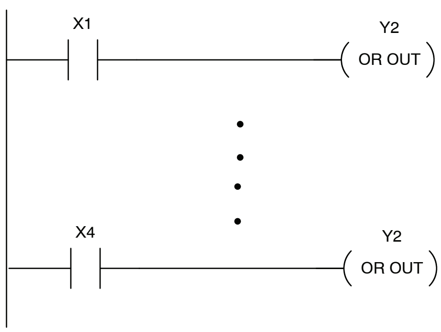
The Or Out instruction has been designed to used more than 1 rung of discrete logic to control a single output. Multiple Or Out instructions referencing the same output coil may be used, since *all* contacts controlling the output are ored together. If the status of *any* rung is on, the output will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



Operand Data Type		DL350 Range
	A	**aaa**
Inputs	X	0–777
Outputs	Y	0–777
Control Relays	C	0–1777

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

***Direct*SOFT**

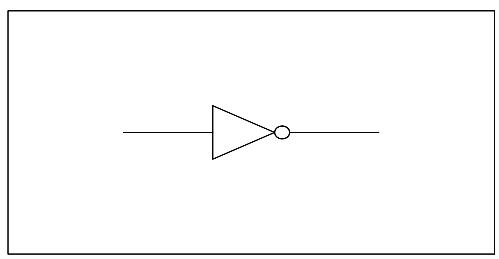


Handheld Programmer Keystrokes

STR	→	1	ENT				
INST#	3	5	ENT	ENT	→	2	ENT
STR	→	4	ENT				
INST#	3	5	ENT	ENT	→	2	ENT

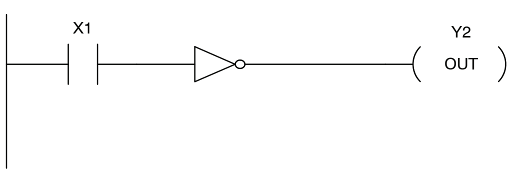
Not (NOT)

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.

***Direct*SOFT**



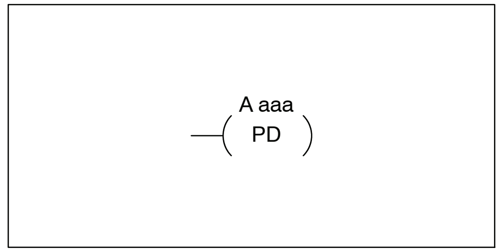
Handheld Programmer Keystrokes

STR	→	1	ENT	
SHFT	N	O	T	ENT
OUT	→	2	ENT	

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

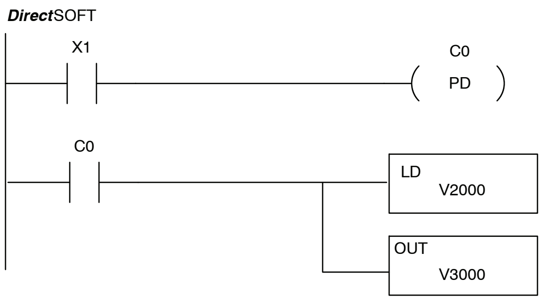
Positive Differential (PD)

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.



Operand Data Type		DL350 Range
	A	**aaa**
Inputs	X	0–777
Outputs	Y	0–777
Control Relays	C	0–1777

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

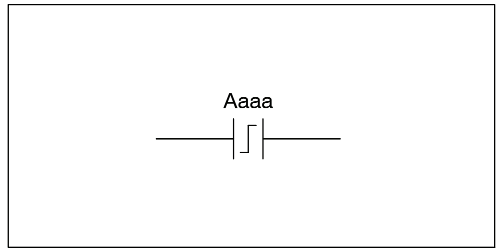


Handheld Programmer Keystrokes

STR	→	1	ENT			
SHFT	P	SHFT	D	→	0	ENT

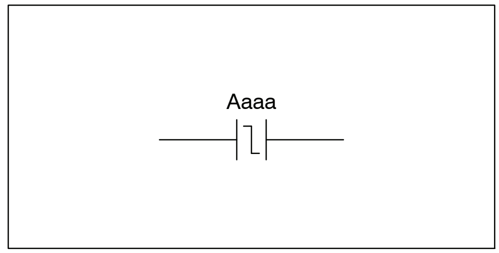
Store Positive Differential (STRPD)

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”.



Store Negative Differential (STRND)

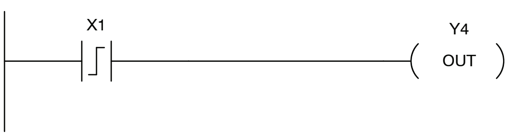
The Store Negative Differential instruction begins a new rung or an additional branch in a rung with a normally closed contact. The contact closes for one CPU scan when the state of the associated image register point makes an On-to-Off transition. Thereafter, the contact remains open until the next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



Operand Data Type		DL350 Range
	A	aaa
Inputs	X	0–777
Outputs	Y	0–777
Control Relays	C	0–1777
Stage	S	0–1777
Timer	T	0–377
Counter	CT	0–177

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

***Direct*SOFT**



Handheld Programmer Keystrokes

STR	SHFT	P	D	→
X	1	ENT		
OUT	Y	4	ENT	

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

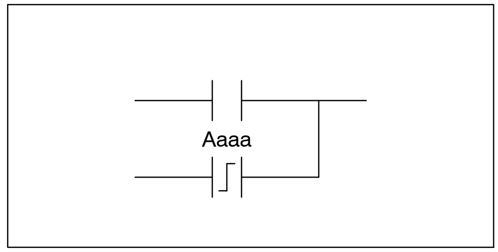
***Direct*SOFT**

Handheld Programmer Keystrokes

STR	SHFT	N	D	→
X	1	ENT		
OUT	Y	4	ENT	

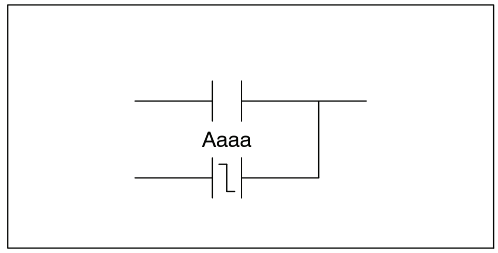
Or Positive Differential (ORPD)

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)

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		DL350 Range
	A	aaa
Inputs	X	0–777
Outputs	Y	0–777
Control Relays	C	0–1777
Stage	S	0–1777
Timer	T	0–377
Counter	CT	0–177

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

***Direct*SOFT**

Handheld Programmer Keystrokes

STR	X	1	ENT	
OR	SHFT	P	D	→
X	2	ENT		
OUT	Y	5	ENT	

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

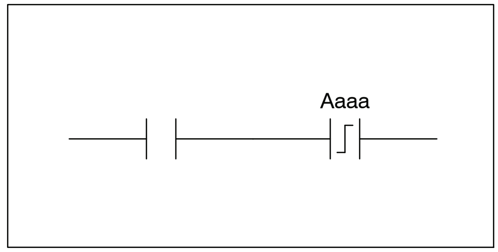
***Direct*SOFT**

Handheld Programmer Keystrokes

STR	X	1	ENT	
OR	SHFT	N	D	→
X	2	ENT		
OUT	Y	5	ENT	

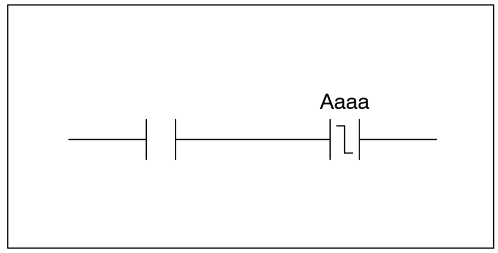
And Positive Differential (ANDPD)

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)

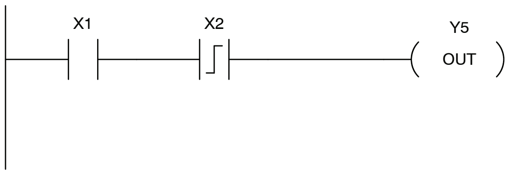
The And Negative Differential instruction logically ands a normally open contact in series with another contact in a rung. The status of the contact will be open until the associated image register point makes an On-to-Off transition, closing it for one CPU scan. Thereafter, it remains open until another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



Operand Data Type		DL350 Range
	A	aaa
Inputs	X	0–777
Outputs	Y	0–777
Control Relays	C	0–1777
Stage	S	0–1777
Timer	T	0–377
Counter	CT	0–177

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

***Direct*SOFT**

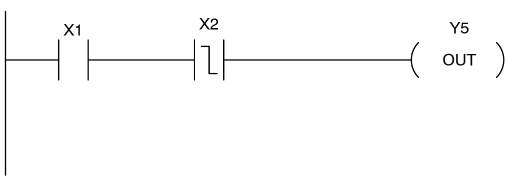


Handheld Programmer Keystrokes

STR	X	1	ENT	
AND	SHFT	P	D	→
X	2	ENT		
OUT	Y	5	ENT	

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

***Direct*SOFT**

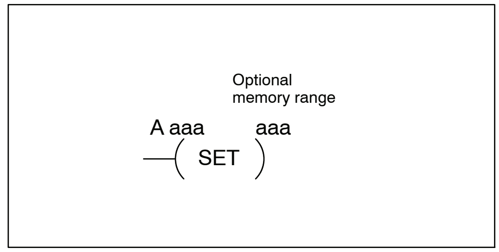


Handheld Programmer Keystrokes

STR	X(IN)	1	ENT	
AND	SHFT	N	D	→
X(IN)	2	ENT		
OUT	Y(OUT)	5	ENT	

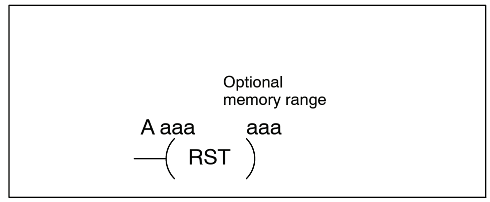
Set (SET)

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



Reset (RST)

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.



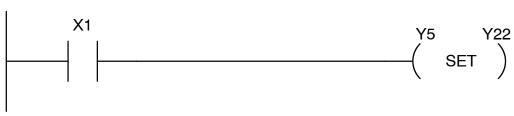
Operand Data Type		DL350 Range
	A	aaa
Inputs	X	0–777
Outputs	Y	0–777
Control Relays	C	0–1777
Stage	S	0–1777
Timer*	T	0–377
Counter*	CT	0–177

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

NOTE: You cannot set inputs (X's) that are assigned to input modules

In the following example when X1 is on, Y5 through Y22 will energize.

***Direct*SOFT**



Handheld Programmer Keystrokes

STR	→	1	ENT			
SET	→	5	→	2	2	ENT

In the following example when X1 is on, Y5 through Y22 will be reset or de-energized.

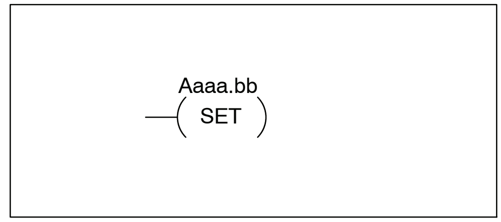
***Direct*SOFT**

Handheld Programmer Keystrokes

STR	→	1	ENT			
RST	→	5	→	2	2	ENT

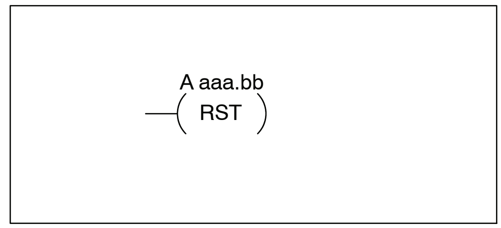
Set Bit-of-Word (SETB)

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



Reset Bit-of-Word (RSTB)

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



Operand Data Type		DL350 Range	
	A	aaa	bb
V–memory	B	All (See p. 3–29)	0 to 15
Pointer	PB	All (See p. 3–29)	0 to 15

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

***Direct*SOFT**

Handheld Programmer Keystrokes

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

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

***Direct*SOFT**

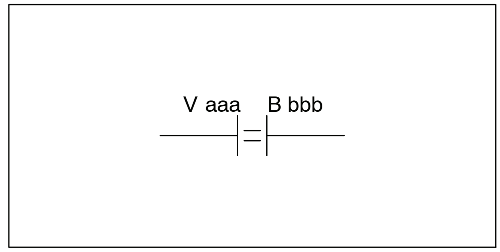
Handheld Programmer Keystrokes

STR	→	1	ENT					
RST	SHFT	B	→	V	1	4	0	0
→	K	1	5	ENT				

Comparative Boolean

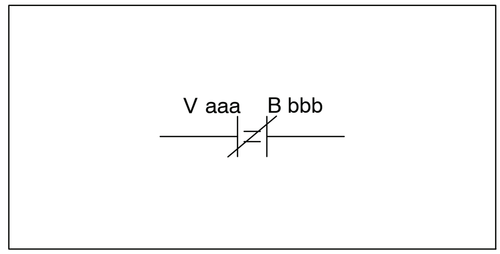
Store If Equal (STRE)

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



Store If Not Equal (STRNE)

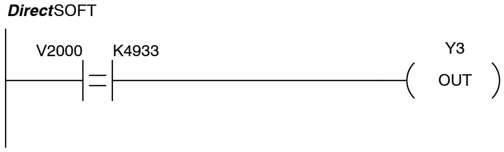
The Store If Not Equal instruction begins a new rung or additional branch in a rung with a normally closed comparative contact. The contact will be on when Vaaa $\neq$ Bbbb.



Operand Data Type		DL350 Range	
	B	aaa	bbb
V-memory	V	All (See page 3-29)	All (See page 3-29)
Pointer	P	--	All V mem. (See page 3-29)
Constant	K	--	0-FFFF

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

DirectSOFT



Handheld Programmer Keystrokes

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

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

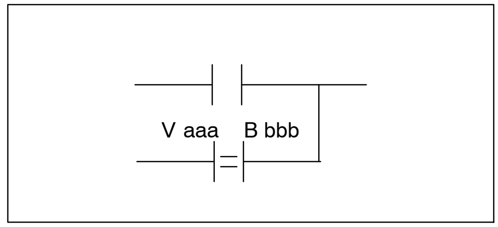
DirectSOFT

Handheld Programmer Keystrokes

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

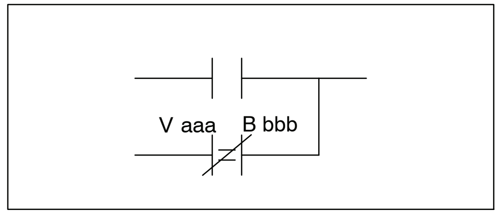
Or If Equal (ORE)

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



Or If Not Equal (ORNE)

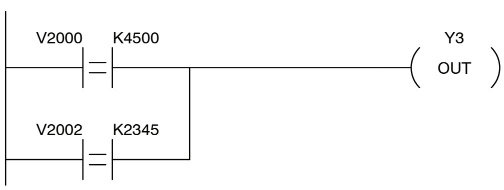
The Or If Not Equal instruction connects a normally closed comparative contact in parallel with another contact. The contact will be on when Vaaa ≠ Bbbb.



Operand Data Type		DL350 Range	
	B	aaa	bbb
V–memory	V	All (See page 3–29)	All (See page 3–29)
Pointer	P	--	All V mem. (See page 3–29)
Constant	K	--	0–FFFF

In the following example, when the value in V–memory location V2000 = 4500 or V2002 = 2345 , Y3 will energize.

*Direct*SOFT

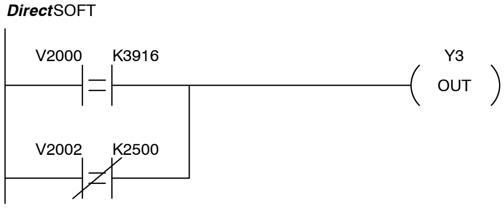


Handheld Programmer Keystrokes

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

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

*Direct*SOFT

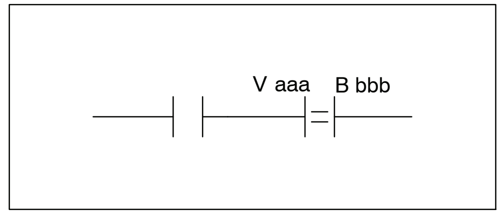


Handheld Programmer Keystrokes

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

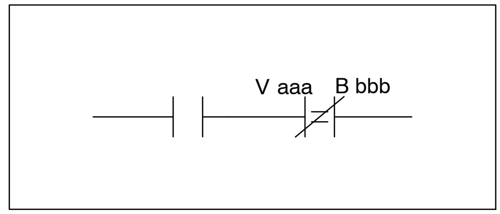
And If Equal (ANDE)

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



And If Not Equal (ANDNE)

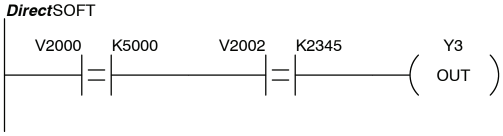
The And If Not Equal instruction connects a normally closed comparative contact in series with another contact. The contact will be on when Vaaa ≠ Bbbb



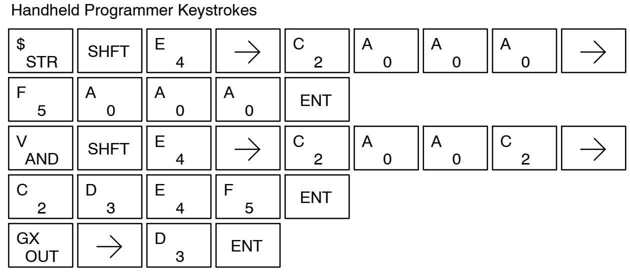
Operand Data Type		DL350 Range	
	A/B	aaa	bbb
V–memory	V	All (See page 3–29)	All (See page 3–29)
Pointer	P	--	All V mem. (See page 3–29)
Constant	K	--	0–FFFF

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

*Direct*SOFT

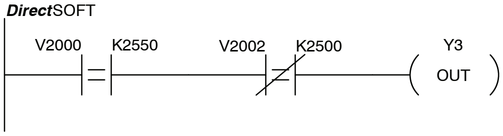


Handheld Programmer Keystrokes

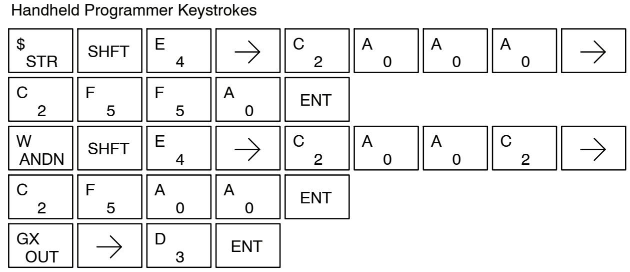


In the following example, when the value in V–memory location V2000 = 2550 and V2002 ≠ 2500, Y3 will energize.

*Direct*SOFT

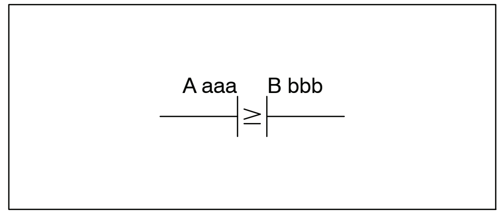


Handheld Programmer Keystrokes



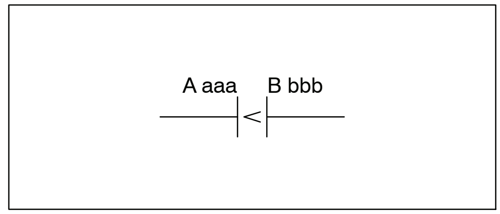
Store (STR)

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.



Store Not (STRN)

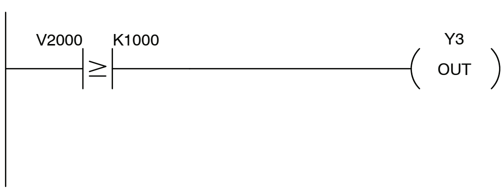
The Comparative Store Not instruction begins a new rung or additional branch in a rung with a normally closed comparative contact. The contact will be on when Aaaa < Bbbb.



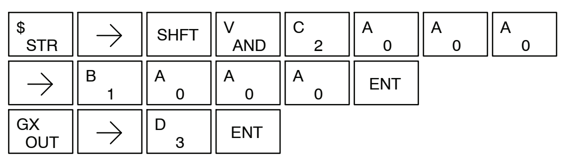
Operand Data Type	A/B	DL350 Range aaa	DL350 Range bbb
V–memory	V	All (See page 3–29)	All (See page 3–29)
Pointer	P	--	All V mem. (See page 3–29)
Constant	K	--	0–FFFF
Timer	T	0–377	
Counter	CT	0–177	

In the following example, when the value in V–memory location V2000 $\geq$ 1000, Y3 will energize.

***Direct*SOFT**

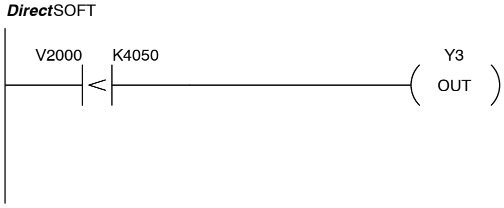


Handheld Programmer Keystrokes

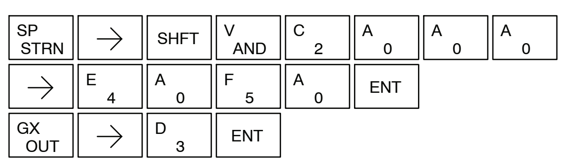


In the following example, when the value in V–memory location V2000 < 4050, Y3 will energize.

***Direct*SOFT**

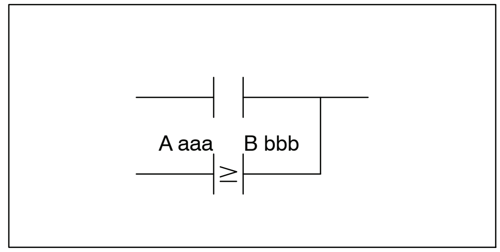


Handheld Programmer Keystrokes



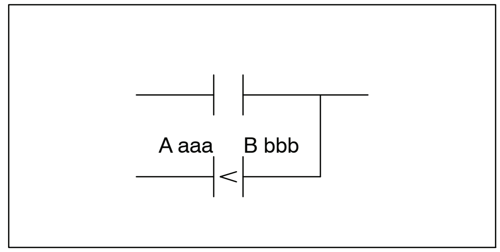
Or (OR)

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



Or Not (ORN)

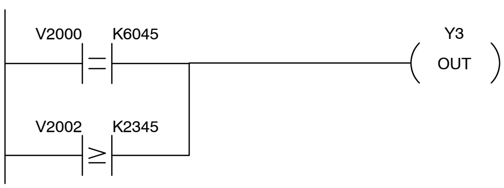
The Comparative Or Not instruction connects a normally open comparative contact in parallel with another contact. The contact will be on when Aaaa < Bbbb.



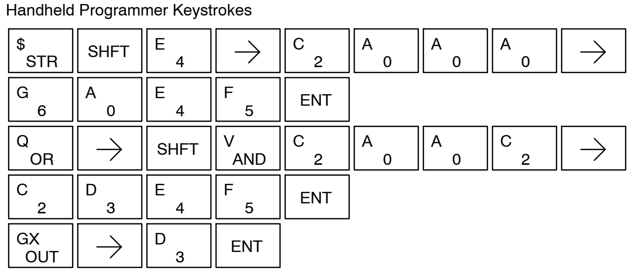
Operand Data Type		DL350 Range	
	A/B	aaa	bbb
V–memory	V	All (See page 3–29)	All (See page 3–29)
Pointer	P	--	All V mem. (See page 3–29)
Constant	K	--	0–FFFF
Timer	T	0–377	
Counter	CT	0–177	

In the following example, when the value in V–memory location V2000 = 6045 or V2002 $\geq$ 2345, Y3 will energize.

***Direct*SOFT**

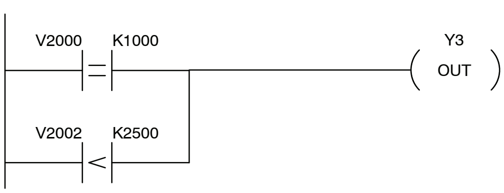


Handheld Programmer Keystrokes

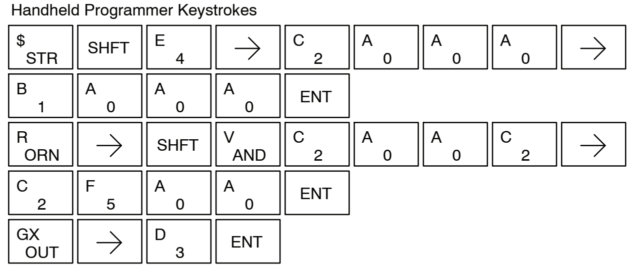


In the following example when the value in V–memory location V2000 = 1000 or V2002 < 2500, Y3 will energize.

***Direct*SOFT**

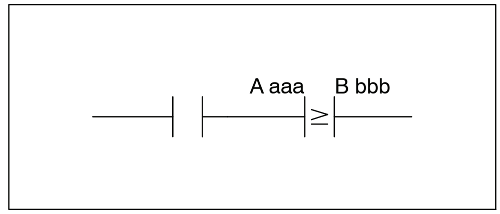


Handheld Programmer Keystrokes



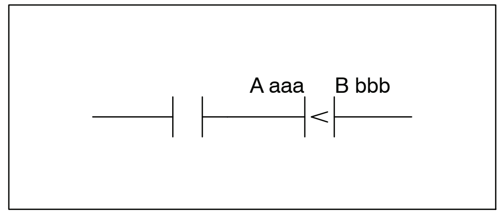
And (AND)

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



And Not (ANDN)

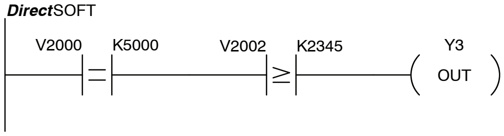
The Comparative And Not instruction connects a normally open comparative contact in series with another contact. The contact will be on when Aaaa < Bbbb.



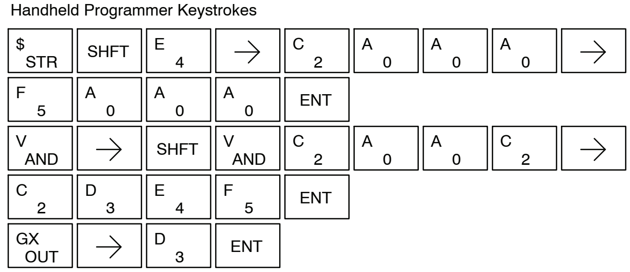
Operand Data Type		DL350 Range	
	A/B	aaa	bbb
V–memory	V	All (See page 3–29)	All (See page 3–29)
Pointer	P	--	All V mem. (See page 3–29)
Constant	K	--	0–FFFF
Timer	T	0–377	
Counter	CT	0–177	

In the following example, when the value in V–memory location V2000 = 5000, and V2002 $\geq$ 2345, Y3 will energize.

***Direct*SOFT**

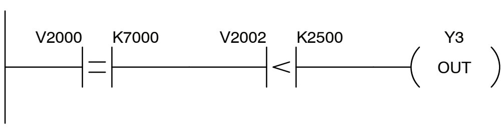


Handheld Programmer Keystrokes

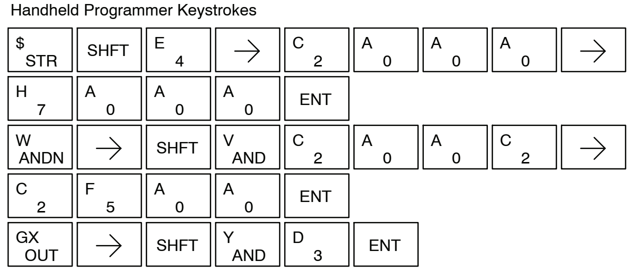


In the following example, when the value in V–memory location V2000 = 7000 and V2002 < 2500, Y3 will energize.

***Direct*SOFT**



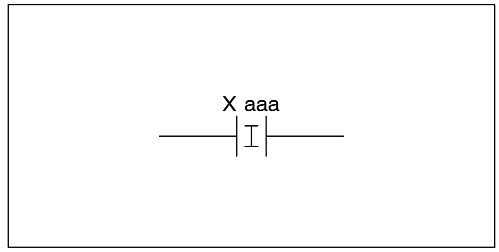
Handheld Programmer Keystrokes



Immediate Instructions

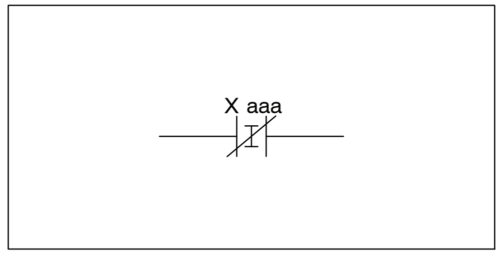
Store Immediate (STRI)

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.



Store Not Immediate (STRNI)

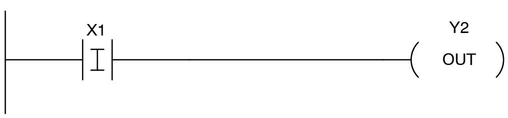
The Store Not Immediate instruction begins a new rung or additional branch in a rung. The status of the contact will be opposite the status of the associated input point on the module *at the time the instruction is executed*. The image register is not updated.



Operand Data Type		DL350 Range
		aaa
Inputs	X	0–777

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

***Direct*SOFT**



Handheld Programmer Keystrokes

$ STR	SHFT	I 8	→	B 1	ENT
GX OUT	→	C 2	ENT		

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

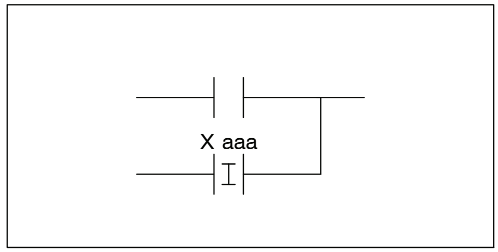
***Direct*SOFT**

Handheld Programmer Keystrokes

SP STRN	SHFT	I 8	→	B 1	ENT
GX OUT	→	C 2	ENT		

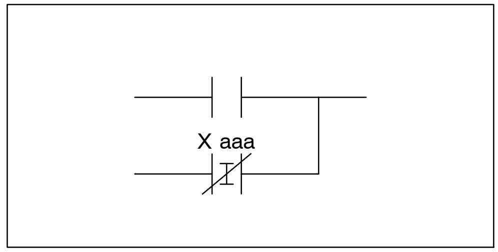
Or Immediate (ORI)

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)

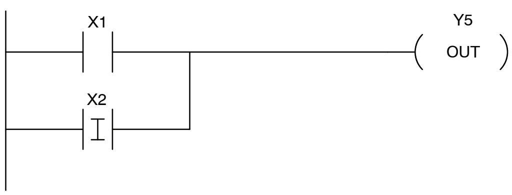
The Or Not Immediate connects two contacts in parallel. The status of the contact will be opposite the status of the associated input point on the module *at the time the instruction is executed*. The image register is not updated.



Operand Data Type		DL350 Range
		aaa
Inputs	X	0–777

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

***Direct*SOFT**

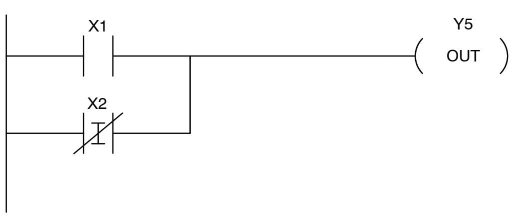


Handheld Programmer Keystrokes

$ STR	→	B 1	ENT		
Q OR	SHFT	I 8	→	C 2	ENT
GX OUT	→	F 5	ENT		

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

***Direct*SOFT**

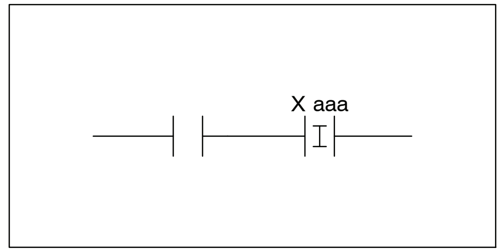


Handheld Programmer Keystrokes

$ STR	→	B 1	ENT		
R ORN	SHFT	I 8	→	C 2	ENT
GX OUT	→	F 5	ENT		

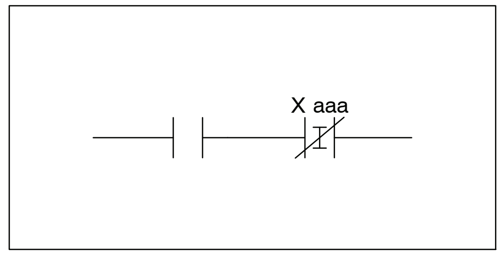
And Immediate (ANDI)

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)

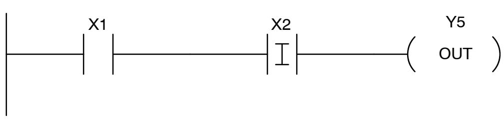
The And Not Immediate connects two contacts in series. The status of the contact will be opposite the status of the associated input point on the module *at the time the instruction is executed*. The image register is not updated.



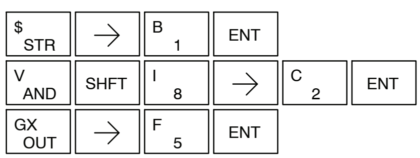
Operand Data Type		DL350 Range
		aaa
Inputs	X	0–777

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

***Direct*SOFT**

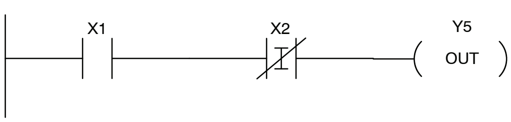


Handheld Programmer Keystrokes

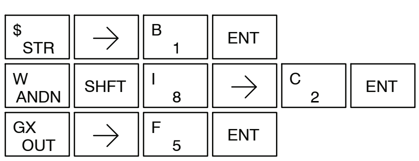


In the following example, when X1 is on and X2 is off, Y5 will energize.

***Direct*SOFT**

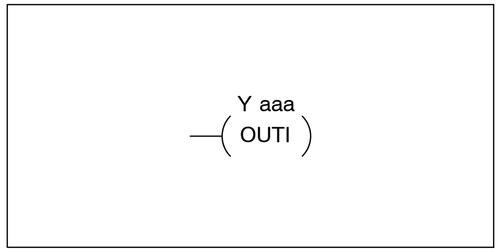


Handheld Programmer Keystrokes



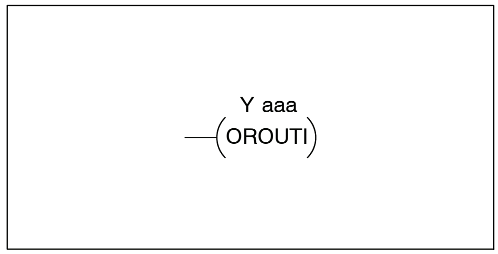
Out Immediate (OUTI)

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



Or Out Immediate (OROUTI)

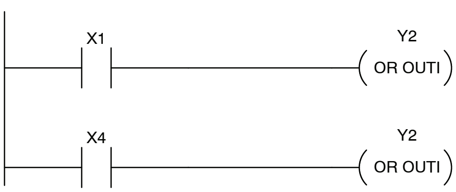
The Or Out Immediate instruction has been designed to use more than 1 rung of discrete logic to control a single output. Multiple Or Out Immediate instructions referencing the same output coil may be used, since all contacts controlling the output are ored together. If the status of *any* rung is on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



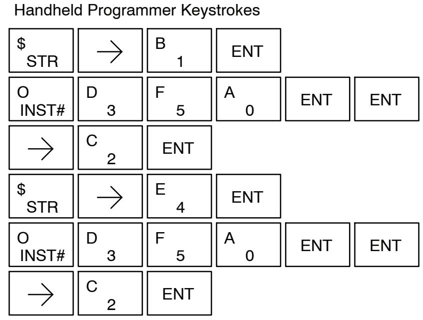
Operand Data Type		DL350 Range
		aaa
Outputs	Y	0–777

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

***Direct*SOFT**



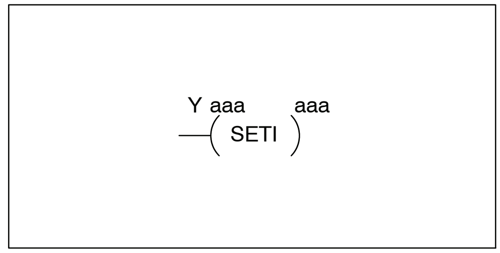
Handheld Programmer Keystrokes



Standard RLL Instructions

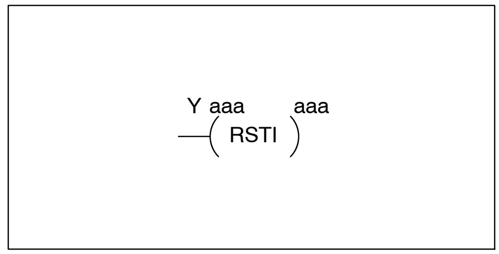
Set Immediate (SETI)

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



Reset Immediate (RSTI)

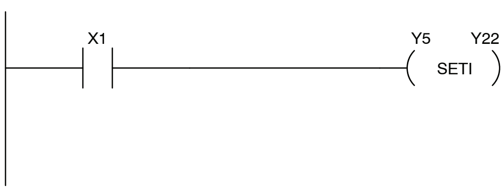
The Reset Immediate instruction immediately resets, or turns off an output or a range of outputs in the image register and the output module(s) *at the time the instruction is executed*. Once the outputs are reset it is not necessary for the input to remain on.



Operand Data Type		DL350 Range
		aaa
Outputs	Y	0–777

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

*Direct*SOFT



Handheld Programmer Keystrokes

$ STR	→	B 1	ENT					
X SET	SHFT	I 8	→	F 5	→	C 2	C 2	ENT

In the following example, when X1 is on, Y5 through Y22 will be reset (off) in the image register and on the corresponding output module(s).

*Direct*SOFT

Handheld Programmer Keystrokes

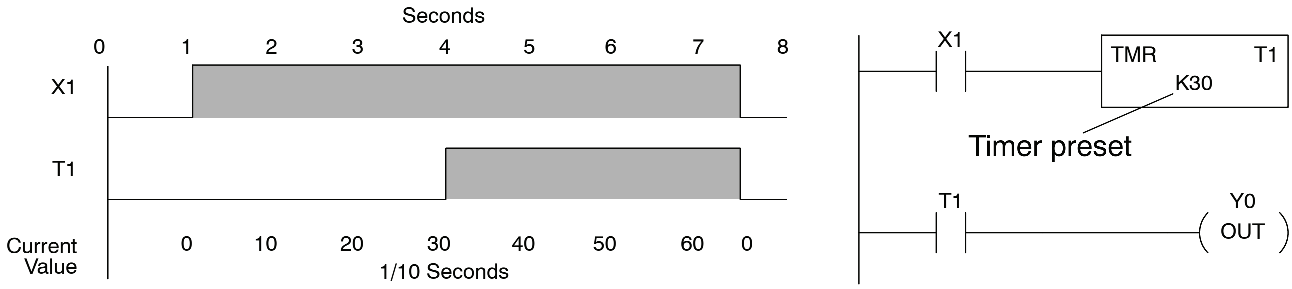
$ STR	→	B 1	ENT					
S RST	SHFT	I 8	→	F 5	→	C 2	C 2	ENT

Timer, Counter and Shift Register Instructions

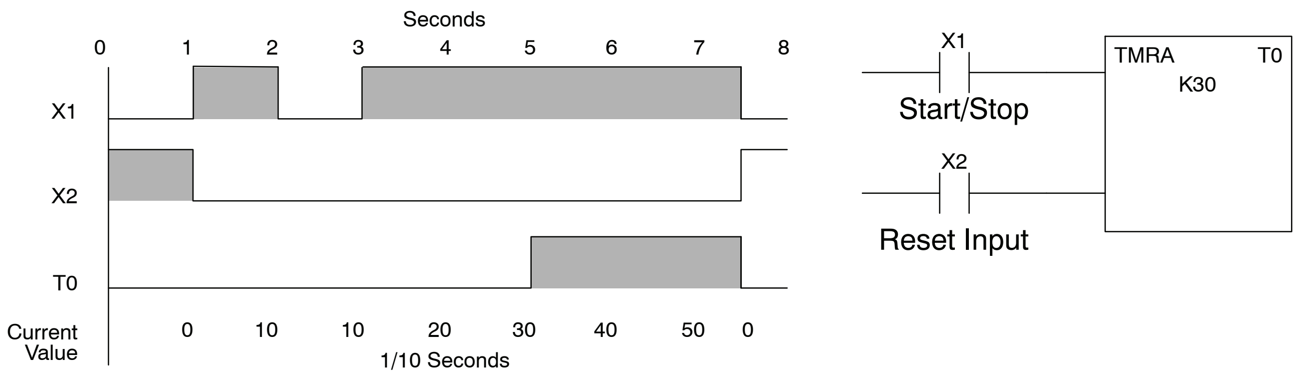
Using Timers

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.

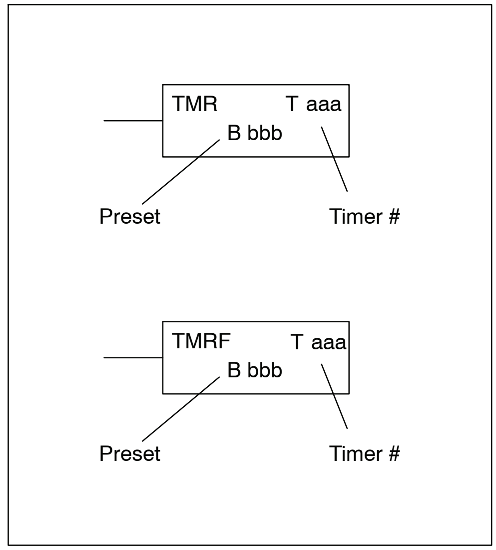


The accumulating timer works similarly to the regular timer, but two inputs are required. The start/stop input starts and stops the timer. When the timer stops, the elapsed time is maintained. When the timer starts again, the timing continues from the elapsed time. When the reset input is turned on, 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.



Timer (TMR) and Timer Fast (TMRF)

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).



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

Instruction Specifications

Timer Reference (Taaa): Specifies the timer number.

Preset Value (Bbbb): Constant value (K), V–memory location, or Pointer (P).

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.

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		DL350 Range	
	A/B	aaa	bbb
Timers	T	0–377	––
V–memory for preset values	V	––	All Data Words (See Page 3–29)
Pointers (preset only)	P	––	All Data Words (See Page 3–29)
Constants (preset only)	K	––	0–9999
Timer discrete status bits	T/V	0–377	
Timer current values	V /T*	0–377	

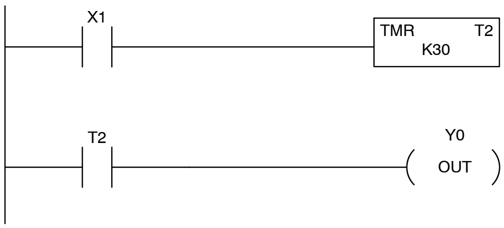
There are two methods of programming timers. You can perform functions when the timer reaches the specified preset using the the discrete status bit, or use the comparative contacts to perform functions at different time intervals based on one timer. The following examples show each method of using timers.

NOTE: The current value of a timer can be accessed by using the TA data type (i.e., TA2). Current values may also be accessed by the V-memory location.

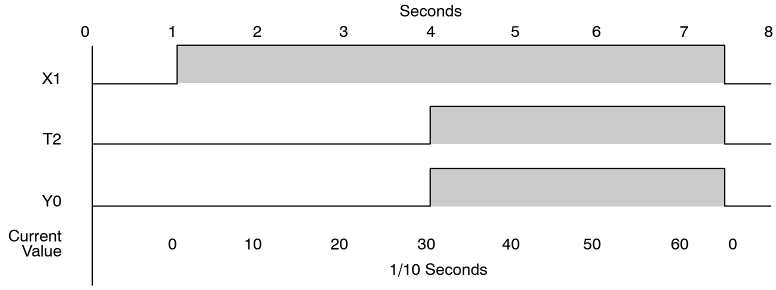
Timer Example Using Discrete Status Bits

In the following example, a single input timer is used with a preset of 3 seconds. The timer discrete status bit (T2) will turn on when the timer has timed for 3 seconds. The timer is reset when X1 turns off, turning the discrete status bit off and resetting the timer current value to 0.

***Direct*SOFT**



Timing Diagram



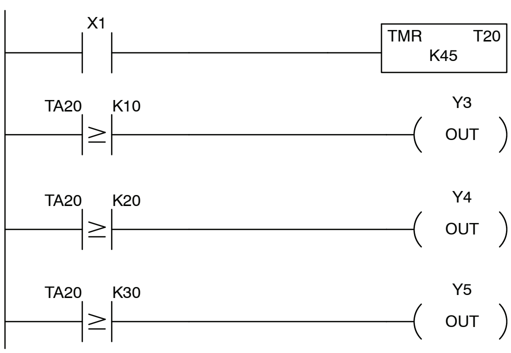
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT			
N TMR	→	C 2	→	D 3	A 0	ENT
$ STR	→	SHFT	T MLR	C 2	ENT	
GX OUT	→	A 0	ENT			

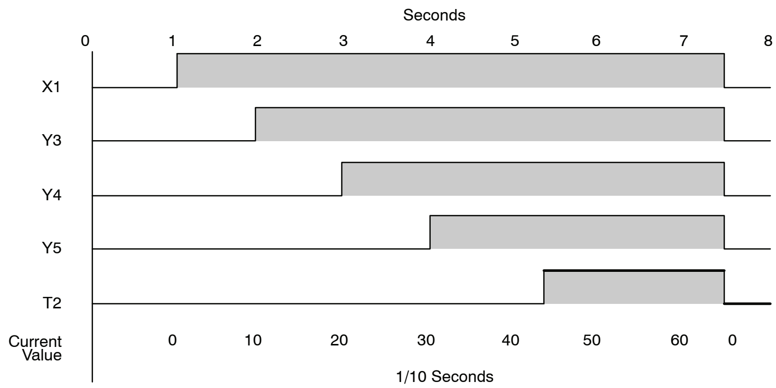
Timer Example Using Comparative Contacts

In the following example, a single input timer is used with a preset of 4.5 seconds. Comparative contacts are used to energize Y3, Y4, and Y5 at one second intervals respectively. When X1 is turned off the timer will be reset to 0 and the comparative contacts will turn off Y3, Y4, and Y5.

***Direct*SOFT**



Timing Diagram



Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
N TMR	→	C 2	A 0	→	E 4	F 5	ENT		
$ STR	→	SHFT	T MLR	C 2	A 0	→	B 1	A 0	ENT
GX OUT	→	D 3	ENT						
$ STR	→	SHFT	T MLR	C 2	A 0	→	C 2	A 0	ENT
GX OUT	→	E 4	ENT						
$ STR	→	SHFT	T MLR	C 2	A 0	→	D 3	A 0	ENT
GX OUT	→	F 5	ENT						

Accumulating Timer (TMRA) Accumulating Fast Timer (TMRAF)

Incorrect. Should be 999999.99

The Accumulating Timer is a 0.1 second two input timer that will time to a maximum of 9999999.9. The Accumulating Fast Timer is a 0.01 second two input timer that will time to a maximum of ~~99999.99~~. 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.

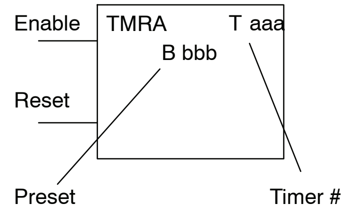
Instruction Specifications

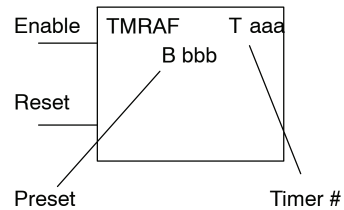
Timer Reference (Taaa): Specifies the timer number.

Preset Value (Bbbb): Constant value (K), V–memory location,or Pointer (P).

Current Value: Timer current values are accessed by referencing the associated V or T memory location (See Note). For example, the timer current value for T3 resides in V-memory location V3.

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.





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 would be T2. Or if T0 was a normal timer, and T1 was an accumulating timer, the next available timer would be T3.

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

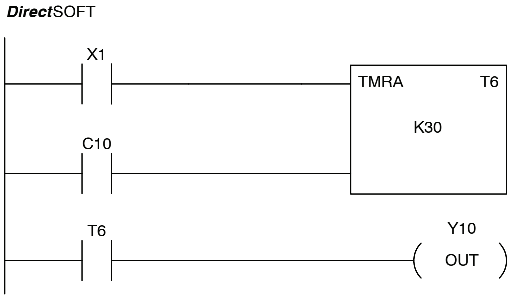
Operand Data Type		DL350 Range	
	A/B	aaa	bbb
Timers	T	0–377	––
V–memory for preset values	V	––	All Data Words (See Page 3–29)
Pointers (preset only)	P	––	All Data Words (See Page 3–29)
Constants (preset only)	K	––	0–9999
Timer discrete status bits	T/V	0–377 or V41100–41117	
Timer current values	V /T*	0–377	

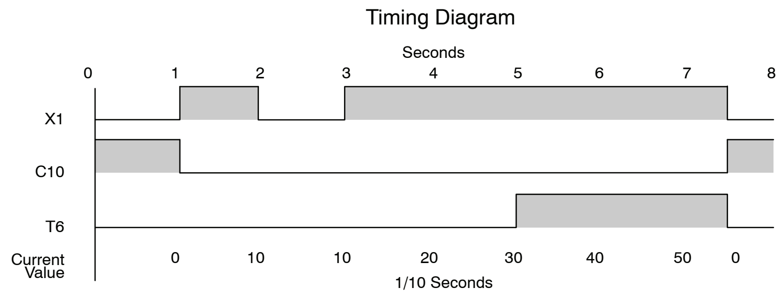
There are two methods of programming timers. You can perform functions when the timer reaches the specified preset using the the discrete status bit, or use the comparative contacts to perform functions at different time intervals based on one timer. The following examples show each method of using timers.

NOTE: The current value of a timer can be accessed by using the TA data type (i.e., TA2). Current values may also be accessed by the V-memory location.

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 turns on, turning the discrete status bit off and resetting the timer current value to 0.





Handheld Programmer Keystrokes

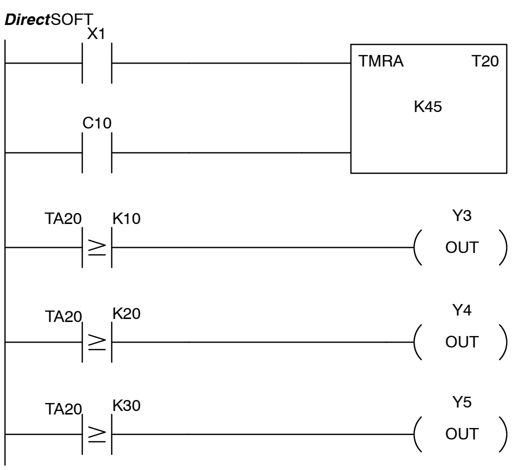
$ STR	→	B 1	ENT			
$ STR	→	SHFT	C 2	B 1	A 0	ENT
N TMR	SHFT	A 0	→	G 6	→	

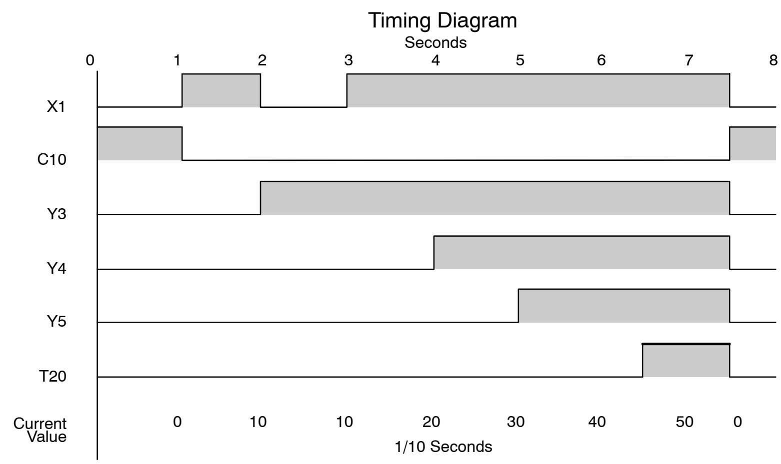
Handheld Programmer Keystrokes (cont)

D 3	A 0	ENT			
$ STR	→	SHFT	T MLR	G 6	ENT
GX OUT	→	B 1	A 0	ENT	

Accumulator Timer Example Using Comparative Contacts

In the following example, a single input timer is used with a preset of 4.5 seconds. Comparative contacts are used to energized Y3, Y4, and Y5 at one second intervals respectively. The comparative contacts will turn off when the timer is reset.





Handheld Programmer Keystrokes

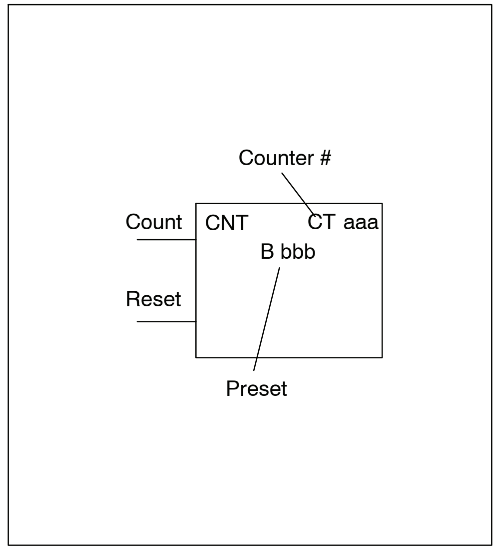
$ STR	→	B 1	ENT						
$ STR	→	SHFT	C 2	B 1	A 0	ENT			
N TMR	SHFT	A 0	→	C 2	A 0	→	E 4	F 5	ENT
$ STR	→	SHFT	T MLR	C 2	A 0	→	B 1	A 0	ENT
GX OUT	→	D 3	ENT						
$ STR	→	SHFT	T MLR	C 2	A 0	→	C 2	A 0	ENT

Handheld Programmer Keystrokes (cont)

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

Counter (CNT)

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.



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), V–memory location, or Pointer (P).

Current Values: Counter current values are accessed by referencing the associated V or CT memory locations*. The V-memory location is the counter location + 1000. For example, the counter current value for CT3 resides in V–memory location V1003.

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

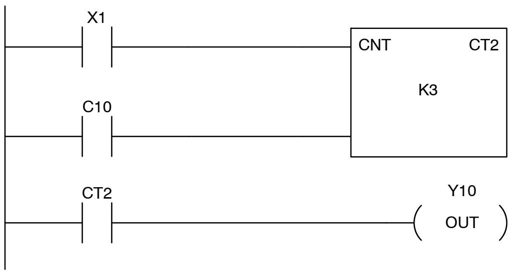
Operand Data Type		DL350 Range	
	A/B	aaa	bbb
Counters	CT	0–177	--
V–memory (preset only)	V	--	All Data Words (See Page 3–29)
Pointers (preset only)	P	--	All Data Words (See Page 3–29)
Constants (preset only)	K	--	0–9999
Counter discrete status bits	CT/V	0–177 or V41140–41147	
Counter current values	V/CT*	1000–1177	

NOTE: The current value of a counter can be accessed by using the CTA data type (i.e., CTA2). Current values may also be accessed by the V-memory location.

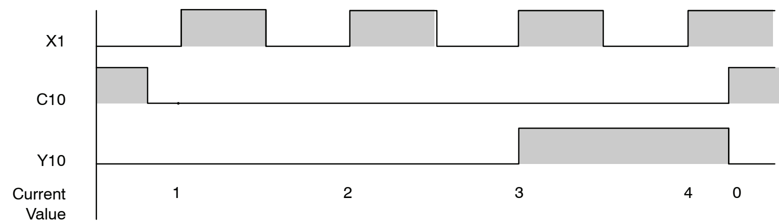
Counter Example Using Discrete Status Bits

In the following example, when X1 makes an off to on transition, counter CTA2 will increment by one. When the current value reaches the preset value of 3, the counter status bit CTA2 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 CTA2 will be held in V–memory location V1002.

Direct**SOFT**



Counting diagram



Handheld Programmer Keystrokes

$ STR	→	B 1	ENT			
$ STR	→	SHFT	C 2	B 1	A 0	ENT
GY CNT	→	C 2	→	D 3	ENT	

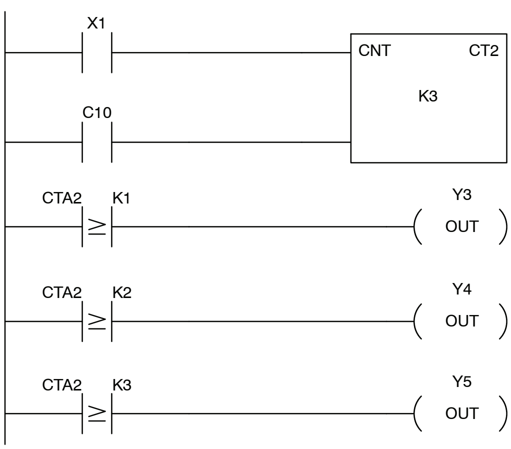
Handheld Programmer Keystrokes (cont)

$ STR	→	SHFT	C 2	SHFT	T MLR	C 2	ENT
GX OUT	→	B 1	A 0	ENT			

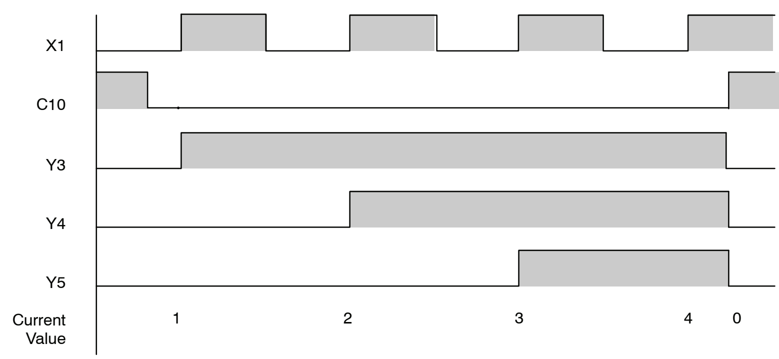
Counter Example Using Comparative Contacts

In the following example, when X1 makes an off to on transition, counter CTA2 will increment by one. Comparative contacts are used to energize Y3, Y4, and Y5 at different counts. When the reset C10 turns on, the counter status bit will turn off and the counter current value will be 0, and the comparative contacts will turn off.

Direct**SOFT**



Counting diagram



Handheld Programmer Keystrokes

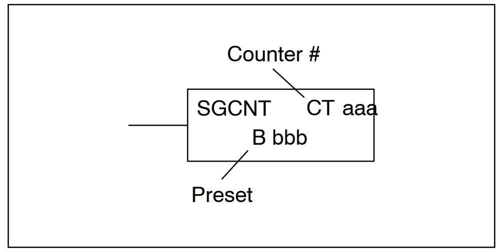
$ STR	→	B 1	ENT			
$ STR	→	SHFT	C 2	B 1	A 0	ENT
GY CNT	→	C 2	→	D 3	ENT	
$ STR	→	SHFT	C 2	SHFT	T MLR	C 2
→	B 1	ENT				
GX OUT	→	D 3	ENT			

Handheld Programmer Keystrokes (cont)

$ STR	→	SHFT	C 2	SHFT	T MLR	C 2
→	C 2	ENT				
GX OUT	→	E 4	ENT			
$ STR	→	SHFT	C 2	SHFT	T MLR	C 2
→	D 3	ENT				
GX OUT	→	F 5	ENT			

Stage Counter (SGCNT)

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), V–memory location or Pointer (P).

Current Values: Counter current values are accessed by referencing the associated V or CTA memory locations*. The V-memory location is the counter location + 1000. For example, the counter current value for CT3 resides in V–memory location V1003.

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

Operand Data Type		DL350 Range	
	A/B	aaa	bbb
Counters	CT	0–177	--
V–memory (preset only)	V	--	All Data Words (See Page 3–29)
Pointers (preset only)	P	--	All Data Words (See Page 3–29)
Constants (preset only)	K	--	0–9999
Counter discrete status bits	CT/V	0–177 or V41140–41147	
Counter current values	V/CTA*	1000–1177	

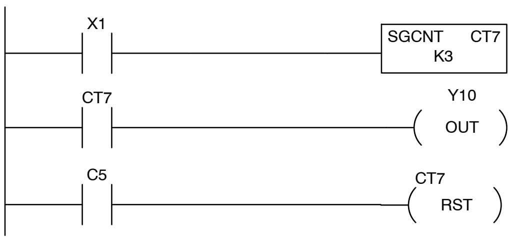
NOTE: The current value of a counter can be accessed by using the CTA data type (i.e., CTA2). Current values may also be accessed by the V-memory location.

Standard RLL Instructions

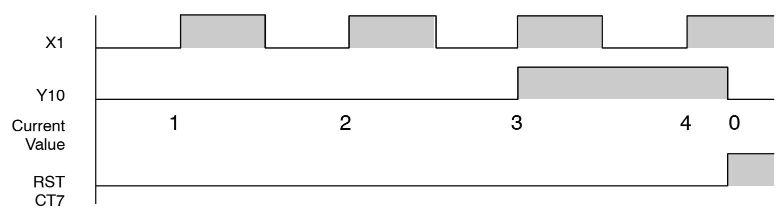
Stage Counter Example Using Discrete Status Bits

In the following example, when X1 makes an off to on transition, stage counter CTA7 will increment by one. When the current value reaches 3, the counter status bit CTA7 will turn on and energize Y10. The counter status bit CTA7 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 current value will be 0. The current value for counter CTA7 will be held in V–memory location V1007.

*Direct*SOFT



Counting diagram



Handheld Programmer Keystrokes

$ STR	→	B 1	ENT			
SHFT	S RST	SHFT	G 6	SHFT	GY CNT	→
H 7	→	D 3	ENT			
$ STR	→	SHFT	C 2	SHFT	T MLR	H 7	ENT

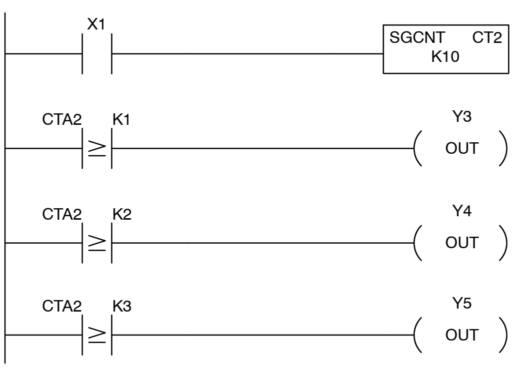
Handheld Programmer Keystrokes (cont)

GX OUT	→	B 1	A 0	ENT			
$ STR	→	SHFT	C 2	F 5	ENT		
S RST	→	SHFT	C 2	SHFT	T MLR	H 7	ENT

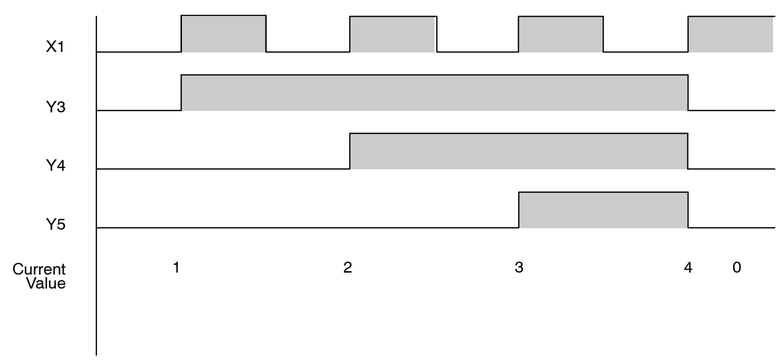
Stage Counter Example Using Comparative Contacts

In the following example, when X1 makes an off to on transition, counter CTA2 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 CTA2 will be held in V–memory location V1007.

*Direct*SOFT



Counting diagram



Handheld Programmer Keystrokes

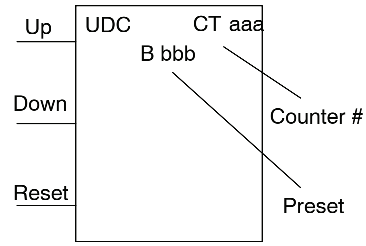
$ STR	→	B 1	ENT			
SHFT	S RST	G 6	SHFT	GY CNT	→	
C 2	→	B 1	A 0	ENT		
$ STR	→	SHFT	C 2	SHFT	T MLR	C 2
→	B 1	ENT				
GX OUT	→	D 3	ENT			

Handheld Programmer Keystrokes (cont)

$ STR	→	SHFT	C 2	SHFT	T MLR	C 2
→	C 2	ENT				
GX OUT	→	E 4	ENT			
$ STR	→	SHFT	C 2	SHFT	T MLR	C 2
→	D 3	ENT				
GX OUT	→	F 5	ENT			

Up Down Counter (UDC)

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



Caution : The UDC uses two V memory locations for the 8 digit current value. This means the UDC uses two consecutive counter locations. If UDC CT1 is used in the program, the next available counter is CT3.

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

Instruction Specification

Counter Reference (CTaaa): Specifies the counter number.

Preset Value (Bbbb): Constant value (K), V–memory locations, or Pointer (P).

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.

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.

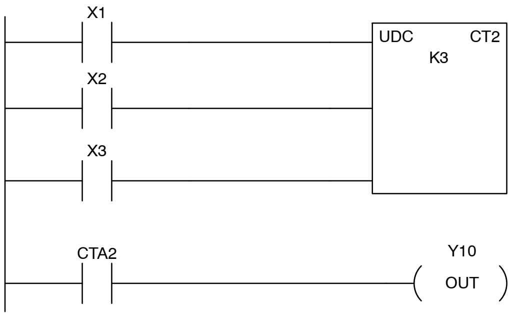
Operand Data Type		DL350 Range	
	A/B	aaa	bbb
Counters	CT	0–177	--
V–memory (preset only)	V	--	All Data Words (See Page 3–29)
Pointers (preset only)	P	--	All Data Words (See Page 3–29)
Constants (preset only)	K	--	0–99999999
Counter discrete status bits	CT/V	0–177 or V41140–41147	
Counter current values	V/CTA*	1000–1177	

NOTE: The current value of a counter can be accessed by using the CTA data type (i.e., CTA2). Current values may also be accessed by the V-memory location.

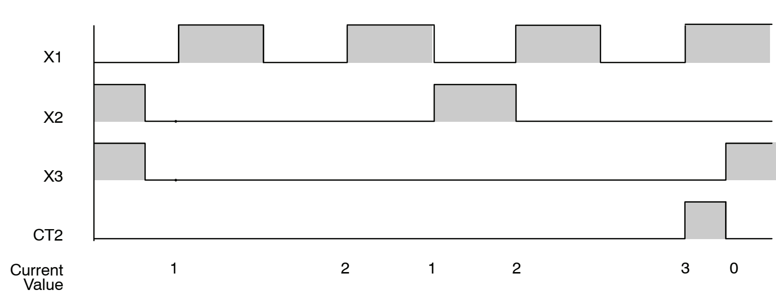
Up / Down Counter Example Using Discrete Status Bits

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	→	B 1	ENT		
$ STR	→	C 2	ENT		
$ STR	→	D 3	ENT		
SHFT	U ISG	D 3	C 2	→	C 2

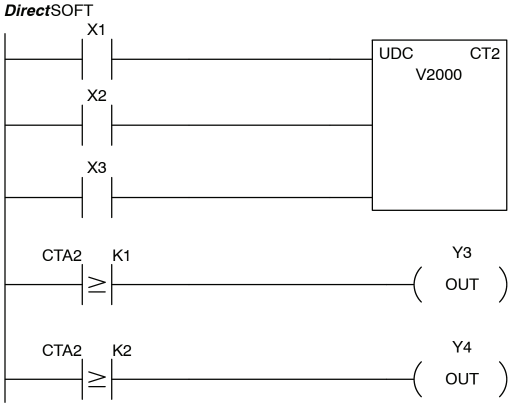
Handheld Programmer Keystrokes (cont)

→	D 3	ENT					
$ STR	→	SHFT	C 2	SHFT	T MLR	C 2	ENT
GX OUT	→	B 1	A 0	ENT			

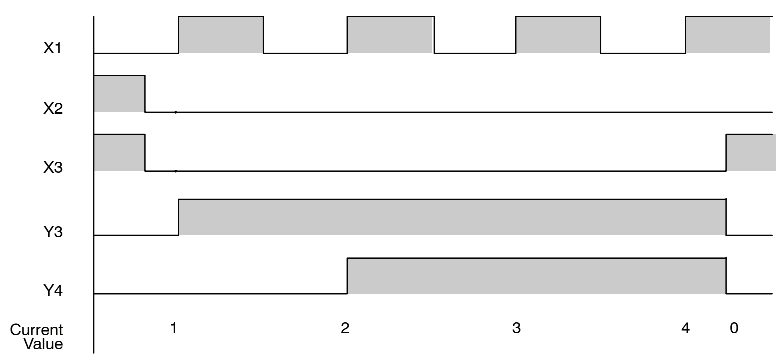
Up / Down Counter Example Using Comparative Contacts

In the following example, when X1 makes an off to on transition, counter CTA2 will increment by one. Comparative contacts are used to energize Y3 and Y4 at different counts. When the reset (X3) turns on, the counter status bit will turn off, the current value will be 0, and the comparative contacts will turn off.

DirectSOFT



Counting Diagram



Handheld Programmer Keystrokes

$ STR	→	B 1	ENT			
$ STR	→	C 2	ENT			
$ STR	→	D 3	ENT			
SHFT	U ISG	D 3	C 2	→	C 2	→
SHFT	V AND	C 2	A 0	A 0	A 0	ENT
$ STR	→	SHFT	C 2	SHFT	T MLR	C 2

Handheld Programmer Keystrokes (cont)

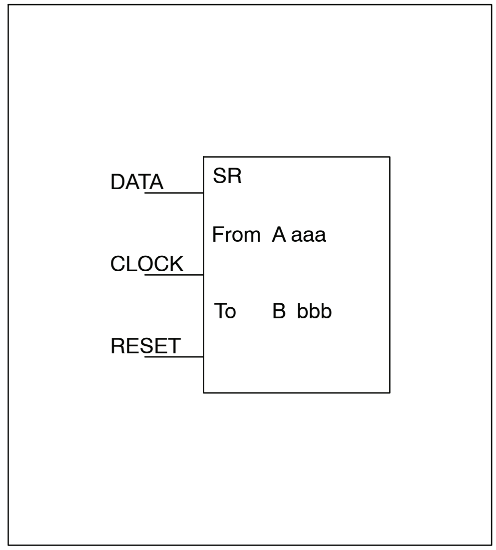
→	B 1	ENT				
GX OUT	→	D 3	ENT			
$ STR	→	SHFT	C 2	SHFT	T MLR	C 2
→	C 2	ENT				
GX OUT	→	E 4	ENT			

Shift Register (SR)

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.

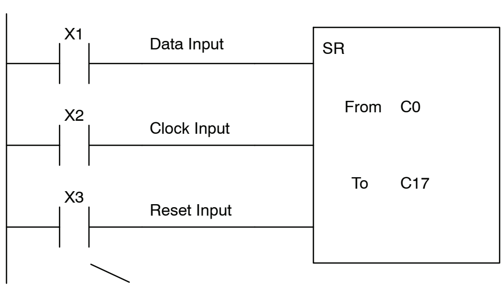
- Data — determines the value (1 or 0) that will enter the register
- Clock — shifts the bits one position on each low to high transition
- Reset —resets the Shift Register to all zeros.



With each off to on transition of the clock input, the bits which make up the shift register block are shifted by one bit position and the status of the data input is placed into the starting bit position in the shift register. The direction of the shift depends on the entry in the From and To fields. From C0 to C17 would define a block of sixteen bits to be shifted from left to right. From C17 to C0 would define a block of sixteen bits, to be shifted from right to left. The maximum size of the shift register block depends on the number of available control relays. The minimum block size is 8 control relays.

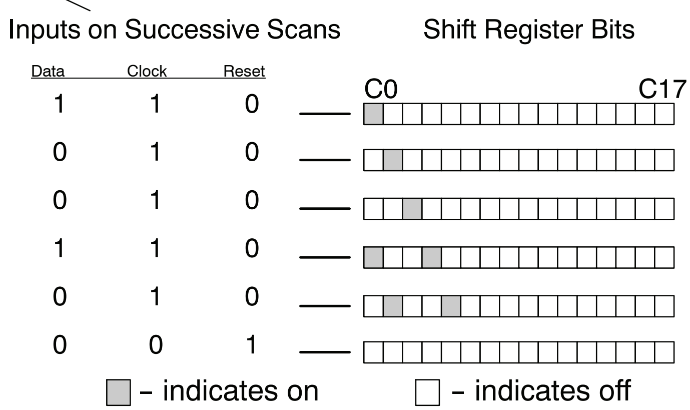
Operand Data Type		DL350 Range	
	A/B	aaa	bbb
Control Relay	C	0–1777	0–1777

*Direct*SOFT



Handheld Programmer Keystrokes

$ STR	→	B 1	ENT			
$ STR	→	C 2	ENT			
$ STR	→	D 3	ENT			
SHFT	S RST	SHFT	R ORN	SHFT	→	A 0
→	B 1	H 7	ENT			



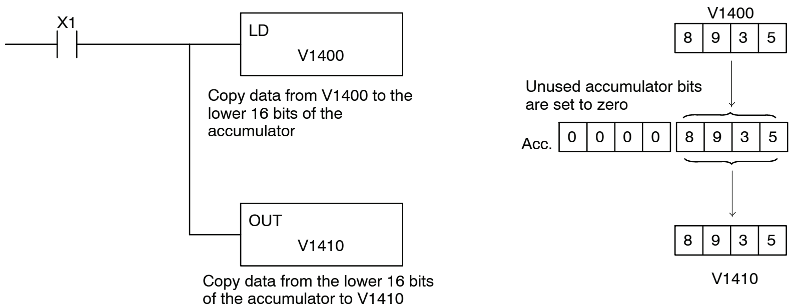
Accumulator / Stack Load and Output Data Instructions

Using the Accumulator

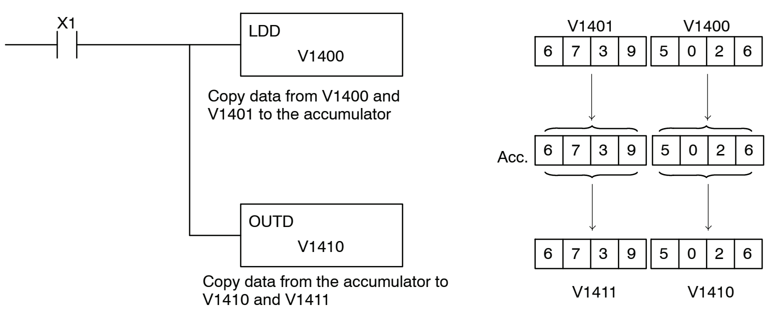
The accumulator in the DL350 CPU is a 32 bit register which is used as a temporary storage location for data that is being copied or manipulated in some manor. For example, you have to use the accumulator to perform math operations such as add, subtract, multiply, etc. Since there are 32 bits, you can 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.

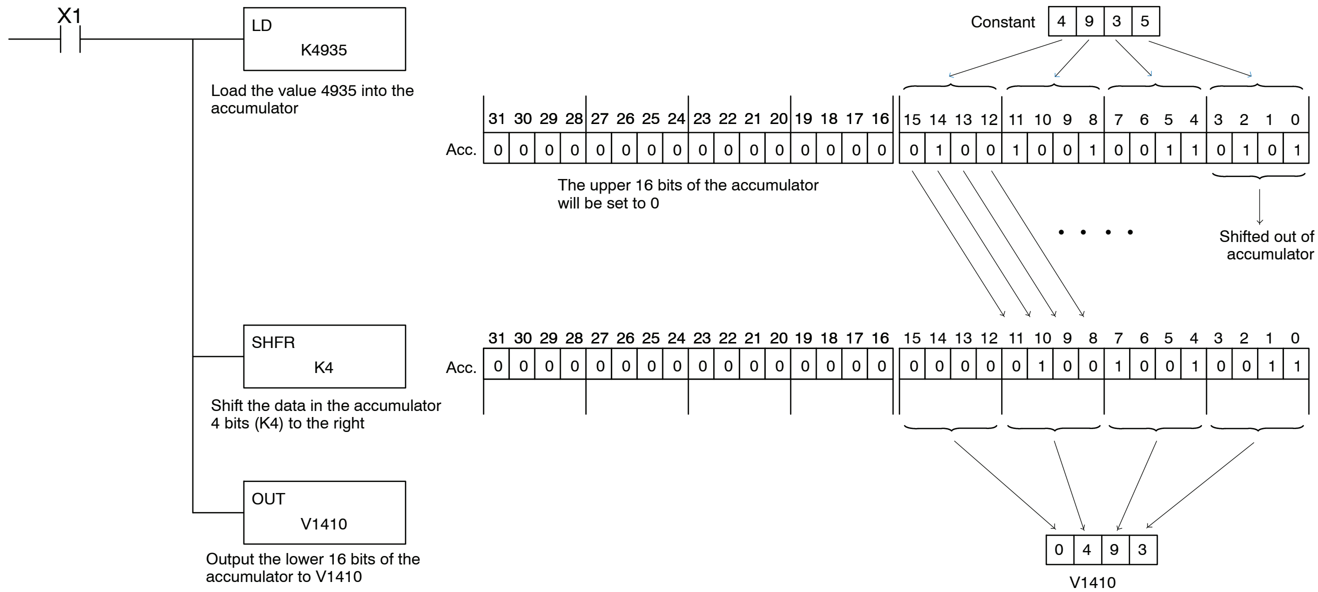


Since the accumulator is 32 bits and V–memory locations are 16 bits the Load Double and Out Double (or variations thereof) use two consecutive V–memory locations or 8 digit BCD constants to copy data either to the accumulator from a V–memory address or from a V–memory address to the accumulator. For 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:

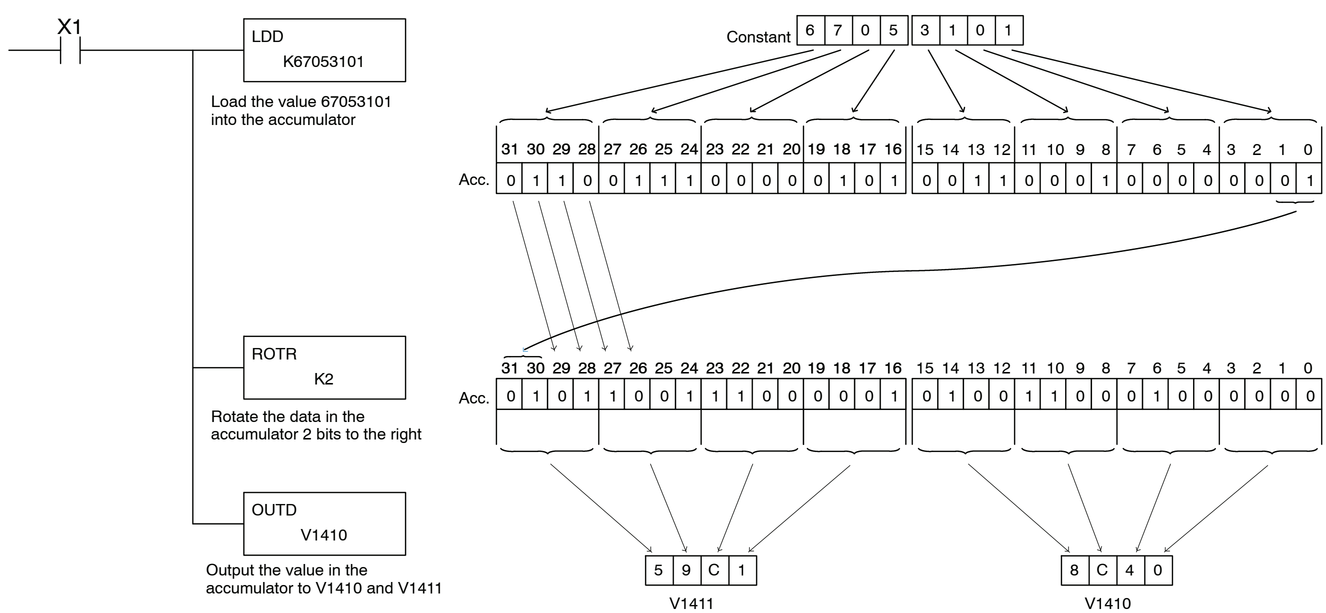


Changing the Accumulator Data

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.

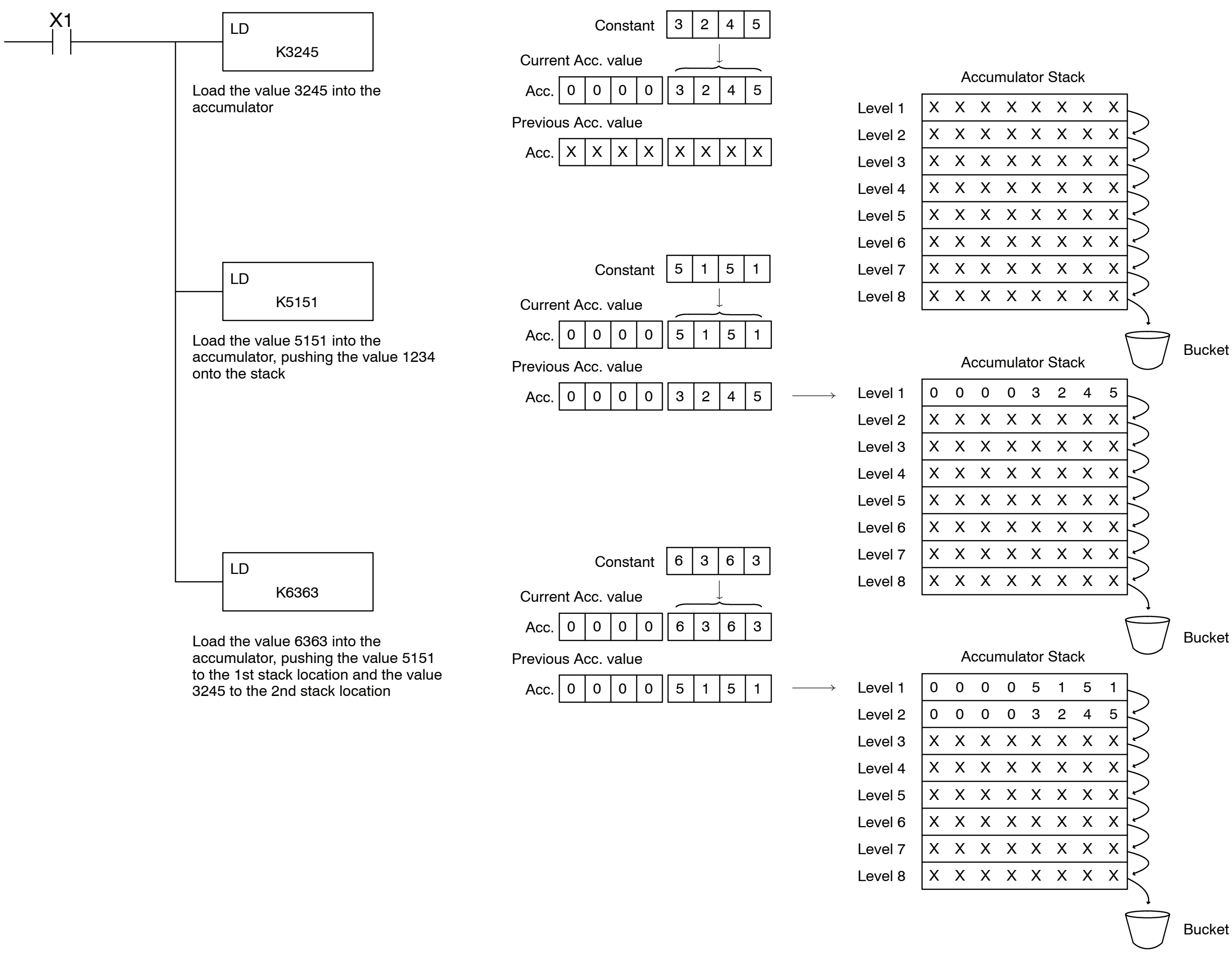


Some of the data manipulation instructions use 32 bits. They use two consecutive V-memory locations or 8 digit BCD constants to manipulate data in the accumulator. The following example rotates the value 67053101 two bits to the right and outputs the value to V1410 and V1411.

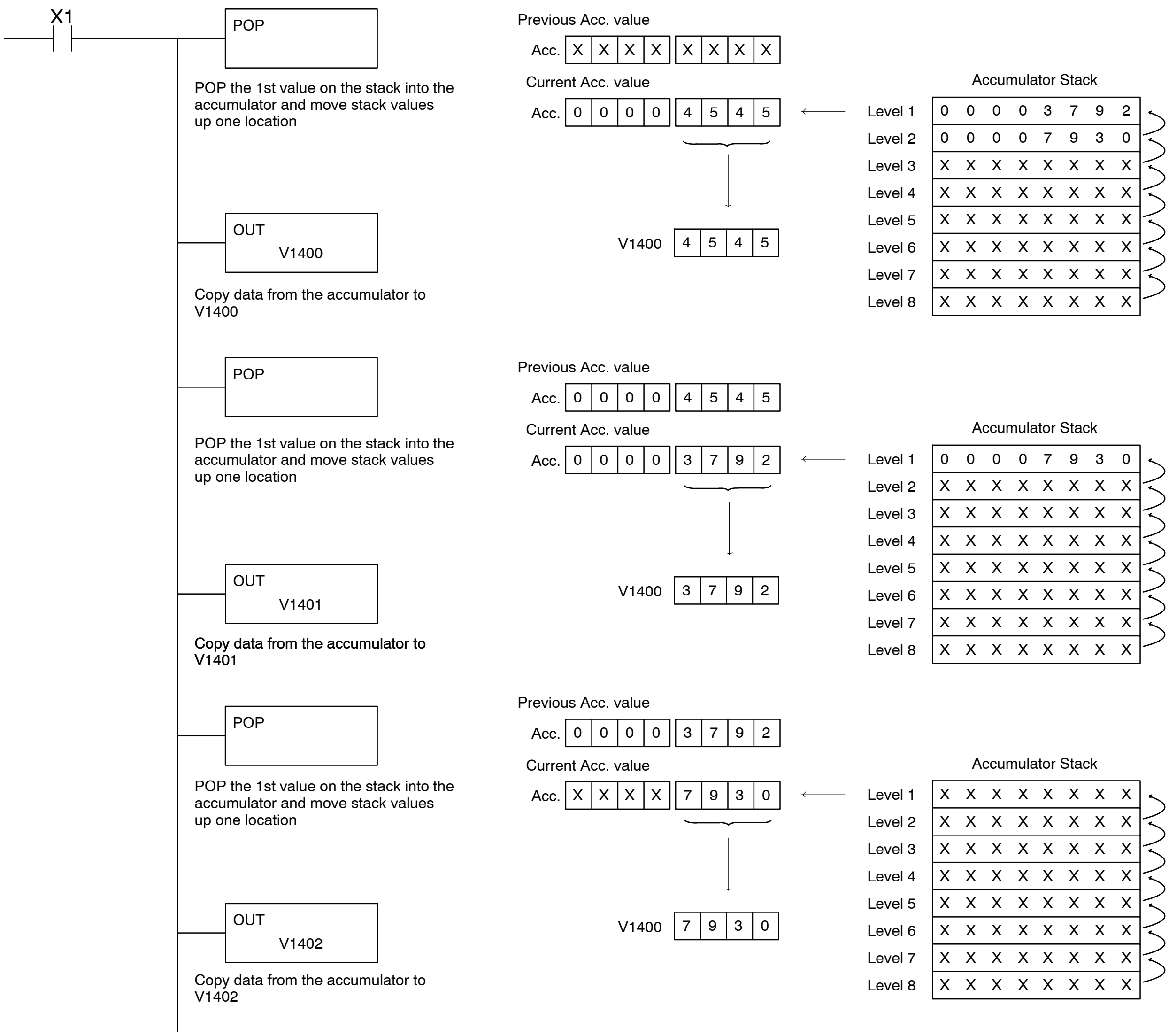


Using the Accumulator Stack

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.

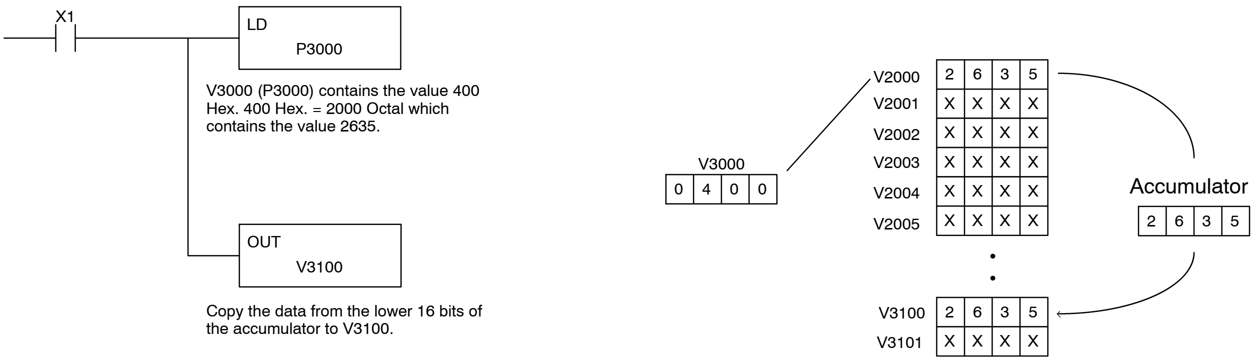

X1
POP
POP the 1st value on the stack into the accumulator and move stack values up one location
Previous Acc. value
Acc. X X X X X X X X
Current Acc. value
Acc. 0 0 0 0 4 5 4 5
Accumulator Stack
Level 1 0 0 0 0 3 7 9 2
Level 2 0 0 0 0 7 9 3 0
Level 3 X X X X X X X X
Level 4 X X X X X X X X
Level 5 X 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
OUT V1400
Copy data from the accumulator to V1400
V1400 4 5 4 5
POP
POP the 1st value on the stack into the accumulator and move stack values up one location
Previous Acc. value
Acc. 0 0 0 0 4 5 4 5
Current Acc. value
Acc. 0 0 0 0 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
OUT V1401
Copy data from the accumulator to V1401
V1400 3 7 9 2
POP
POP the 1st value on the stack into the accumulator and move stack values up one location
Previous Acc. value
Acc. 0 0 0 0 3 7 9 2
Current Acc. value
Acc. X X X X 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
OUT V1402
Copy data from the accumulator to V1402
V1400 7 9 3 0

Using Pointers

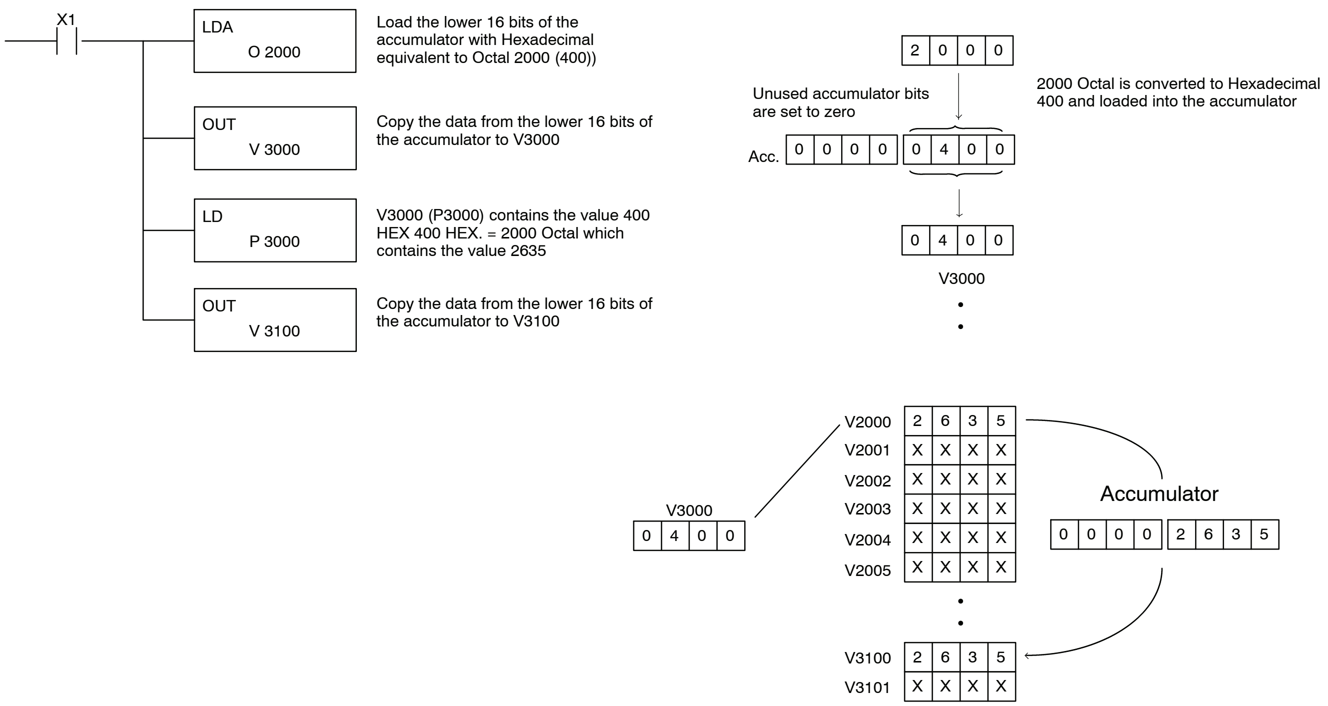
Many of the instructions will allow V–memory pointers as a operand. Pointers can be useful in ladder logic programming, but can be difficult to understand or implement in your application if you do not have prior experience with pointers (commonly known as indirect addressing). Pointers allow instructions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

NOTE: 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 3000 is the pointer location. V3000 contains the value 400 which is the HEX equivalent of the Octal address V-memory location V2000. The CPU copies the data from V2000 into the lower word of the accumulator.

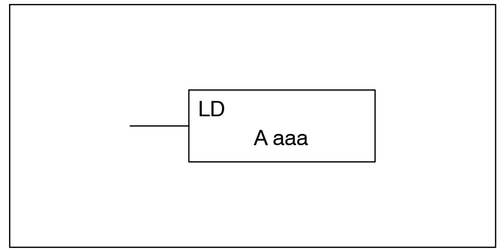


The following example is similar to the one above, except for the LDA (load address) instruction which automatically converts the Octal address to the Hex equivalent.



Load (LD)

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



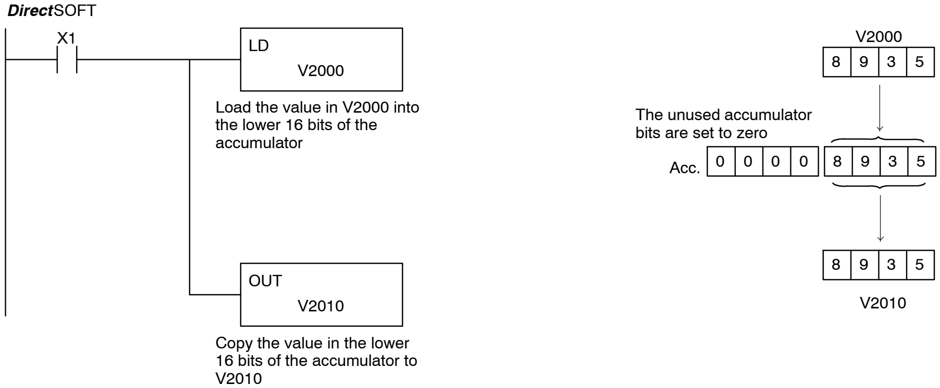
Operand Data Type		DL350 Range
	A	aaa
V–memory	V	All (See page 3–29)
Pointer	P	All V mem. (See page 3–29)
Constant	K	0–FFFF

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

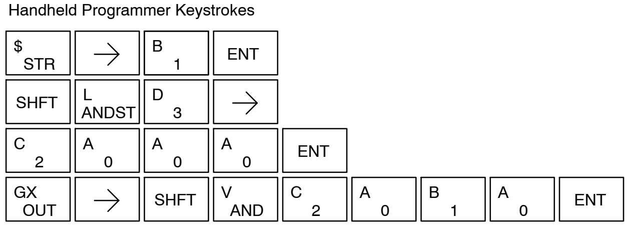
NOTE: Two consecutive Load instructions will place the value of the first load instruction onto the accumulator stack.

In the following example, when X1 is on, the value in V2000 will be loaded into the accumulator and output to V2010.

DirectSOFT

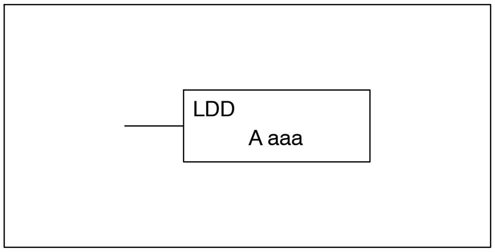


Handheld Programmer Keystrokes



Load Double (LDD)

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.



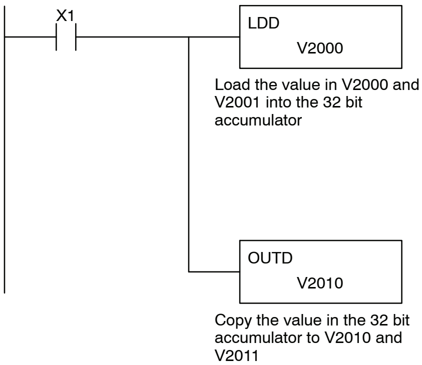
Operand Data Type		DL350 Range
	A	**aaa**
V–memory	V	All (See page 3–29)
Pointer	P	All V mem. (See page 3–29)
Constant	K	0–FFFF

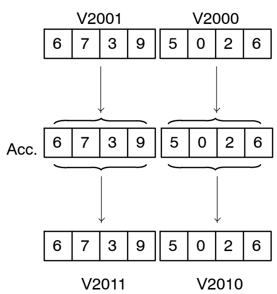
Discrete Bit Flags	Description
SP76	on when the value loaded into the accumulator by any instruction is zero.

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 32 bit value in V2000 and V2001 will be loaded into the accumulator and output to V2010 and V2011.

***Direct*SOFT**



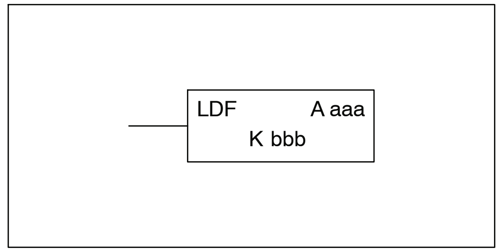


Handheld Programmer Keystrokes

$ STR	→	B 1	ENT	
SHFT	L ANDST	D 3	D 3	→
C 2	A 0	A 0	A 0	ENT
GX OUT	SHFT	D 3	→	
C 2	A 0	B 1	A 0	ENT

Load Formatted (LDF)

The Load Formatted instruction loads 1-32 consecutive bits from discrete memory locations into the accumulator. The instruction requires a starting location (Aaaa) and the number of bits (Kbbb) to be loaded. Unused accumulator bit locations are set to zero.

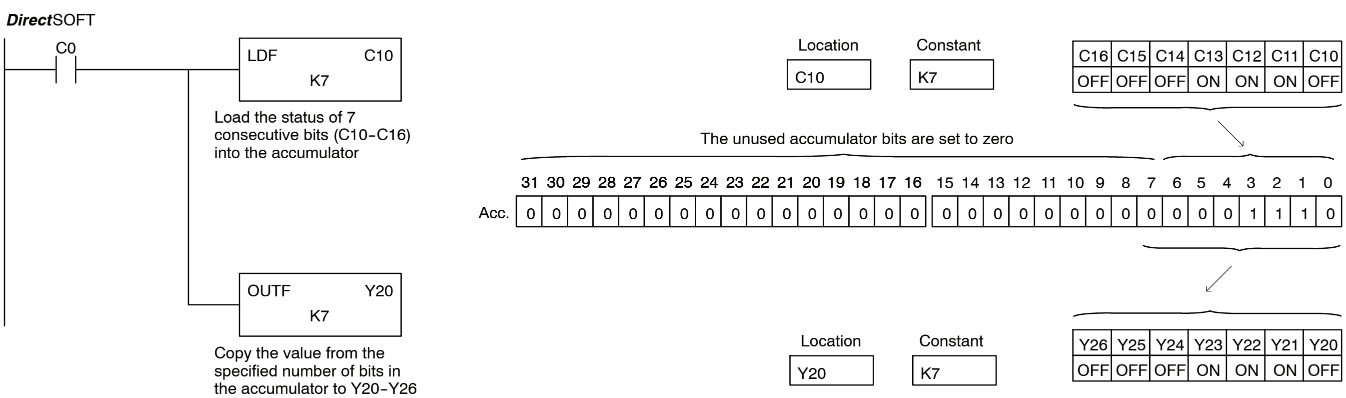


Operand Data Type		DL350 Range	
	A	aaa	bbb
Inputs	X	0-777	--
Outputs	Y	0-777	--
Control Relays	C	0-1777	--
Stage Bits	S	0-1777	--
Timer Bits	T	0-377	--
Counter Bits	CT	0-177	--
Special Relays	SP	0-777	--
Constant	K	--	1-32

Discrete Bit Flags	Description
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.

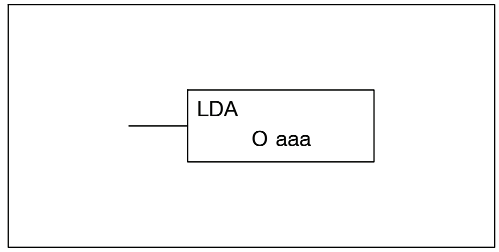


Handheld Programmer Keystrokes

$ STR	→	SHFT	C 2	A 0	ENT	
SHFT	L ANDST	D 3	F 5	→		
SHFT	C 2	B 1	A 0	→	H 7	ENT
GX OUT	SHFT	F 5	→			
C 2	A 0	→	H 7	ENT		

Load Address (LDA)

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 system are in octal.

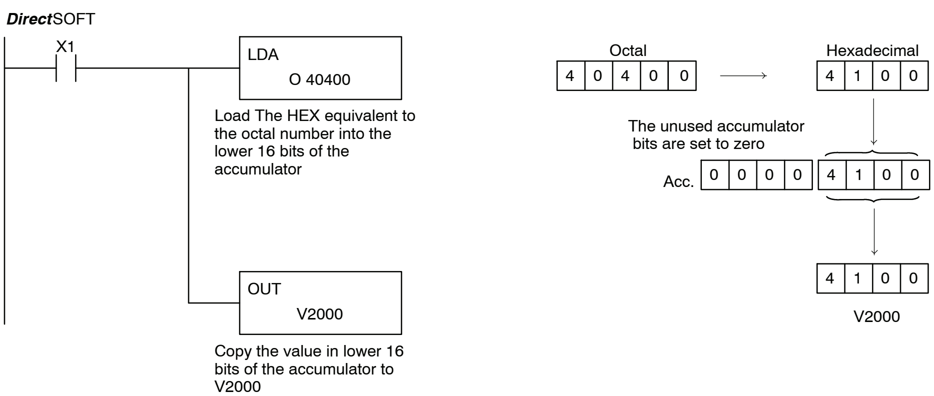


Operand Data Type		DL350 Range
		aaa
Octal Address	O	All V mem. (See page 3-29)

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 V2000 using the Out instruction.

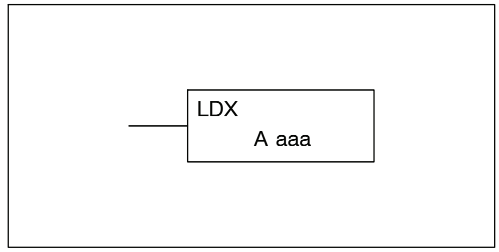


Handheld Programmer Keystrokes

$ STR	→	B 1	ENT					
SHFT	L ANDST	D 3	A 0	→				
E 4	A 0	E 4	A 0	A 0	ENT			
GX OUT	→	SHFT	V AND	C 2	A 0	A 0	A 0	ENT

Load Accumulator Indexed (LDX)

Load Accumulator Indexed is a 16 bit instruction that specifies a source address (V–memory) which will be offset by the value in the first stack location. This instruction interprets the value in the first stack location as HEX. The value in the offset address (source address + offset) is loaded into the lower 16 bits of the accumulator. The upper 16 bits of the accumulator are set to 0.

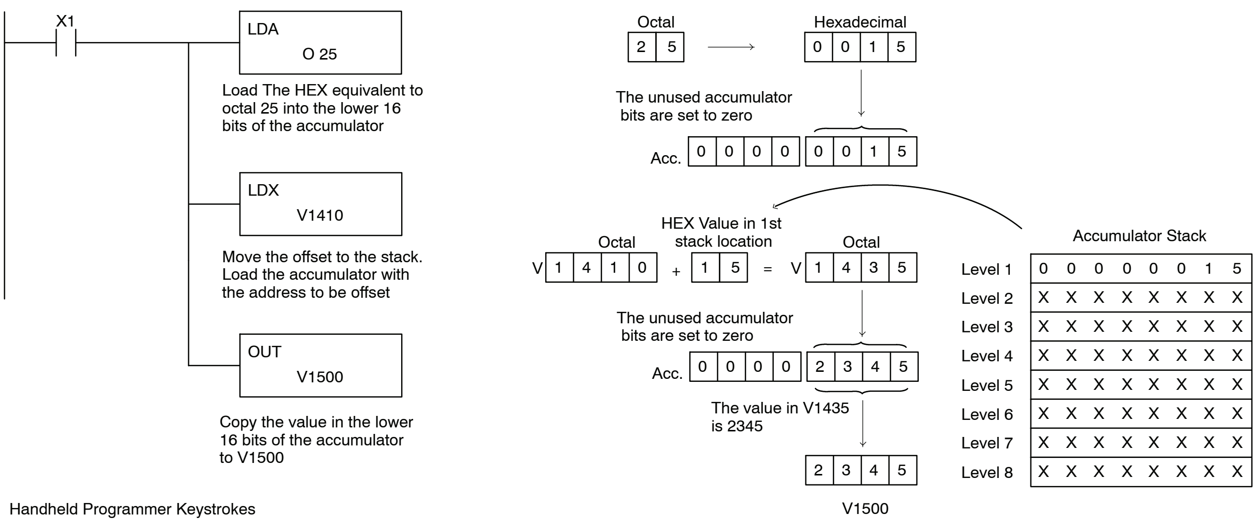


Helpful Hint: — The Load Address instruction can be used to convert an octal address to a HEX address and load the value into the accumulator.

Operand Data Type		DL350 Range
	A	aaa
V–memory	V	All (See p. 3–29)
Pointer	P	All (See p. 3–29)

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.

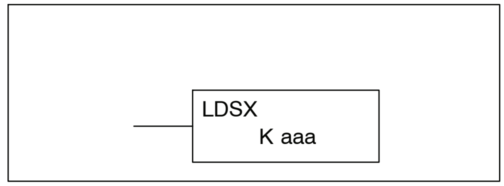


Handheld Programmer Keystrokes

STR	→	X	1	ENT							
SHFT	L	D	A	→	O	2	5	ENT			
SHFT	L	D	X	→	SHFT	V	1	4	1	0	ENT
OUT	→	V	1	5	0	0	ENT				

Load Accumulator Indexed from Data Constants (LDSX)

The Load Accumulator Indexed from Data Constants is a 16 bit instruction. The instruction specifies a Data Label Area (DLBL) where numerical or ASCII constants are stored. This value will be loaded into the lower 16 bits.



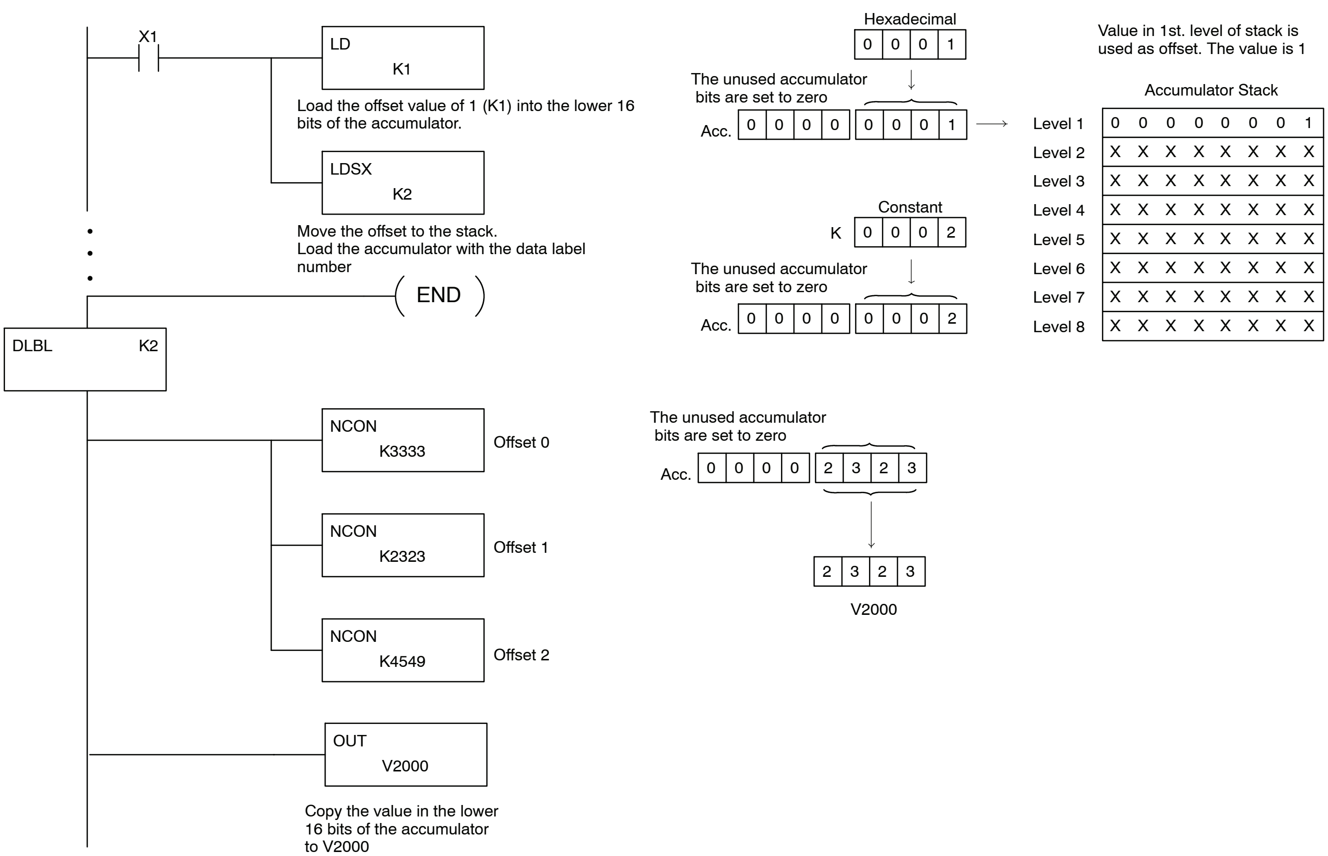
The LDSX instruction uses the value in the first level of the accumulator stack as an offset to determine which numerical or ASCII constant within the Data Label Area will be loaded into the accumulator. The LDSX instruction interprets the value in the first level of the accumulator stack as a HEX value.

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

Operand Data Type		DL350 Range
		aaa
Constant	K	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 offset of 1 is loaded into the accumulator. This value will be placed into the first level of the accumulator stack when the LDSX instruction is executed. The LDSX instruction specifies the Data Label (DLBL K2) where the numerical constant(s) are located in the program and loads the constant value, indicated by the offset in the stack, into the lower 16 bits of the accumulator.

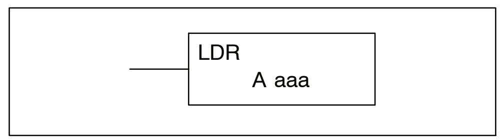


Handheld Programmer Keystrokes

$ STR	→	B 1	ENT							
SHFT	L ANDST	D 3	→	SHFT	K JMP	B 1	ENT			
SHFT	L ANDST	D 3	S RST	X SET	→	C 2	ENT			
SHFT	E 4	N TMR	D 3	ENT						
SHFT	D 3	L ANDST	B 1	L ANDST	→	C 2	ENT			
SHFT	N TMR	C 2	O INST#	N TMR	→	D 3	D 3	D 3	D 3	ENT
SHFT	N TMR	C 2	O INST#	N TMR	→	C 2	D 3	C 2	D 3	ENT
SHFT	N TMR	C 2	O INST#	N TMR	→	E 4	F 5	E 4	J 9	ENT
GX OUT	→	SHFT	V AND	C 2	A 0	A 0	A 0	ENT		

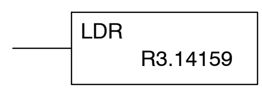
Load Real Number (LDR)

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

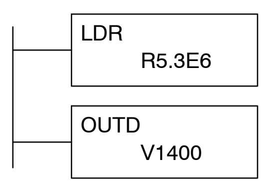


Operand Data Type		DL350 Range
	A	**aaa**
V–memory	V	All V mem (See p. 3–29)
Pointer	P	All V mem (See p. 3–29)
Real Constant	R	Full IEEE 32-bit range

***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”.

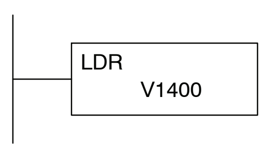


For very large numbers or very small numbers, you can use exponential notation. The number to the right is 5.3 million. The OUTD instruction stores it in V1400 and V1401.



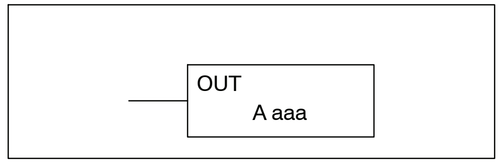
These real numbersare in the IEEE 32-bit floating point format. They occupy two V-memory locations, *regardless of how big or small the number may be!* If you view a stored real number in hex, binary, or even BCD, the number shown will be very difficult to decipher. Like all other number types, you must 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 retreive 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.



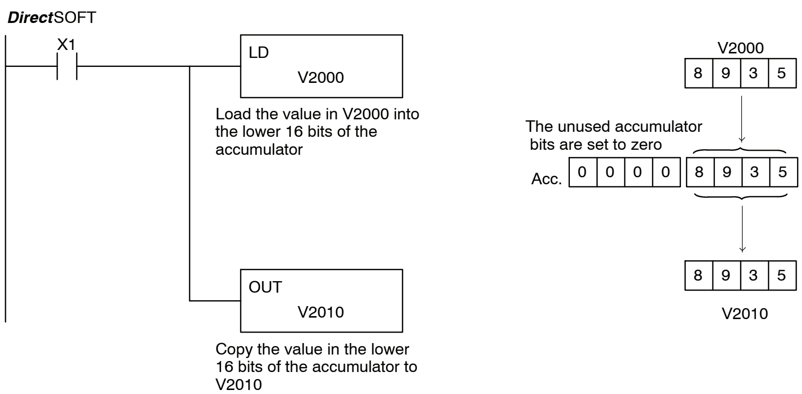
Out (OUT)

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).



Operand Data Type		DL350 Range
	A	aaa
V-memory	V	All (See page 3-29)
Pointer	P	All V mem. (See page 3-29)

In the following example, when X1 is on, the value in V2000 will be loaded into the lower 16 bits of the accumulator using the Load instruction. The value in the lower 16 bits of the accumulator are copied to V2010 using the Out instruction.

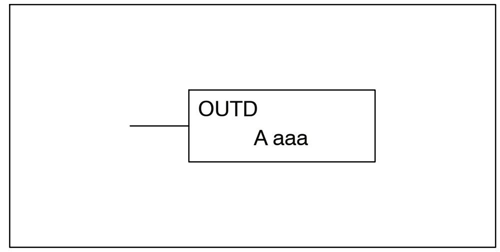


Handheld Programmer Keystrokes

$ STR	→	B 1	ENT					
SHFT	L ANDST	D 3	→					
C 2	A 0	A 0	A 0	ENT				
GX OUT	→	SHFT	V AND	C 2	A 0	B 1	A 0	ENT

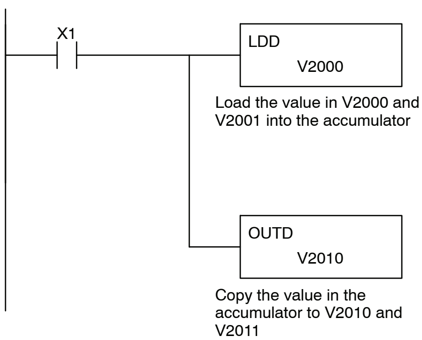
Out DOUBLE (OUTD)

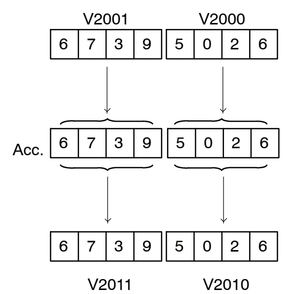
The Out Double instruction is a 32 bit instruction that copies the value in the accumulator to two consecutive V–memory locations at a specified starting location (Aaaa).



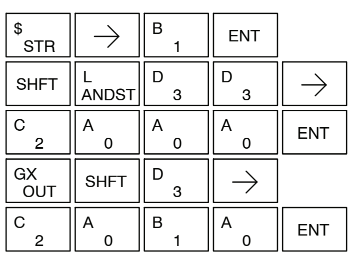
Operand Data Type		DL350 Range
	A	**aaa**
V–memory	V	All (See page 3–29)
Pointer	P	All V mem. (See page 3–29)

In the following example, when X1 is on, the 32 bit value in V2000 and V2001 will be loaded into the accumulator using the Load Double instruction. The value in the accumulator is output to V2010 and V2011 using the Out Double instruction.





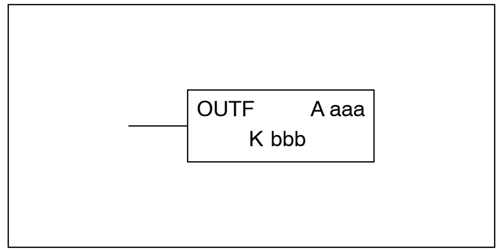
Handheld Programmer Keystrokes



Standard RLL Instructions

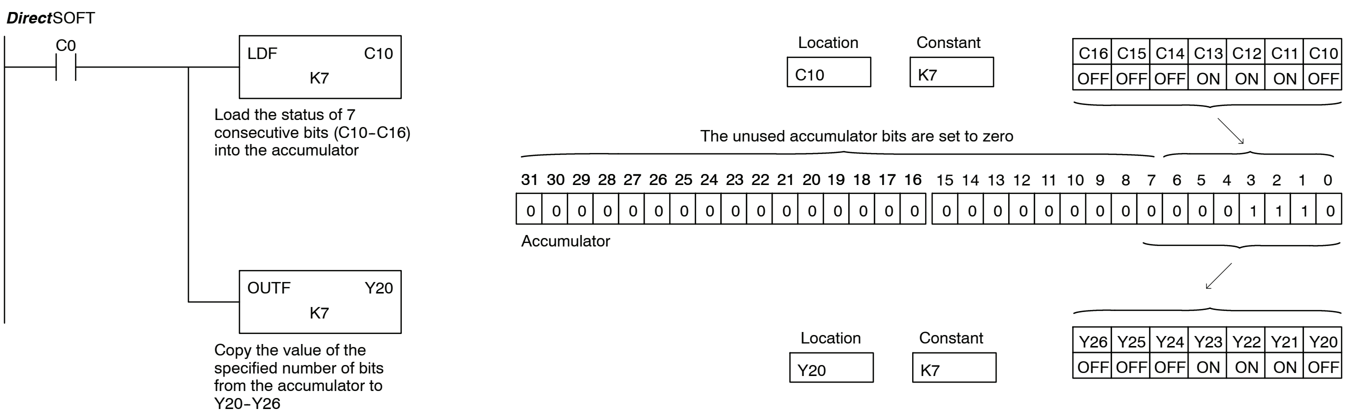
Out Formatted (OUTF)

The Out Formatted instruction outputs 1–32 bits from the accumulator to the specified discrete memory locations. The instruction requires a starting location (Aaaa) for the destination and the number of bits (Kbbb) to be output.

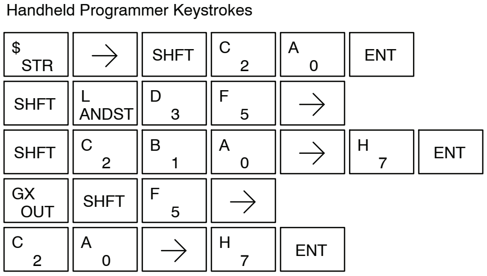


Operand Data Type	A	DL350 Range	
	A	aaa	bbb
Inputs	X	0–777	--
Outputs	Y	0–777	--
Control Relays	C	0–1777	--
Constant	K	--	1–32

In the following example, when C0 is on, the binary pattern of C10–C16 (7 bits) will be loaded into the accumulator using the Load Formatted instruction. The lower 7 bits of the accumulator are output to Y20–Y26 using the Out Formatted instruction.



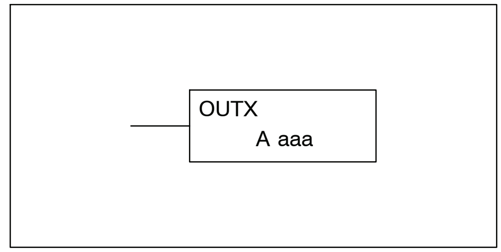
Handheld Programmer Keystrokes



Standard RLL Instructions

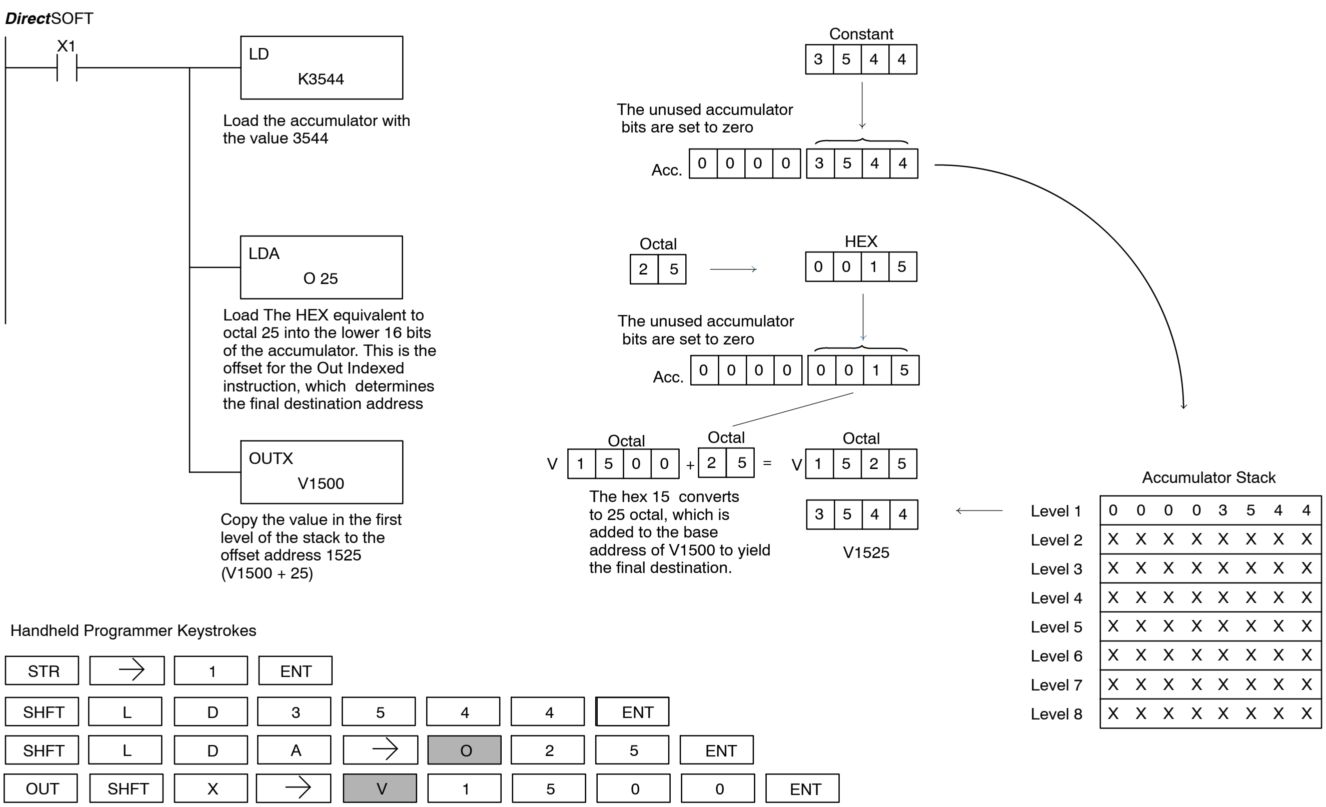
Out Indexed (OUTX)

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.



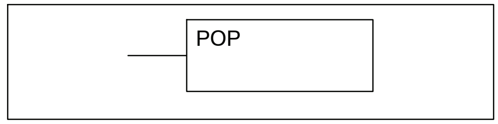
Operand Data Type		DL350 Range
	A	aaa
V–memory	V	All (See p. 3–29)
Pointer	P	All (See p. 3–29)

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.



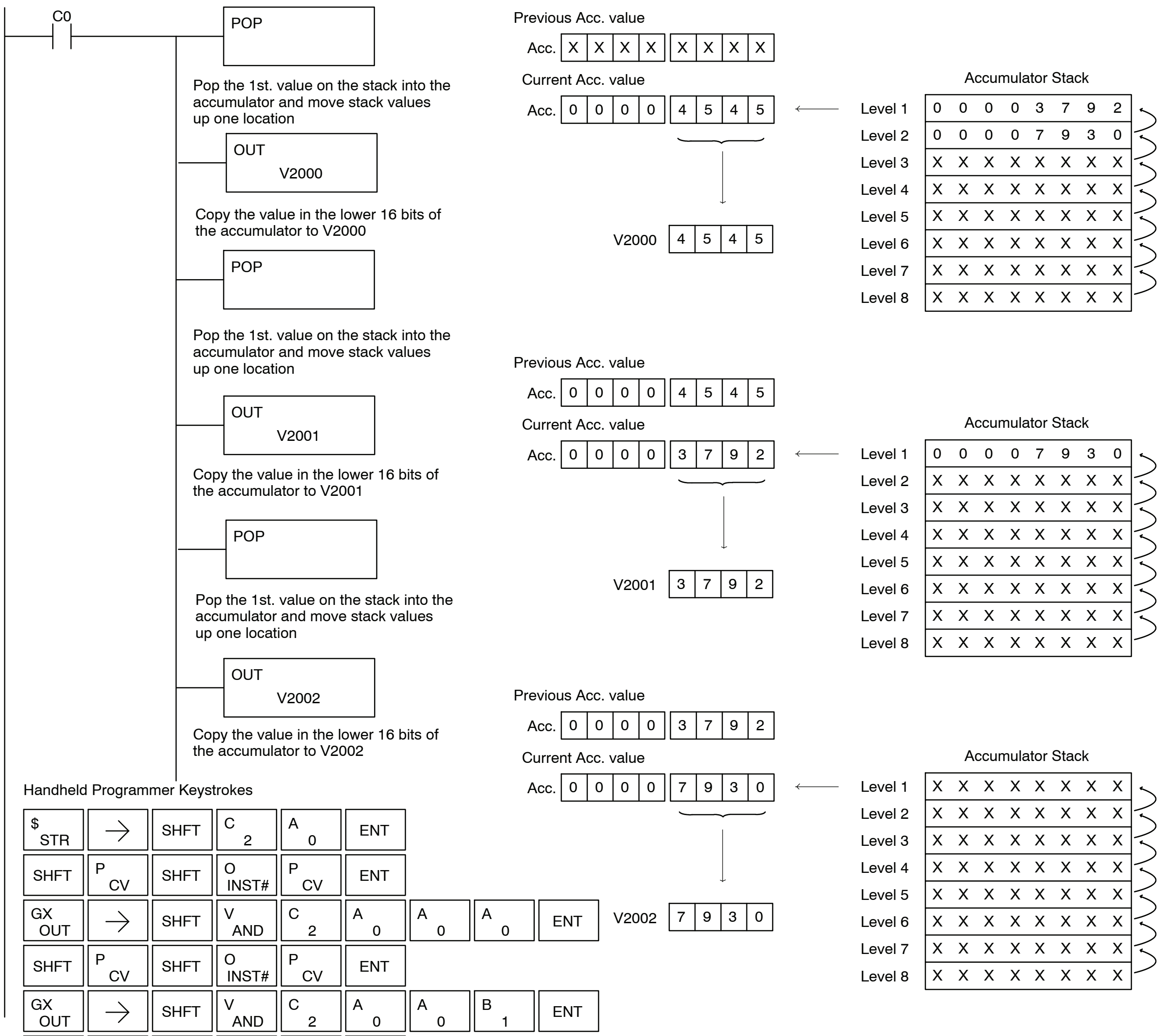
Pop (POP)

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.



In the example, when C0 is on, the value 4545 that was on top of the stack is moved into the accumulator using the Pop instruction The value is output to V2000 using the Out instruction. The next Pop moves the value 3792 into the accumulator and outputs the value to V2001. The last Pop moves the value 7930 into the accumulator and outputs the value to V2002. Please note if the value in the stack were greater than 16 bits (4 digits) the Out Double instruction would be used and two V–memory locations for each Out Double need to be allocated.

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



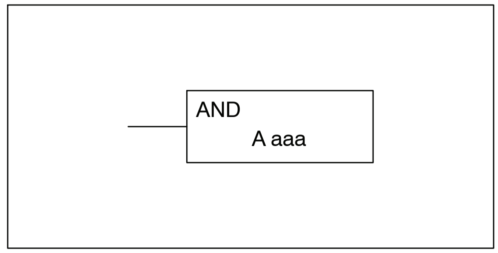
Handheld Programmer Keystrokes

$ STR	→	SHFT	C 2	A 0	ENT			
SHFT	P CV	SHFT	O INST#	P CV	ENT			
GX OUT	→	SHFT	V AND	C 2	A 0	A 0	A 0	ENT
SHFT	P CV	SHFT	O INST#	P CV	ENT			
GX OUT	→	SHFT	V AND	C 2	A 0	A 0	B 1	ENT
SHFT	P CV	SHFT	O INST#	P CV	ENT			
GX OUT	→	SHFT	V AND	C 2	A 0	A 0	C 2	ENT

Accumulator Logical Instructions

And (AND)

The And instruction is a 16 bit instruction that logically ands the value in the lower 16 bits of the accumulator with a specified V-memory location (Aaaa). The result resides in the accumulator. The discrete status flag indicates if the result of the And is zero.

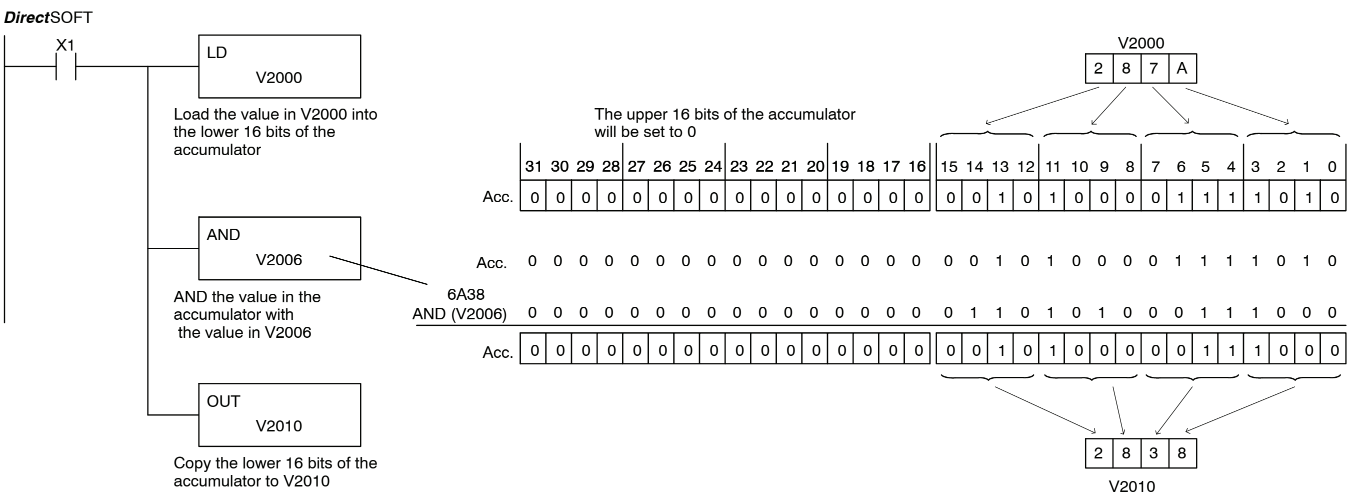


Operand Data Type		DL3540 Range
	A	aaa
V-memory	V	All (See page 3-29)
Pointer	P	All V mem. (See page 3-29)

Discrete Bit Flags	Description
SP63	Will be on if the result in 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 V2000 will be loaded into the accumulator using the Load instruction. The value in the accumulator is anded with the value in V2006 using the And instruction. The value in the lower 16 bits of the accumulator is output to V2010 using the Out instruction.

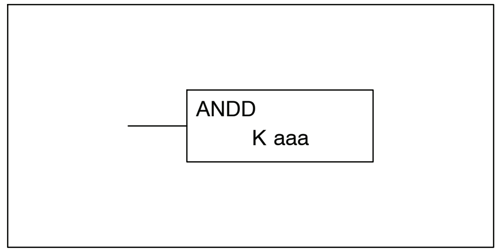


Handheld Programmer Keystrokes

$ STR	→	B 1	ENT					
SHFT	L ANDST	D 3	→	C 2	A 0	A 0	A 0	ENT
V AND	→	SHFT	V AND	C 2	A 0	A 0	G 6	ENT
GX OUT	→	SHFT	V AND	C 2	A 0	B 1	A 0	ENT

And Double (ANDD)

The And Double is a 32 bit instruction that logically ands the value in the accumulator with 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).

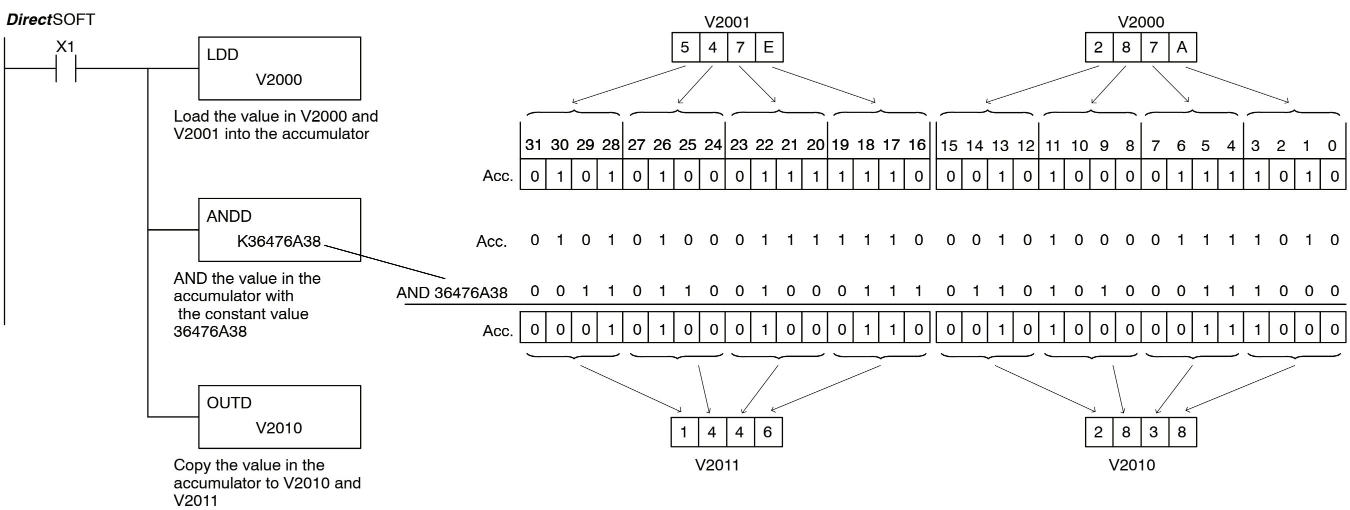


Operand Data Type		DL350 Range
		aaa
Constant	K	0–FFFF

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: The status flags are only valid until another instruction that uses the same flags is executed.

In the following example, when X1 is on, the value in V2000 and V2001 will be loaded into the accumulator using the Load Double instruction. The value in the accumulator is anded with 36476A38 using the And double instruction. The value in the accumulator is output to V2010 and V2011 using the Out Double instruction.

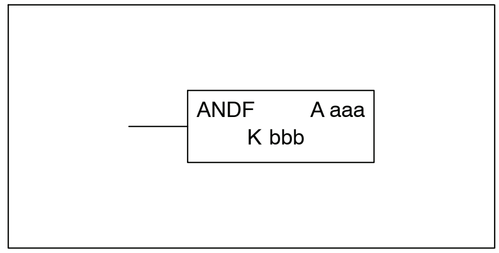


Handheld Programmer Keystrokes

$ STR	→	B 1	ENT													
SHFT	L ANDST	D 3	D 3	→	C 2	A 0	A 0	A 0	ENT							
V AND	SHFT	D 3	→	SHFT	K JMP	D 3	G 6	E 4	H 7	G 6	SHFT	A 0	SHFT	D 3	I 8	ENT
GX OUT	SHFT	D 3	→	C 2	A 0	B 1	A 0	ENT								

And Formatted (ANDF)

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).

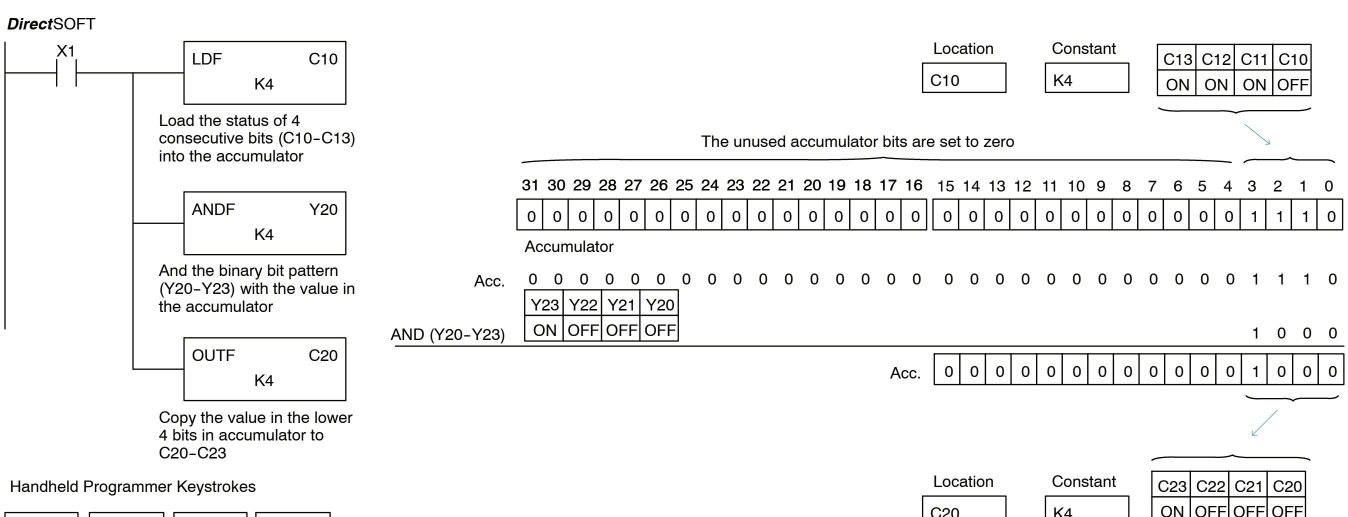


Operand Data Type	A/B	DL350 Range aaa	DL350 Range bbb
Inputs	X	0–777	--
Outputs	Y	0–777	--
Control Relays	C	0–1777	--
Stage Bits	S	0–1777	--
Timer Bits	T	0–377	--
Counter Bits	CT	0–177	--
Special Relays	SP	0–777	--
Constant	K	--	1–32

Discrete Bit Flags	Description
SP63	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.

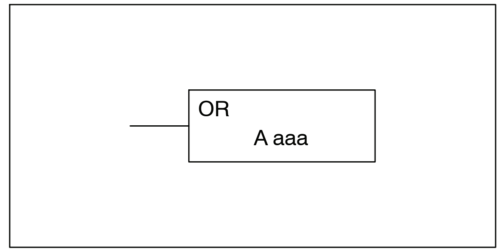


Handheld Programmer Keystrokes

STR	→	1	ENT								
SHFT	L	D	F	→	C	1	0	→	K	4	ENT
AND	SHFT	F	→	Y	2	0	→	K	4	ENT	
OUT	SHFT	F	→	C	2	0	→	K	4	ENT	

Or
(OR)

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

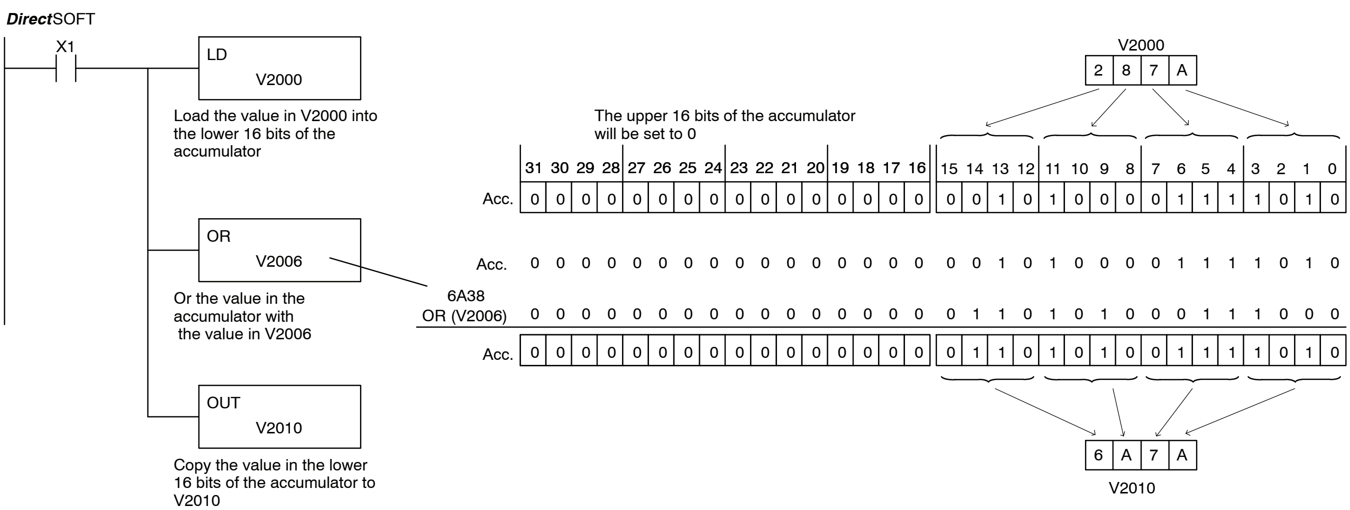


Operand Data Type		DL350 Range
	A	aaa
V–memory	V	All (See page 3–29)
Pointer	P	All V mem. (See page 3–29)

Discrete Bit Flags	Description
SP63	Will be on if the result in 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 V2000 will be loaded into the accumulator using the Load instruction. The value in the accumulator is ored with V2006 using the Or instruction. The value in the lower 16 bits of the accumulator are output to V2010 using the Out instruction.

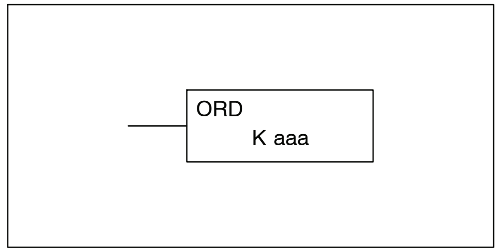


Handheld Programmer Keystrokes

$ STR	→	B 1	ENT					
SHFT	L ANDST	D 3	→	C 2	A 0	A 0	A 0	ENT
Q OR	→	SHFT	V AND	C 2	A 0	A 0	G 6	ENT
GX OUT	→	SHFT	V AND	C 2	A 0	B 1	A 0	ENT

Or Double (ORD)

The Or Double is a 32 bit instruction that ors the value in the accumulator with 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).

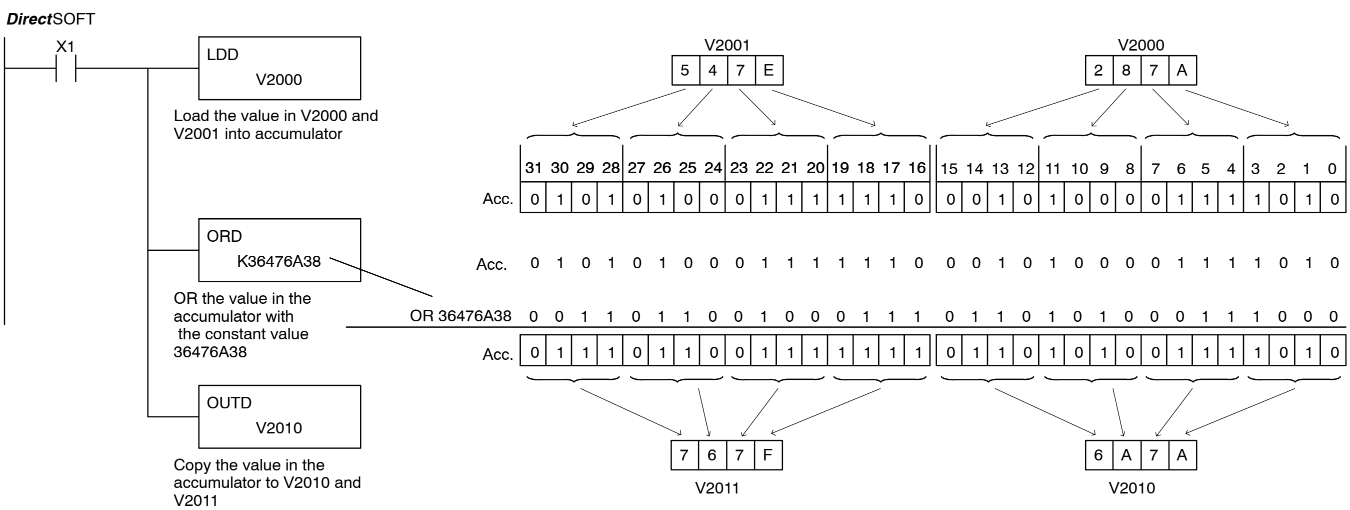


Operand Data Type		DL350 Range
	A	**aaa**
Constant	K	0–FFFF

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: The status flags are only valid until another instruction that uses the same flags is executed.

In the following example, when X1 is on, the value in V2000 and V2001 will be loaded into the accumulator using the Load Double instruction. The value in the accumulator is ored with 36476A38 using the Or Double instruction. The value in the accumulator is output to V2010 and V2011 using the Out Double instruction.

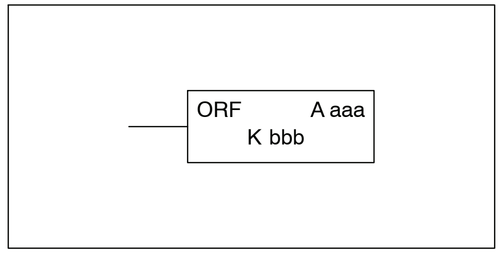


Handheld Programmer Keystrokes

$ STR	→	B 1	ENT													
SHFT	L ANDST	D 3	D 3	→	C 2	A 0	A 0	A 0	ENT							
Q OR	SHFT	D 3	→	SHFT	K JMP	D 3	G 6	E 4	H 7	G 6	SHFT	A 0	SHFT	D 3	I 8	ENT
GX OUT	SHFT	D 3	→	C 2	A 0	B 1	A 0	ENT								

Or Formatted (ORF)

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).

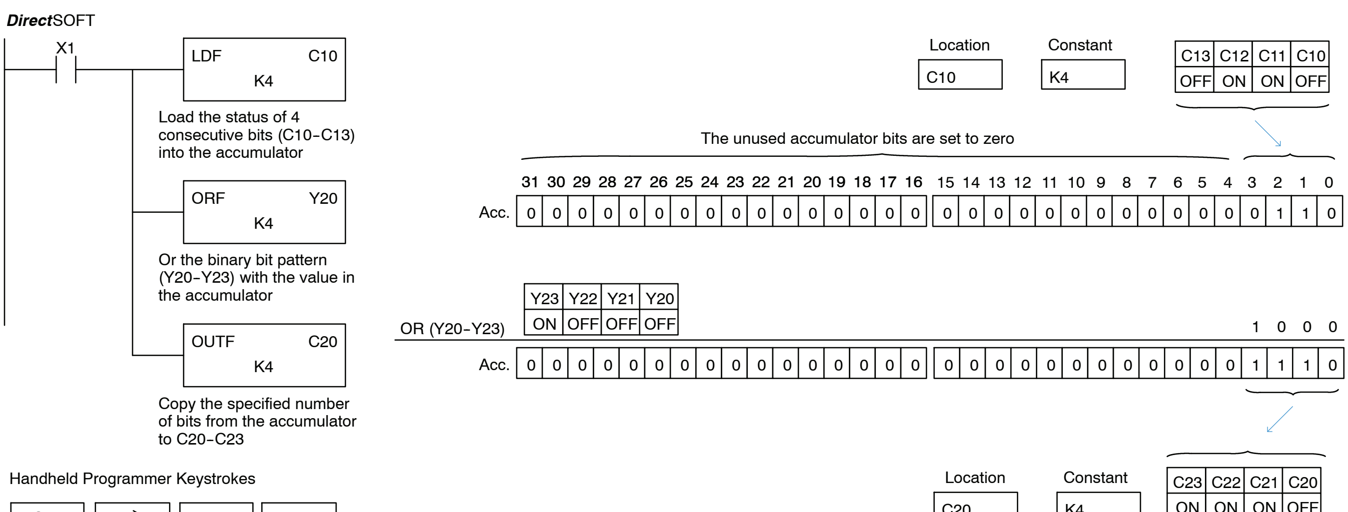


Operand Data Type	A/B	DL350 Range aaa	bbb
Inputs	X	0–777	--
Outputs	Y	0–777	--
Control Relays	C	0–1777	--
Stage Bits	S	0–1777	--
Timer Bits	T	0–377	--
Counter Bits	CT	0–177	--
Special Relays	SP	0–777	--
Constant	K	--	1–32

Discrete Bit Flags	Description
SP63	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.

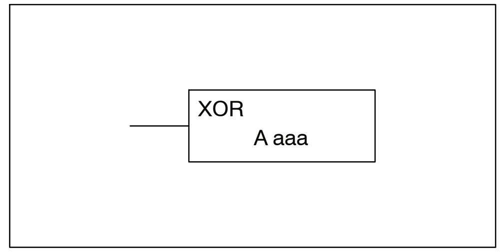


Handheld Programmer Keystrokes

STR	→	1	ENT								
SHFT	L	D	F	→	C	1	0	→	K	4	ENT
OR	SHFT	F	→	Y	2	0	→	K	4	ENT	
OUT	SHFT	F	→	C	2	0	→	K	4	ENT	

Exclusive Or (XOR)

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

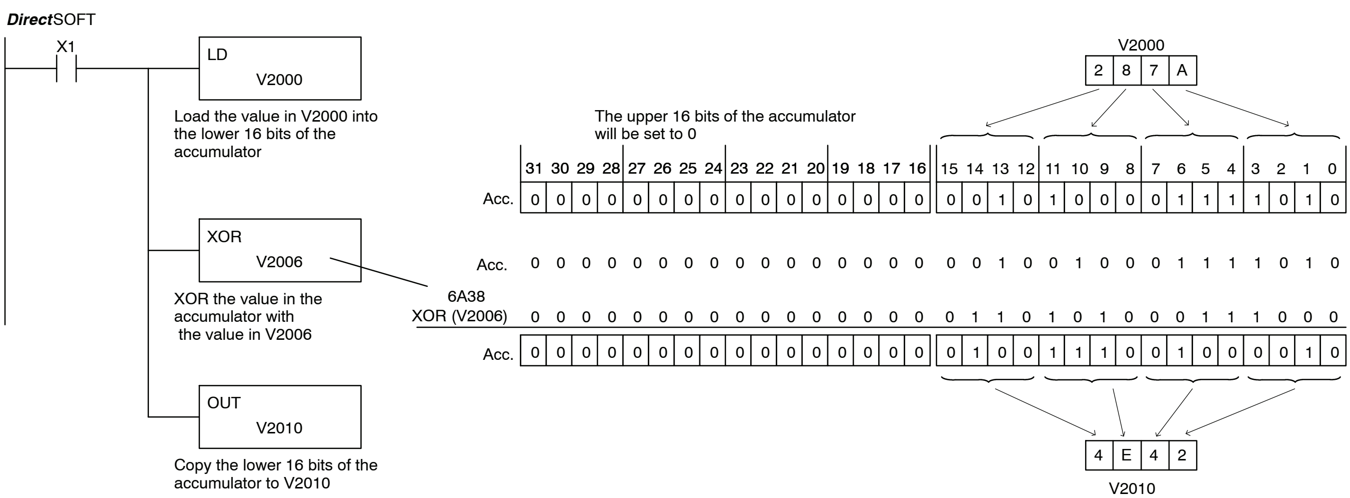


Operand Data Type	A	DL350 Range aaa
V–memory	V	All (See page 3–29)
Pointer	P	All V mem. (See page 3–29)

Discrete Bit Flags	Description
SP63	Will be on if the result in 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 V2000 will be loaded into the accumulator using the Load instruction. The value in the accumulator is exclusive ored with V2006 using the Exclusive Or instruction. The value in the lower 16 bits of the accumulator are output to V2010 using the Out instruction.

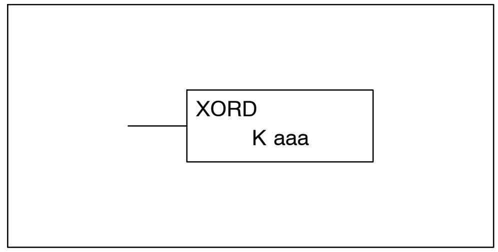


Handheld Programmer Keystrokes

$ STR	→	SHFT	X SET	B 1	ENT						
SHFT	L ANDST	D 3	→	SHFT	V AND	C 2	A 0	A 0	A 0	ENT	
SHFT	X SET	SHFT	Q OR	→	SHFT	V AND	C 2	A 0	A 0	G 6	ENT
GX OUT	→	SHFT	V AND	C 2	A 0	B 1	A 0	ENT			

Exclusive Or Double (XORD)

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 a 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).

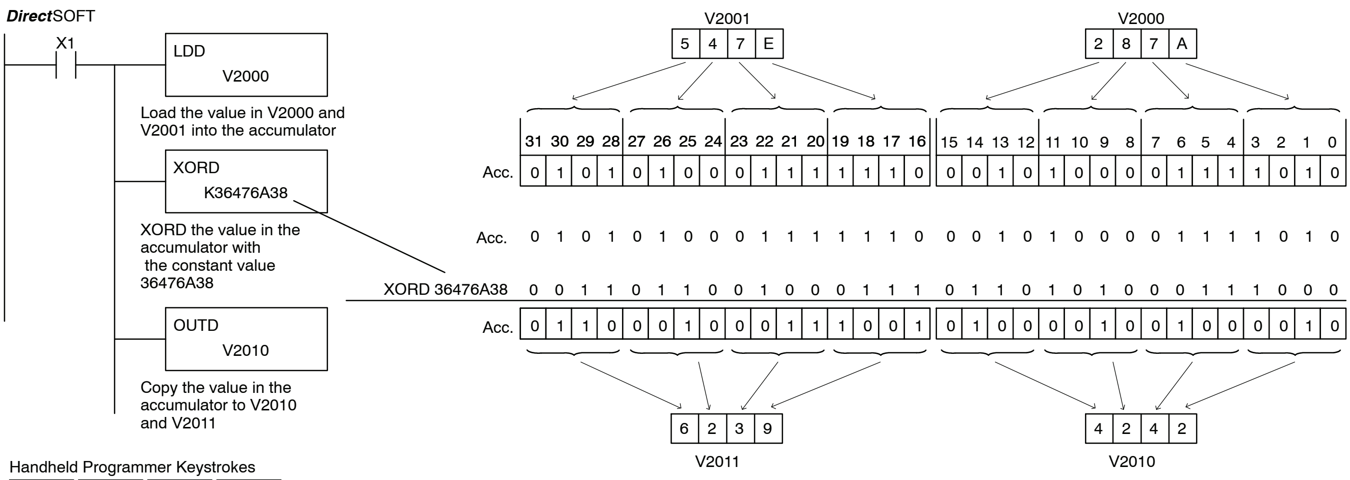


Operand Data Type		DL350 Range
	A	aaa
Constant	K	0–FFFF

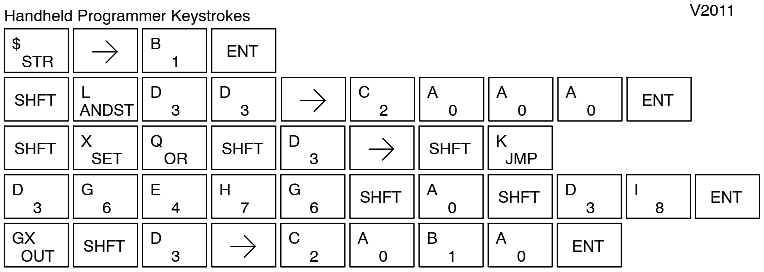
Discrete Bit Flags	Description
SP63	Will be on if the result in the accumulator is zero
SP70	Will be on is the result in the accumulator is negative

NOTE: The status flags are only valid until another instruction that uses the same flags is executed.

In the following example, when X1 is on, the value in V2000 and V2001 will be loaded into the accumulator using the Load Double instruction. The value in the accumulator is exclusively ored with 36476A38 using the Exclusive Or Double instruction. The value in the accumulator is output to V2010 and V2011 using the Out Double instruction.

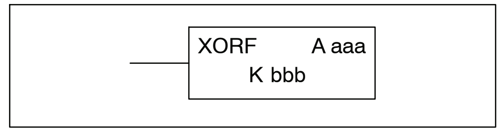


Handheld Programmer Keystrokes



Exclusive Or Formatted (XORF)

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).



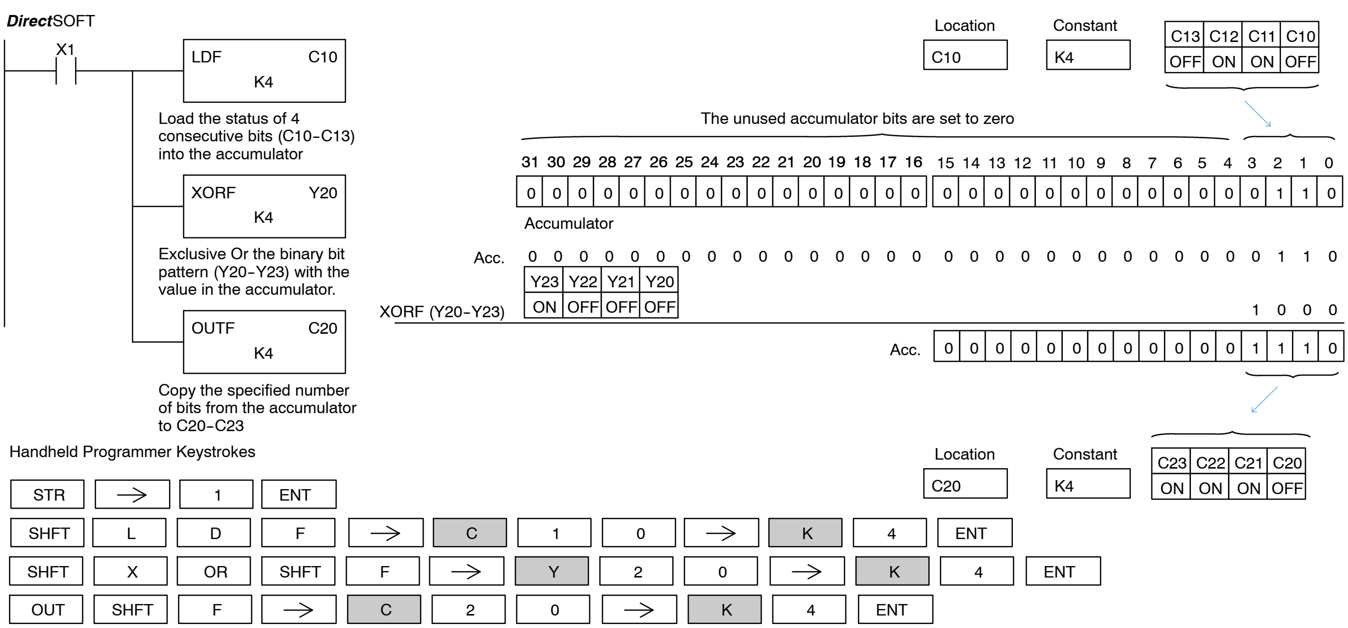
The instruction requires a starting location (Aaaa) and the number of bits (Bbbb) to be exclusive ORed. Discrete status flags indicate if the result of the Exclusive Or Formatted is zero or negative (the most significant bit =1).

Operand Data Type		DL350 Range	
	A/B	aaa	bbb
Inputs	X	0–777	--
Outputs	Y	0–777	--
Control Relays	C	0–1777	--
Stage Bits	S	0–1777	--
Timer Bits	T	0–377	--
Counter Bits	CT	0–177	--
Special Relays	SP	0–777	--
Constant	K	--	1–32

Discrete Bit Flags	Description
SP63	Will be on if the result in the accumulator is zero
SP70	Will be on is the result in the accumulator is negative

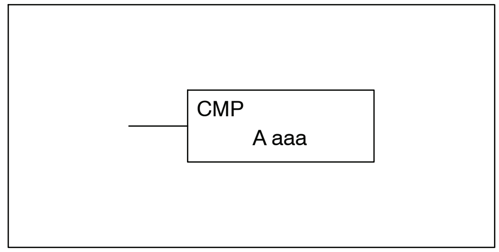
NOTE: Status flags are valid only until another instruction uses the same flag.

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 Ored with the bit pattern from Y20–Y23 using the Exclusive Or Formatted instruction. The value in the lower 4 bits of the accumulator are output to C20–C23 using the Out Formatted instruction.



Compare (CMP)

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.



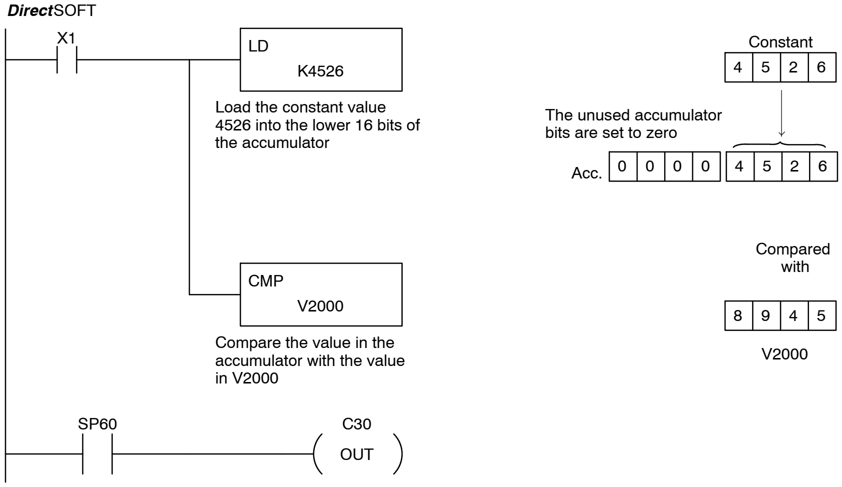
Operand Data Type		DL350 Range
	A	aaa
V–memory	V	All (See page 3–29)
Pointer	P	All V mem. (See page 3–29)

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: The status flags are updated immediately after the instruction is carried out during the scan of the CPU, therefore, it is only valid until another instruction that uses the same flags is executed.

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 V2000 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.

*Direct*SOFT

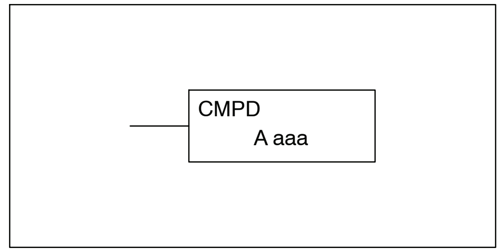


Handheld Programmer Keystrokes

$ STR	→	B 1	ENT							
SHFT	L ANDST	D 3	→	SHFT	K JMP	E 4	F 5	C 2	G 6	ENT
SHFT	C 2	SHFT	M ORST	P CV	→	C 2	A 0	A 0	A 0	ENT
$ STR	→	SHFT	SP STRN	G 6	A 0	ENT				
GX OUT	→	SHFT	C 2	D 3	A 0	ENT				

Compare Double (CMPD)

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.

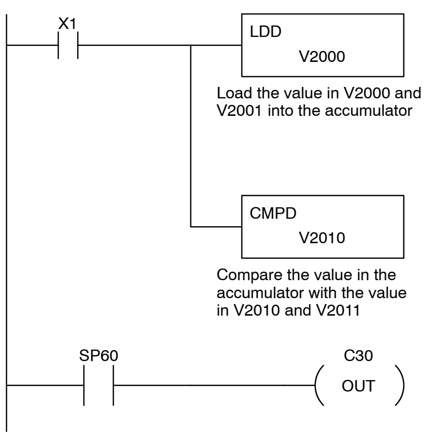


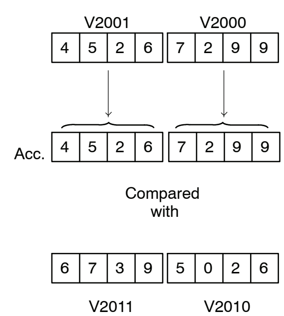
Operand Data Type		DL350 Range
	A	aaa
V–memory	V	All (See page3–29)
Pointer	P	All V mem. (See page 3–29)
Constant	K	1–FFFFFFFF

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: The status flags are updated immediately after the instruction is carried out during the scan of the CPU, therefore, it is only valid until another instruction that uses the same flags is executed.

In the following example when X1 is on, the value in V2000 and V2001 will be loaded into the accumulator using the Load Double instruction. The value in the accumulator is compared with the value in V2010 and V2011 using the CMPD instruction. The corresponding discrete status flag will be turned on indicating the result of the comparison. In this example, if the value in the accumulator is less than the value specified in the Compare instruction, SP60 will turn on energizing C30.



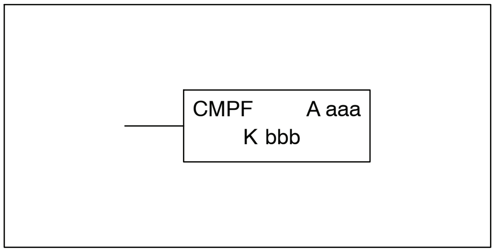


Handheld Programmer Keystrokes

$ STR	→	B 1	ENT								
SHFT	L ANDST	D 3	D 3	→	C 2	A 0	A 0	A 0	ENT		
SHFT	C 2	SHFT	M ORST	P CV	D 3	→	C 2	A 0	B 1	A 0	ENT
$ STR	→	SHFT	SP STRN	G 6	A 0	ENT					
GX OUT	→	SHFT	C 2	D 3	A 0	ENT					

Compare Formatted (CMPF)

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.



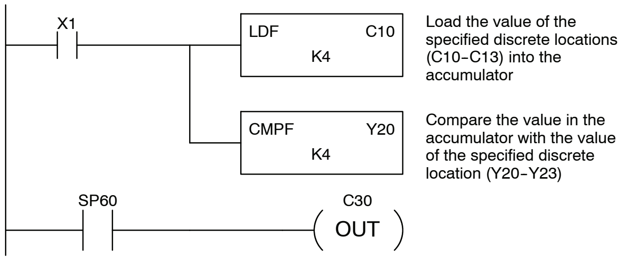
Operand Data Type		DL350 Range	
	A/B	aaa	bbb
Inputs	X	0–777	--
Outputs	Y	0–777	--
Control Relays	C	0–1777	--
Stage Bits	S	0–1777	--
Timer Bits	T	0–377	--
Counter Bits	CT	0–177	--
Special Relays	SP	0–777	--
Constant	K	--	1–32

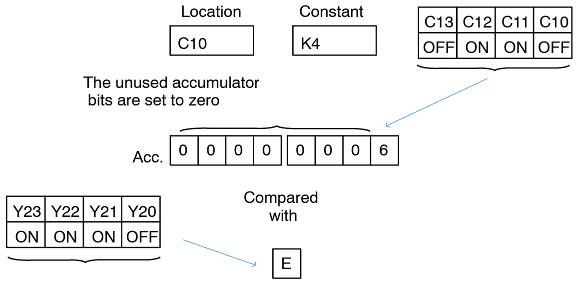
Discrete Bit Flags	Description
SP60	On when the value in the accumulator is less than the instruction value.
SP61	On when the value in the accumulator is equal to the instruction value.
SP62	On when the value in the accumulator is greater than the instruction value.

NOTE: Status flags are valid only until another instruction uses the same flag.

In the following example, when X1 is on the Load Formatted instruction loads the binary value (6) from C10–C13 into the accumulator. The CMPF instruction compares the value in the accumulator to the value in Y20–Y23 (E hex). The corresponding discrete status flag will be turned on indicating the result of the comparison. In this example, if the value in the accumulator is less than the value specified in the Compare instruction, SP60 will turn on energizing C30.

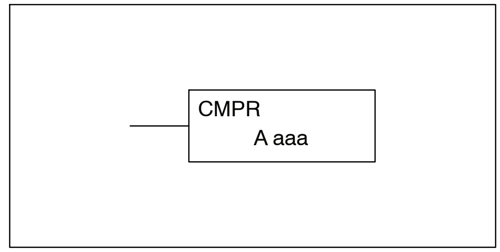
*Direct*SOFT





Compare Real Number (CMPR)

The Compare Real Number instruction compares a real number value in the accumulator with two consecutive V–memory locations containing a real number. The corresponding status flag will be turned on indicating the result of the comparison. Both numbers being compared are 32 bits long.

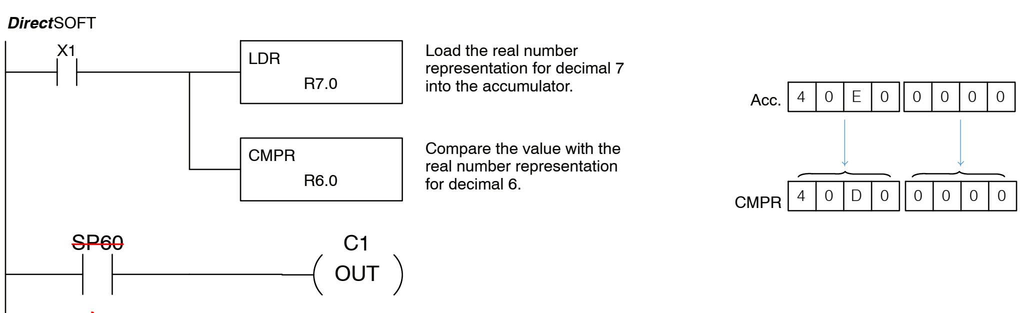


Operand Data Type		DL350 Range
	A	aaa
V–memory	V	All (See p. 3–29)
Pointer	P	All (See p. 3–29)
Constant	R	–3.402823E+038 to + –3.402823E+038

Discrete Bit Flags	Description
SP60	On when the value in the accumulator is less than the instruction value.
SP61	On when the value in the accumulator is equal to the instruction value.
SP62	On when the value in the accumulator is greater than the instruction value.
SP71	On anytime the V-memory specified by a pointer (P) is not valid.
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 SP60).

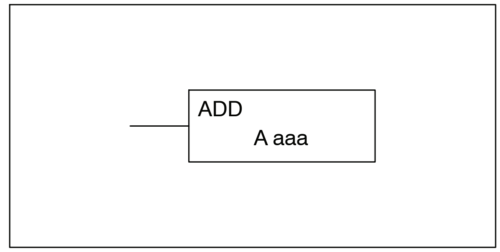


Incorrect. Should be SP62

Math Instructions

Add (ADD)

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. You cannot use a constant as the parameter in the box.

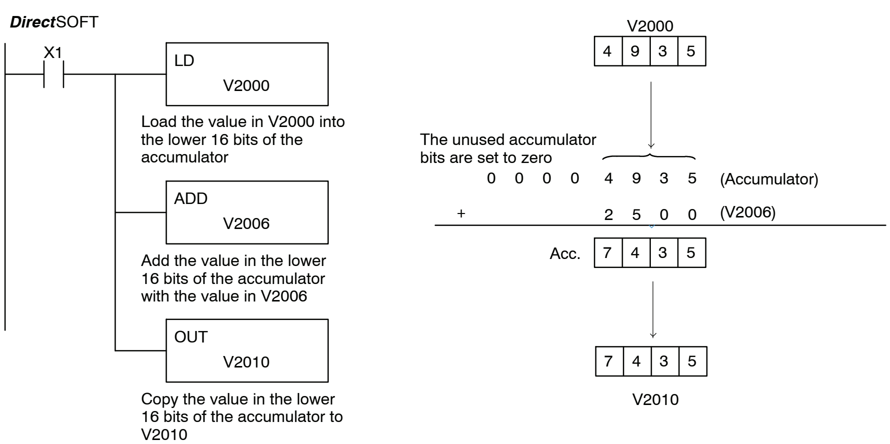


Operand Data Type		DL350 Range
	A	aaa
V–memory	V	All (See page 3–29)
Pointer	P	All V mem. (See page 3–29)

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: The status flags are only valid until another instruction that uses the same flags is executed.

In the following example, when X1 is on, the value in V2000 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 V2006 using the Add instruction. The value in the accumulator is copied to V2010 using the Out instruction.

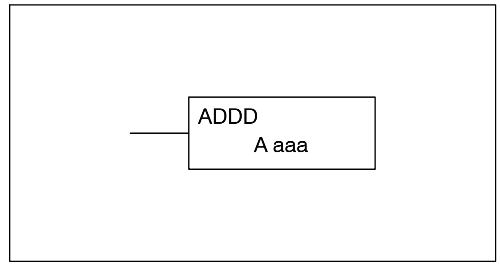


Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	→	C 2	A 0	A 0	A 0	ENT	
SHFT	A 0	D 3	D 3	→	C 2	A 0	A 0	G 6	ENT
GX OUT	→	SHFT	V AND	C 2	A 0	B 1	A 0	ENT	

Add Double (ADDD)

Add Double is a 32 bit instruction that adds the BCD value in the accumulator with a BCD value (Aaaa), which is either two consecutive V-memory locations or an 8-digit (max.) BCD constant. The result resides in the accumulator.

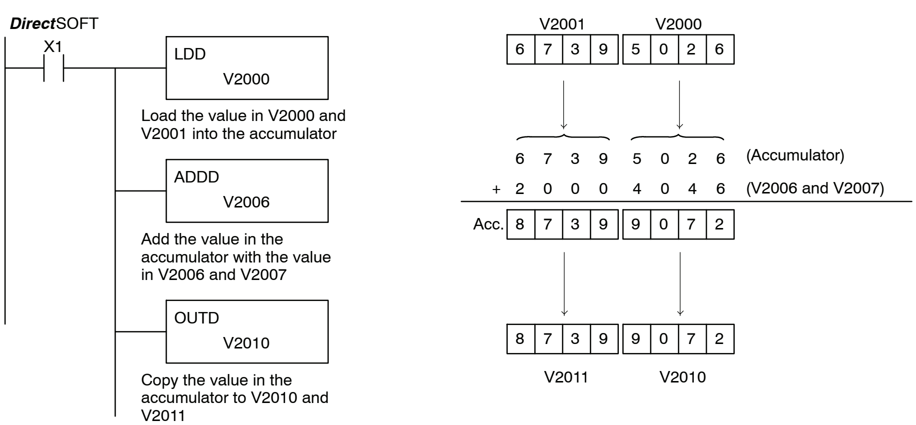


Operand Data Type		DL350 Range
	A	aaa
V-memory	V	All (See page 3-29)
Pointer	P	All V mem. (See page 3-29)
Constant	K	0-99999999

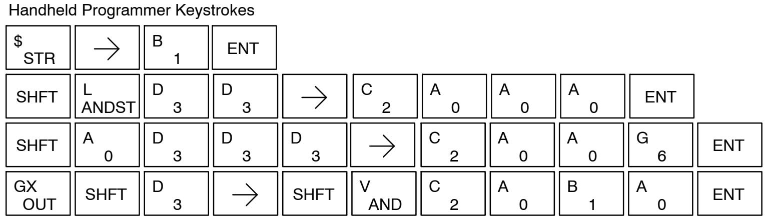
Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP66	On when the 16 bit addition instruction results in a carry.
SP67	On when the 32 bit addition instruction results in a carry.
SP70	On anytime the value in the accumulator is negative.
SP75	On when a BCD instruction is executed and a NON-BCD number was encountered.

NOTE: The status flags are only valid until another instruction that uses the same flags is executed.

In the following example, when X1 is on, the value in V2000 and V2001 will be loaded into the accumulator using the Load Double instruction. The value in the accumulator is added with the value in V2006 and V2007 using the Add Double instruction. The value in the accumulator is copied to V2010 and V2011 using the Out Double instruction.

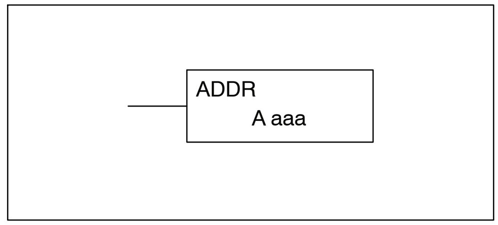


Handheld Programmer Keystrokes



Add Real (ADDR)

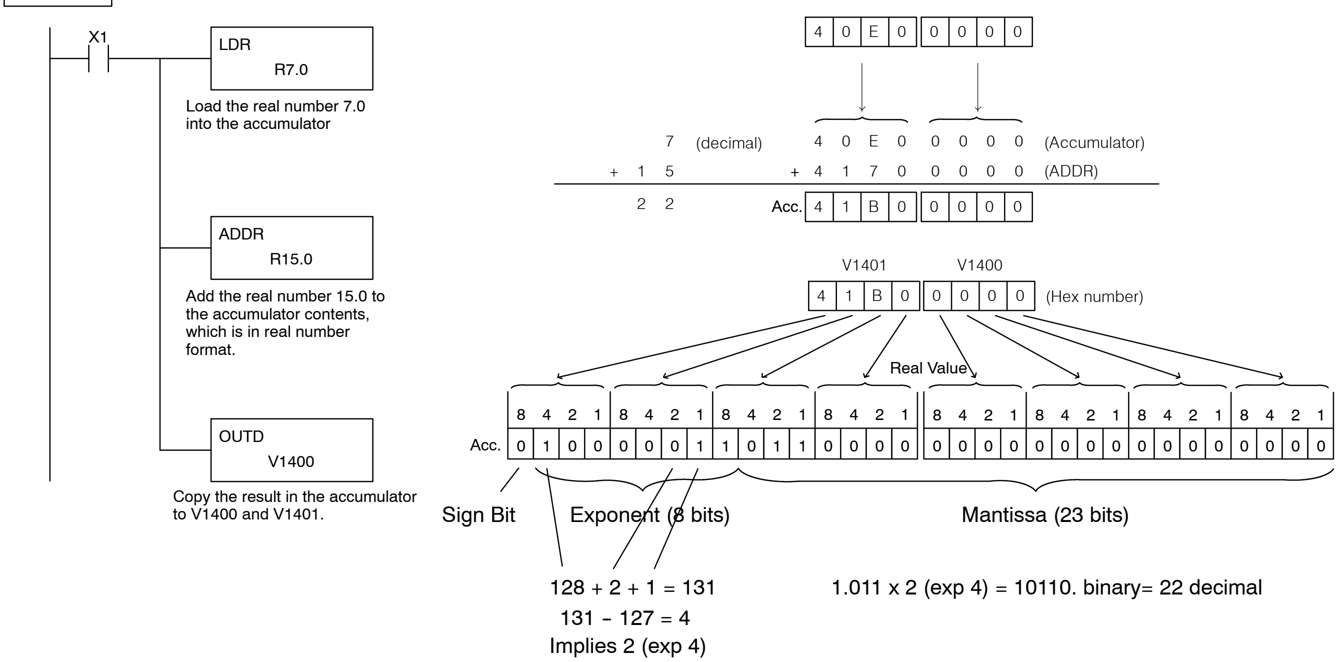
Add Real is a 32–bit instruction that adds a real number, which is either two consecutive V–memory locations or a 32–bit constant, to a real number in the accumulator. Both numbers must conform to the IEEE floating point format. The result is a 32–bit real number that resides in the accumulator.



Operand Data Type		DL350 Range
	A	aaa
V–memory	V	All (See p. 3–29)
Pointer	P	All V mem (See p. 3–29)
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 an invalid floating point number.
SP73	on when a signed addition or subtraction results in a incorrect sign bit.
SP74	On anytime a floating point math operation results in an underflow error.
SP75	On when a real number instruction is executed and a non-real number was encountered.

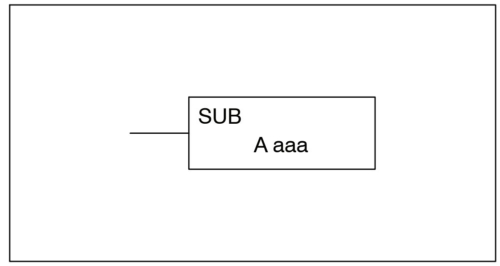
NOTE: Status flags are valid only until another instruction uses the same flag.



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)

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. You cannot use a constant as the parameter in the box.

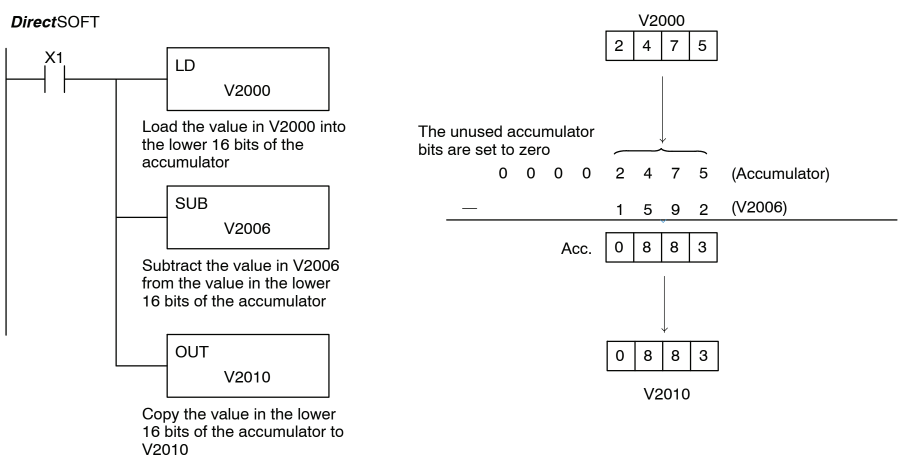


Operand Data Type		DL350 Range
	A	aaa
V–memory	V	All (See page 3–29)
Pointer	P	All V mem. (See page 3–29)

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: The status flags are only valid until another instruction that uses the same flags is executed.

In the following example, when X1 is on, the value in V2000 will be loaded into the accumulator using the Load instruction. The value in V2006 is subtracted from the value in the accumulator using the Subtract instruction. The value in the accumulator is copied to V2010 using the Out instruction.

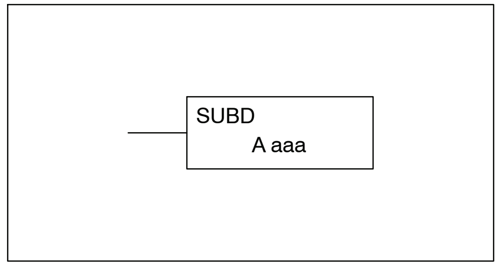


Handheld Programmer Keystrokes

$ STR	→	B 1	ENT								
SHFT	L ANDST	D 3	→	C 2	A 0	A 0	A 0	ENT			
SHFT	S RST	U ISG	B 1	→	SHFT	V AND	C 2	A 0	A 0	G 6	ENT
GX OUT	→	SHFT	V AND	C 2	A 0	B 1	A 0	ENT			

Subtract Double (SUBD)

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.

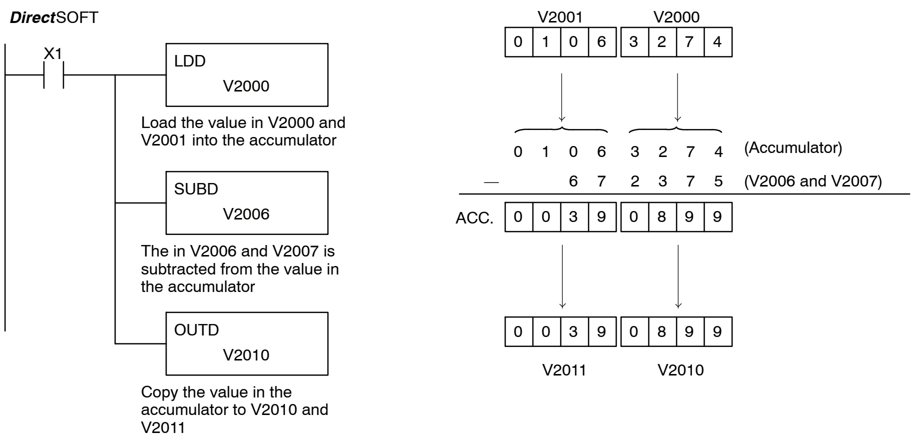


Operand Data Type		DL350 Range
	A	aaa
V–memory	V	All (See page 3–29)
Pointer	P	All V mem. (See page 3–29)
Constant	K	0–99999999

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP64	On when the 16 bit subtraction instruction results in a borrow.
SP65	On when the 32 bit subtraction instruction results in a borrow.
SP70	On anytime the value in the accumulator is negative.
SP75	On when a BCD instruction is executed and a NON–BCD number was encountered.

NOTE: The status flags are only valid until another instruction that uses the same flags is executed.

In the following example, when X1 is on, the value in V2000 and V2001 will be loaded into the accumulator using the Load Double instruction. The value in V2006 and V2007 is subtracted from the value in the accumulator. The value in the accumulator is copied to V2010 and V2011 using the Out Double instruction.

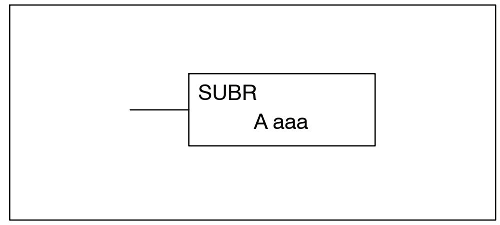


Handheld Programmer Keystrokes

$ STR	→	B 1	ENT								
SHFT	L ANDST	D 3	D 3	→	C 2	A 0	A 0	A 0	ENT		
SHFT	S RST	SHFT	U ISG	B 1	D 3	→	C 2	A 0	A 0	G 6	ENT
GX OUT	SHFT	D 3	→	C 2	A 0	B 1	A 0	ENT			

Subtract Real (SUBR)

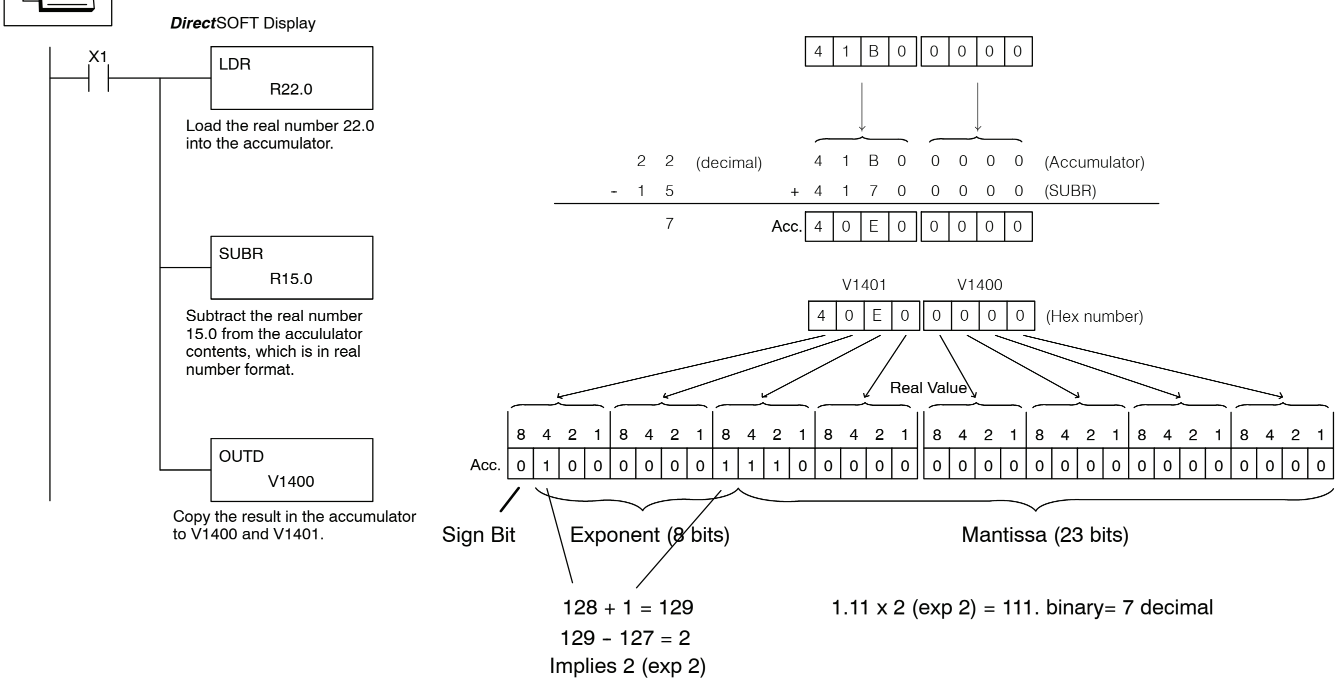
Subtract Real is a 32-bit instruction that subtracts a real number, which is either two consecutive V-memory locations or a 32-bit constant, from a real number in the accumulator. Both numbers must conform to the IEEE floating point format. The result is a 32-bit real number that resides in the accumulator.



Operand Data Type	A	DL350 Range aaa
V-memory	V	All (See p. 3-29)
Pointer	P	All V mem (See p. 3-29)
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.

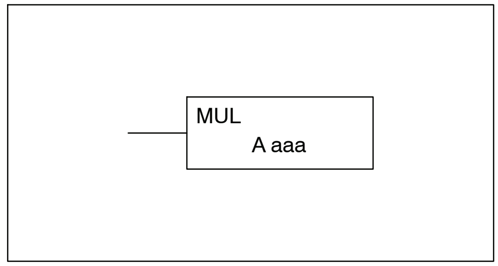
NOTE: Status flags are valid only until another instruction uses the same flag.



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)

Multiply is a 16 bit instruction that multiplies the BCD value (Aaaa), which is either a V–memory location or a 4–digit (max.) constant, by the BCD value in the lower 16 bits of the accumulator The result can be up to 8 digits and resides in the accumulator.

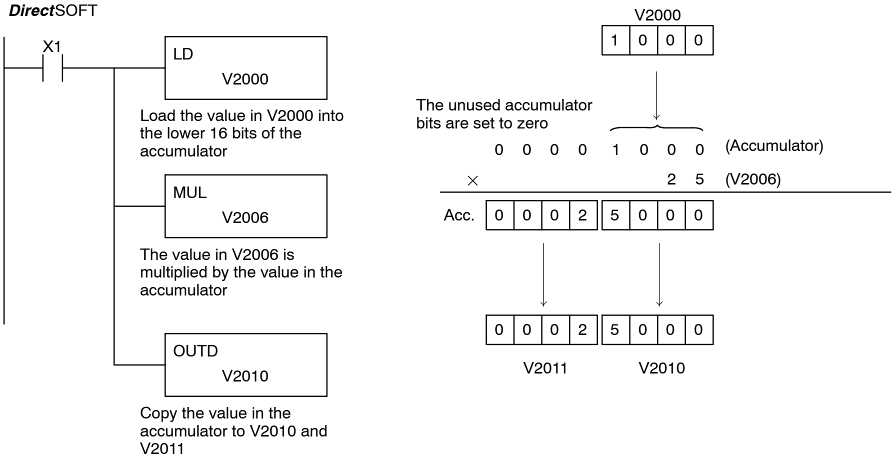


Operand Data Type		DL350 Range
	A	aaa
V–memory	V	All (See page 3–29)
Pointer	P	All V mem. (See page 3–29)
Constant	K	0–9999

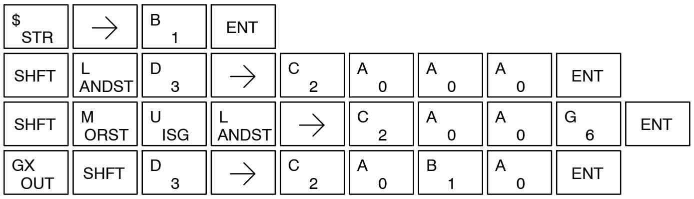
Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.
SP75	On when a BCD instruction is executed and a NON–BCD number was encountered.

NOTE: The status flags are only valid until another instruction that uses the same flags is executed.

In the following example, when X1 is on, the value in V2000 will be loaded into the accumulator using the Load instruction. The value in V2006 is multiplied by the value in the accumulator. The value in the accumulator is copied to V2010 and V2011 using the Out Double instruction.

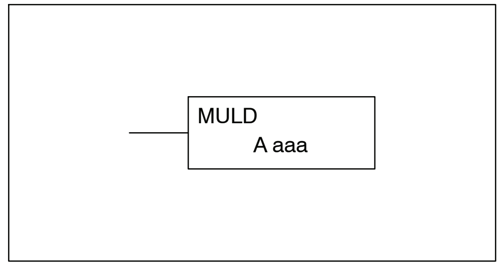


Handheld Programmer Keystrokes



Multiply Double (MULD)

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. You cannot use a constant as the parameter in the box. The lower 8 digits of the results reside in the accumulator. Upper digits of the result reside in the accumulator stack.



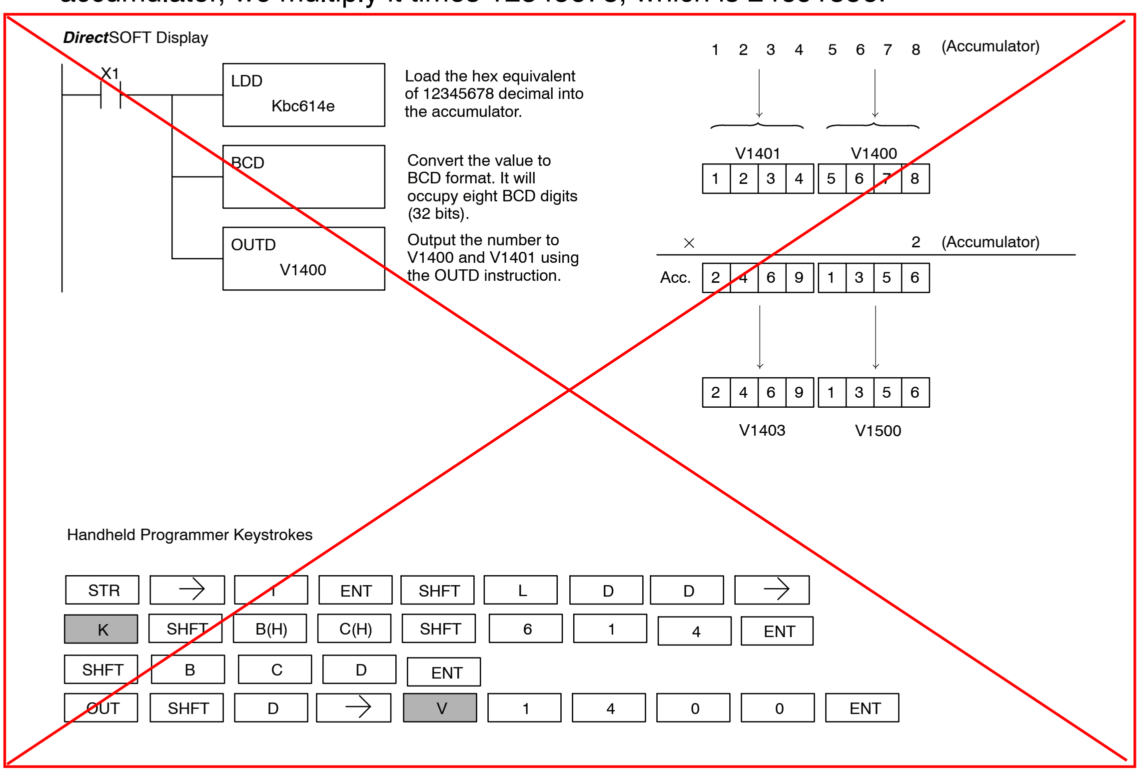
Operand Data Type		DL350 Range
	A	aaa
V–memory	V	All (See p. 3–29)
Pointer	P	––

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.

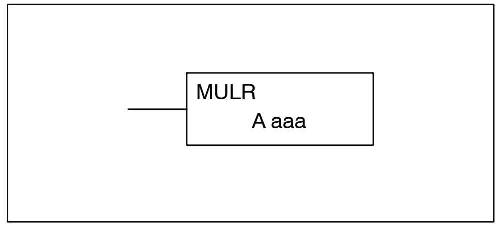
In the following example, when X1 is on, the constant Kbc614e hex will be loaded into the accumulator. When converted to BCD the number is ”12345678”. That numberis stored in V1400 and V1401. After loading the constant K2 into the accumulator, we multiply it times 12345678, which is 24691356.

Incorrect ladder example. See the Errata Sheet at the beginning of this file for a correct example.



Multiply Real (MULR)

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.

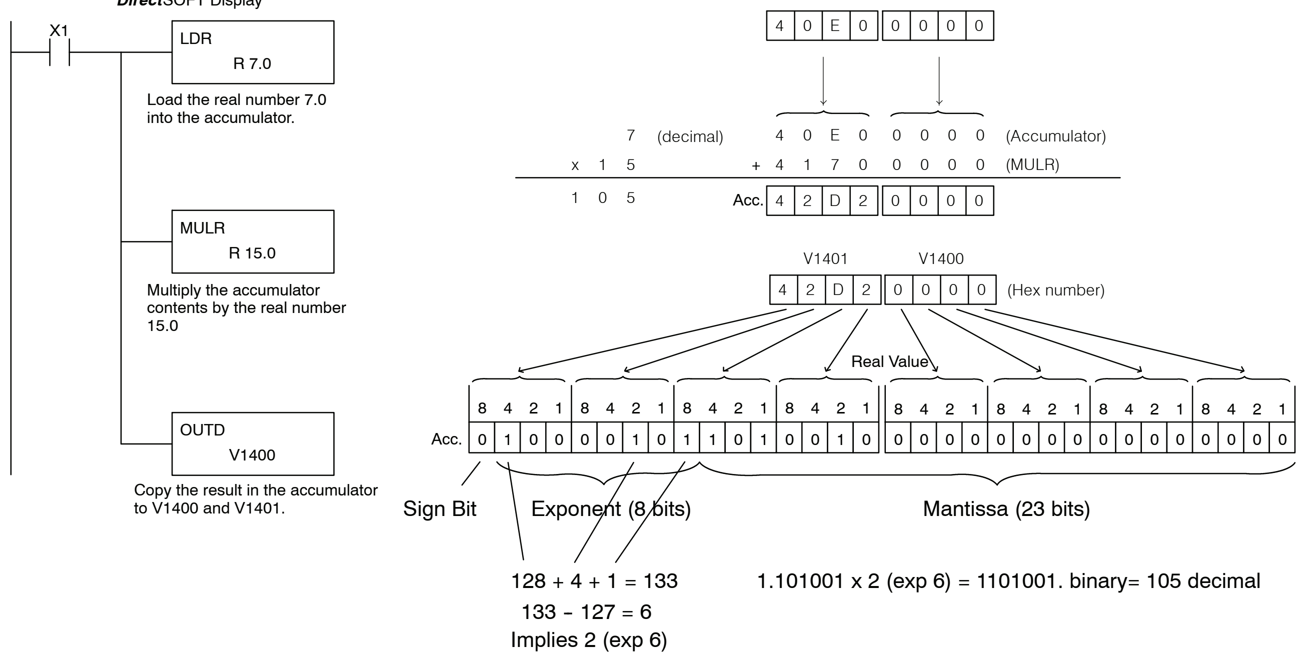


Operand Data Type		DL350 Range
	A	aaa
V–memory	V	All (See p. 3–29)
Pointer	P	All (See p. 3–29)
Constant	R	–3.402823E+038 to +3.402823E+038

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.

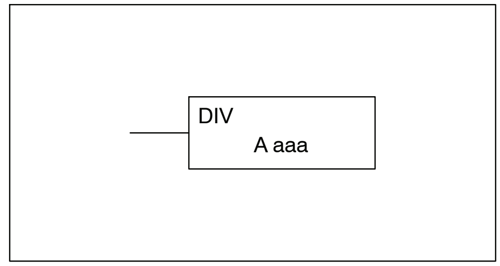
***Direct*SOFT Display**



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.

Divide (DIV)

Divide is a 16 bit instruction that divides the BCD value in the accumulator by a BCD value (Aaaa), which is either a V–memory location or a 4-digit (max.) constant. The first part of the quotient resides in the accumulator and the remainder resides in the first stack location.

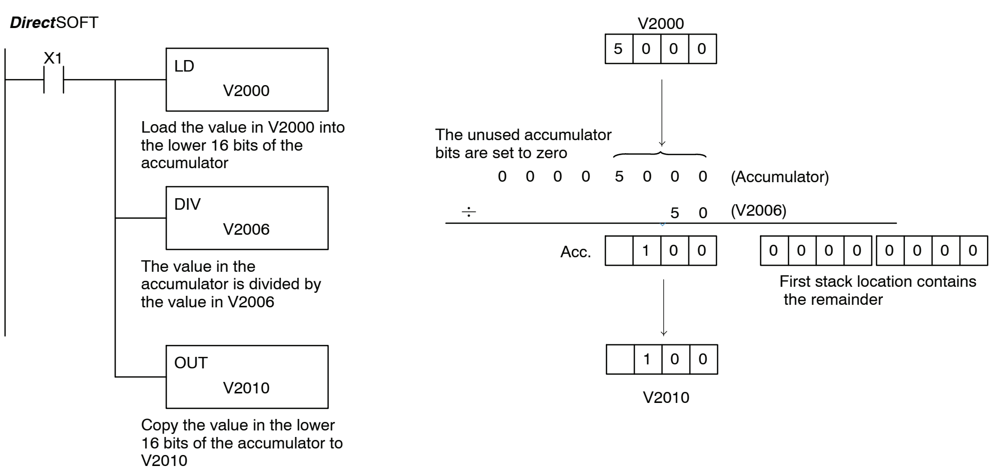


Operand Data Type		DL350 Range
	A	**aaa**
V–memory	V	All (See page 3–29)
Pointer	P	All V mem. (See page 3–29)
Constant	K	0–9999

Discrete Bit Flags	Description
SP53	On when the value of the operand is larger than the accumulator can work with.
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.
SP75	On when a BCD instruction is executed and a NON–BCD number was encountered.

NOTE: The status flags are only valid until another instruction that uses the same flags is executed.

In the following example, when X1 is on, the value in V2000 will be loaded into the accumulator using the Load instruction. The value in the accumulator will be divided by the value in V2006 using the Divide instruction. The value in the accumulator is copied to V2010 using the Out instruction.

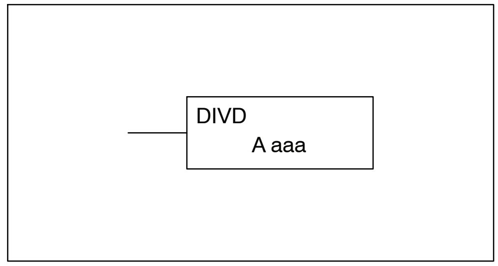


Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	→	C 2	A 0	A 0	A 0	ENT	
SHFT	D 3	I 8	V AND	→	C 2	A 0	A 0	G 6	ENT
GX OUT	→	SHFT	V AND	C 2	A 0	B 1	A 0	ENT	

Divide Double (DIVD)

Divide Double is a 32 bit instruction that divides the BCD value in the accumulator by a BCD value (Aaaa), which must be obtained from two consecutive V–memory locations. You cannot use a constant as the parameter in the box. The first part of the quotient resides in the accumulator and the remainder resides in the first stack location.

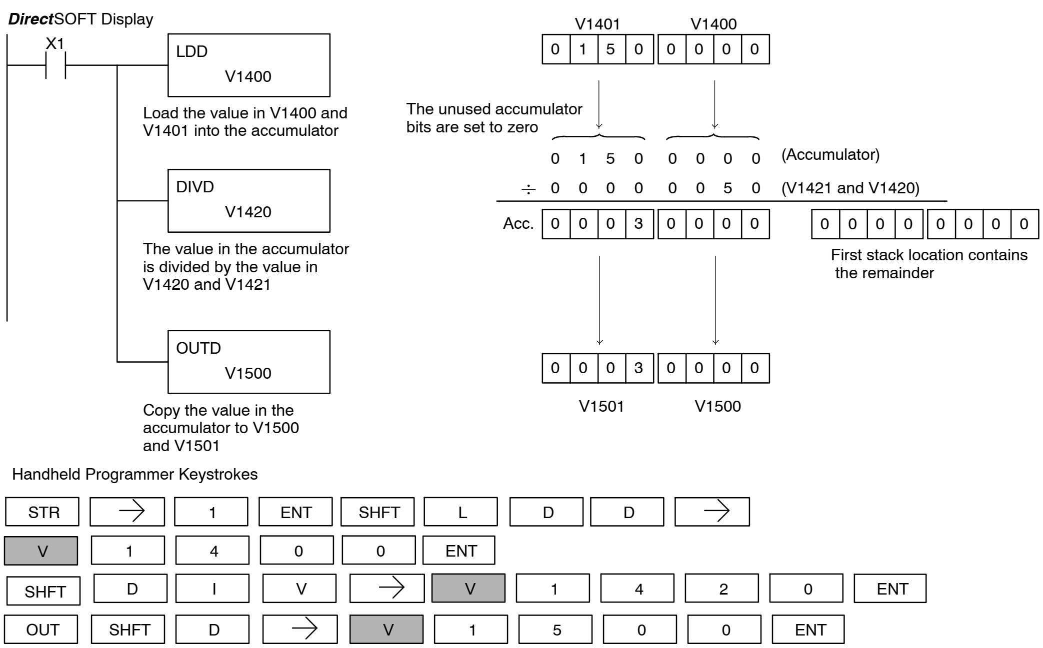


Operand Data Type		DL350 Range
	A	aaa
V–memory	V	All (See p. 3–29)
Pointer	P	All (See p. 3–29)

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.

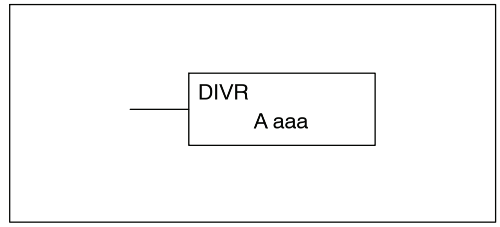
NOTE: Status flags are valid only until another instruction uses the same flag.

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 an the remainder resides in the first stack location. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.



Divide Real (DIVR)

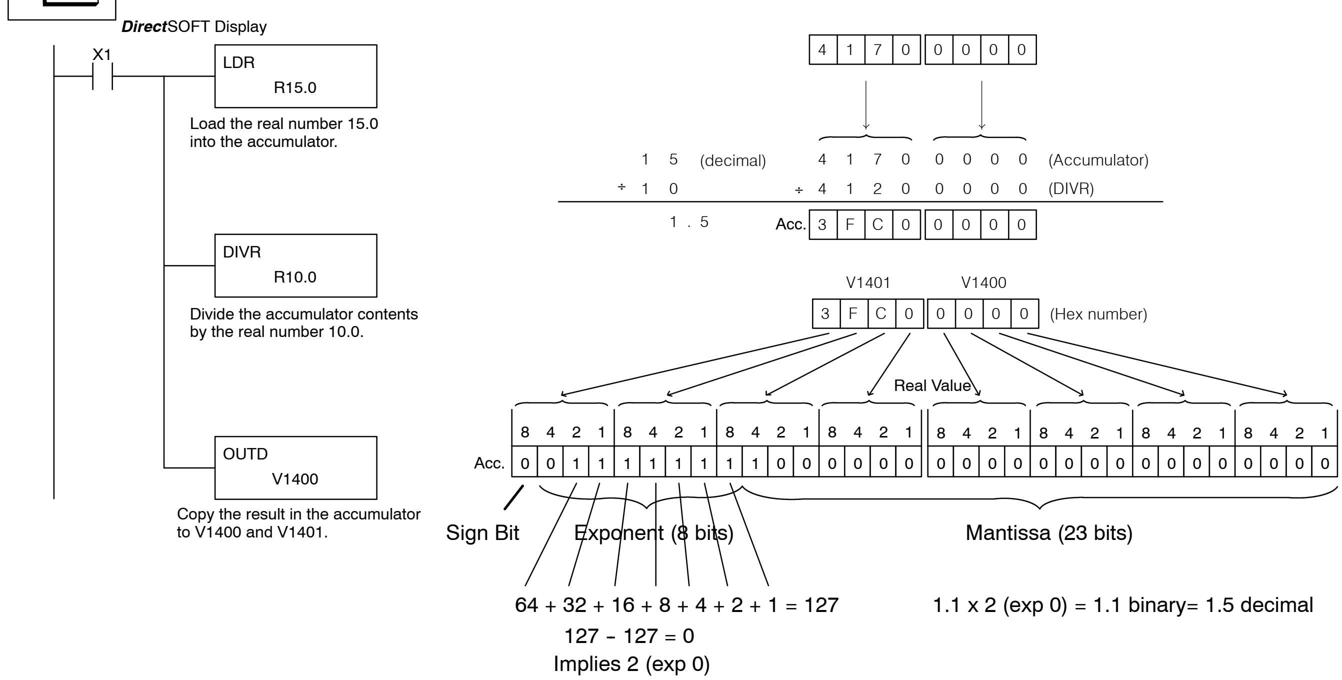
The Divide Real instruction divides a real number in the accumulator by either a real constant or a real number occupying two consecutive V-memory locations. The result resides in the accumulator. Both numbers must conform to the IEEE floating point format.



Operand Data Type		DL350 Range
	A	aaa
V–memory	V	All (See p. 3–29)
Pointer	P	All (See p. 3–29)
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.

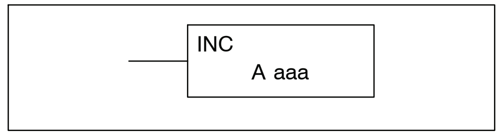
NOTE: Status flags are valid only until another instruction uses the same flag.



NOTE: The current HPP does not support real number entry with automatic conversion to the 32-bit IEEE format. You must use ***Direct***SOFT for this feature.

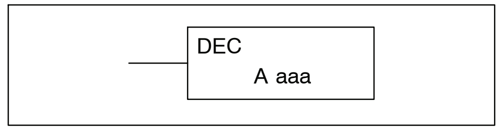
Increment (INC)

The Increment instruction increments a BCD value in a specified V-memory location by “1” each time the instruction is executed.



Decrement (DEC)

The Decrement instruction decrements a BCD value in a specified V-memory location by “1” each time the instruction is executed.

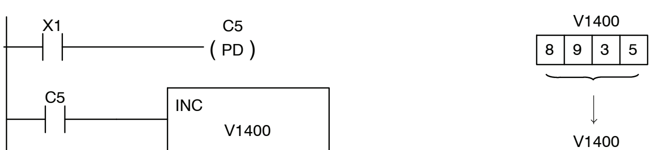


Operand Data Type		DL350 Range
	A	aaa
V-memory	V	All (See p. 3-29)
Pointer	P	All (See p. 3-29)

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.



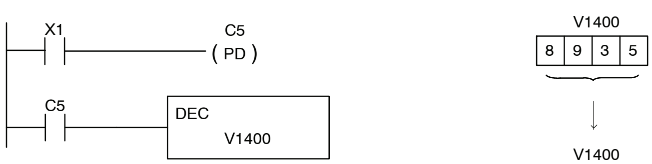
Increment the value in V1400 by “1”.

Handheld Programmer Keystrokes

STR	→	C	5	ENT

SHFT	I	N	C	→	V	1	4	0	0	ENT

In the following decrement example, when C5 is on the value in V1400 is decreased by one.



Decrement the value in V1400 by “1”.

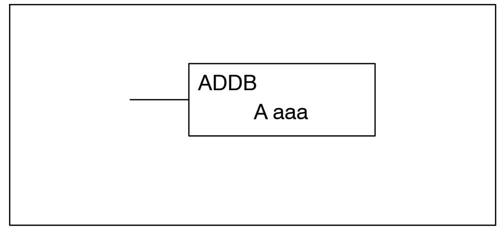
Handheld Programmer Keystrokes

STR	→	C	5	ENT

SHFT	D	E	C	→	V	1	4	0	0	ENT

Add Binary (ADDB)

Add Binary is a 16 bit instruction that adds the binary value in the lower 16 bits of the accumulator with a binary value (Aaaa), which is either a V–memory location or a 16-bit constant. The result can be up to 32 bits and resides in the accumulator.

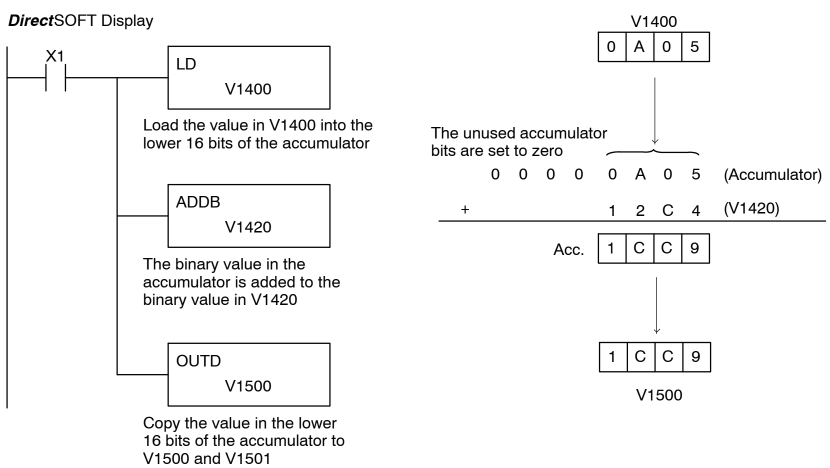


Operand Data Type		DL350 Range
	A	aaa
V–memory	V	All (See p. 3–29)
Pointer	P	All V mem (See p. 3–29)
Constant	K	0–FFFF

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
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 will be loaded into the accumulator using the Load instruction. The binary value in the accumulator will be added to the binary value in V1420 using the Add Binary instruction. The value in the accumulator is copied to V1500 and V1501 using the Out instruction.

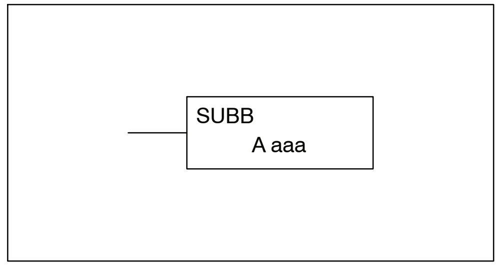


Handheld Programmer Keystrokes

STR	→	X(IN)	1	ENT							
SHFT	L	D	V	1	4	0	0	ENT			
SHFT	A	D	D	B	→	V	1	4	2	0	ENT
OUT	SHFT	D	→	V	1	5	0	0	ENT		

Subtract Binary (SUBB)

Subtract Binary is a 16 bit instruction that subtracts the binary value (Aaaa), which is either a V–memory location or a 4–digit (max.) binary constant, from the binary value in the accumulator. The result resides in the accumulator.



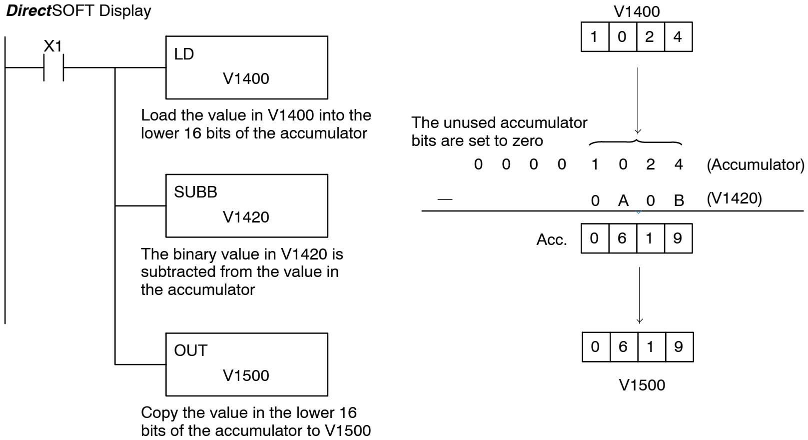
Operand Data Type		DL350 Range
	A	aaa
V–memory	V	All (See p. 3–29)
Pointer	P	All (See p. 3–29)
Constant	K	0–FFFF

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

***Direct*SOFT Display**

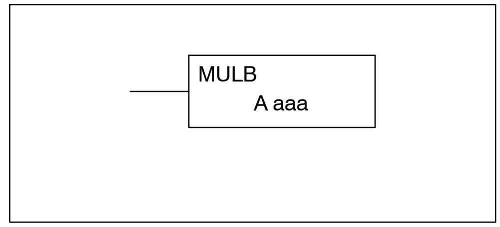


Handheld Programmer Keystrokes

STR	→	X(IN)	1	ENT					
SHFT	L	D	V	1	4	0	0	ENT	
SHFT	S	SHFT	U	B	B	→			
V	1	4	2	0	ENT				
OUT	SHFT	D	→	V	1	5	0	0	ENT

Multiply Binary (MULB)

Multiply Binary is a 16 bit instruction that multiplies the binary value (Aaaa), which is either a V–memory location or a 4–digit (max.) binary constant, by the binary value in the accumulator. The result can be up to 32 bits and resides in the accumulator.



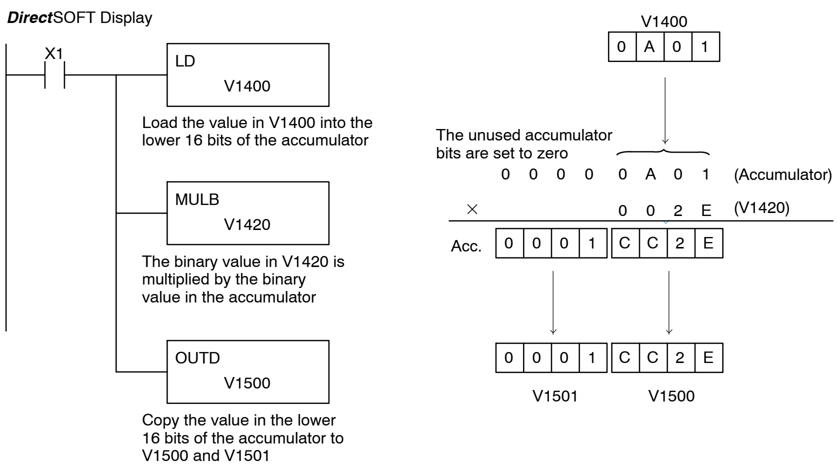
Operand Data Type		DL350 Range
	A	aaa
V–memory	V	All (See p. 3–29)
Pointer	P	All (See p. 3–29)
Constant	K	0–FFFF

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.

NOTE: Status flags are valid only until another instruction uses the same flag.

In the following example, when X1 is on, the value in V1400 will be loaded into the accumulator using the Load instruction. The binary value in V1420 is multiplied by the binary value in the accumulator using the Multiply Binary instruction. The value in the accumulator is copied to V1500 using the Out instruction.

***Direct*SOFT Display**

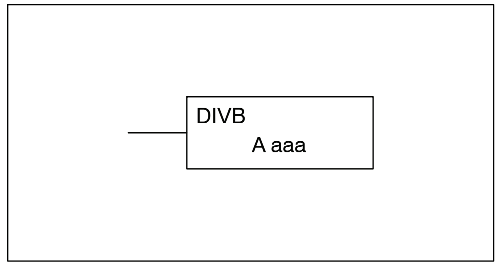


Handheld Programmer Keystrokes

STR	→	X	1	ENT							
SHFT	L	D	V	1	4	0	0	ENT			
SHFT	M	U	L	B	→	V	1	4	2	0	ENT
OUT	SHFT	D	→	V	1	5	0	0	ENT		

Divide Binary (DIVB)

Divide Binary is a 16 bit instruction that divides the binary value in the accumulator by a binary value (Aaaa), which is either a V-memory location or a 16-bit (max.) binary constant. The first part of the quotient resides in the accumulator and the remainder resides in the first stack location.

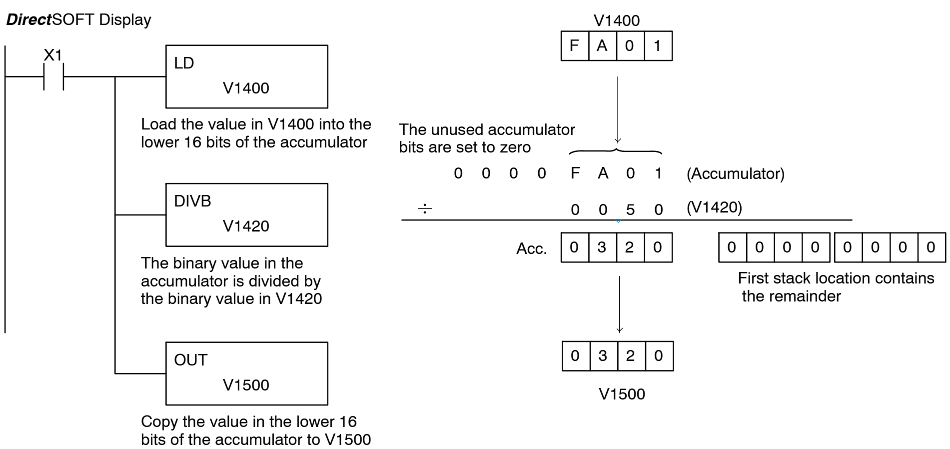


Operand Data Type		DL350 Range
	A	aaa
V-memory	V	All (See p. 3-29)
Pointer	P	All (See p. 3-29)
Constant	K	0-FFFF

Discrete Bit Flags	Description
SP53	On when the value of the operand is larger than the accumulator can work with.
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.

NOTE: Status flags are valid only until another instruction uses the same flag.

In the following example, when X1 is on, the value in V1400 will be loaded into the accumulator using the Load instruction. The binary value in the accumulator is divided by the binary value in V1420 using the Divide Binary instruction. The value in the accumulator is copied to V1500 using the Out instruction.

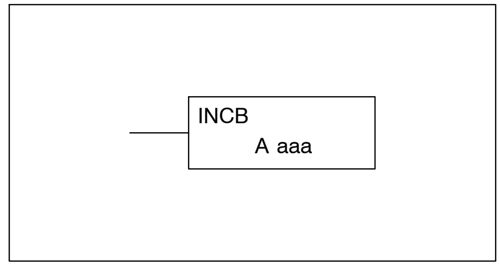


Handheld Programmer Keystrokes

STR	→	X	1	ENT							
SHFT	L	D	V	1	4	0	0	ENT			
SHFT	D	I	V	B	→	V	1	4	2	0	ENT
OUT	SHFT	D	→	V	1	5	0	0	ENT		

Increment Binary (INCB)

The Increment Binary instruction increments a binary value in a specified V–memory location by “1” each time the instruction is executed.

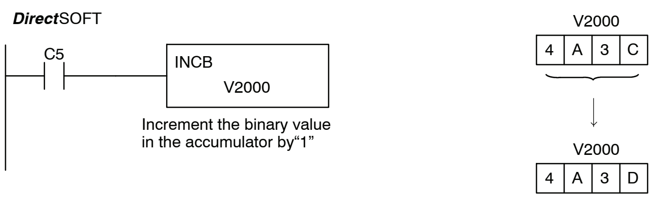


Operand Data Type		DL350 Range
	A	**aaa**
V–memory	V	All (See page 3–29)
Pointer	P	All V mem. (See page 3–29)

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

NOTE: The status flags are only valid until another instruction that uses the same flags is executed.

In the following example when C5 is on, the binary value in V2000 is increased by 1.

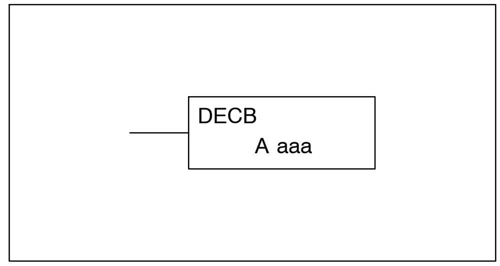


Handheld Programmer Keystrokes

$ STR	→	SHFT	C 2	F 5	ENT					
SHFT	I 8	N TMR	C 2	B 1	→	C 2	A 0	A 0	A 0	ENT

Decrement Binary (DECB)

The Decrement Binary instruction decrements a binary value in a specified V–memory location by "1" each time the instruction is executed.

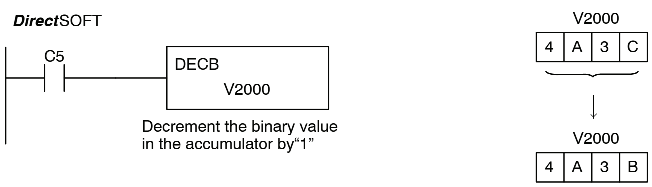


Operand Data Type		DL350 Range
	A	**aaa**
V–memory	V	All (See page 3–29)
Pointer	P	All V mem. (See page 3–29)

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

NOTE: The status flags are only valid until another instruction that uses the same flags is executed.

In the following example when C5 is on, the value in V2000 is decreased by 1.



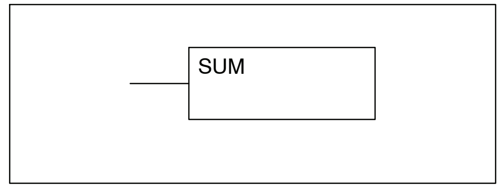
Handheld Programmer Keystrokes

$ STR	→	SHFT	C 2	F 5	ENT					
SHFT	D 3	E 4	C 2	B 1	→	C 2	A 0	A 0	A 0	ENT

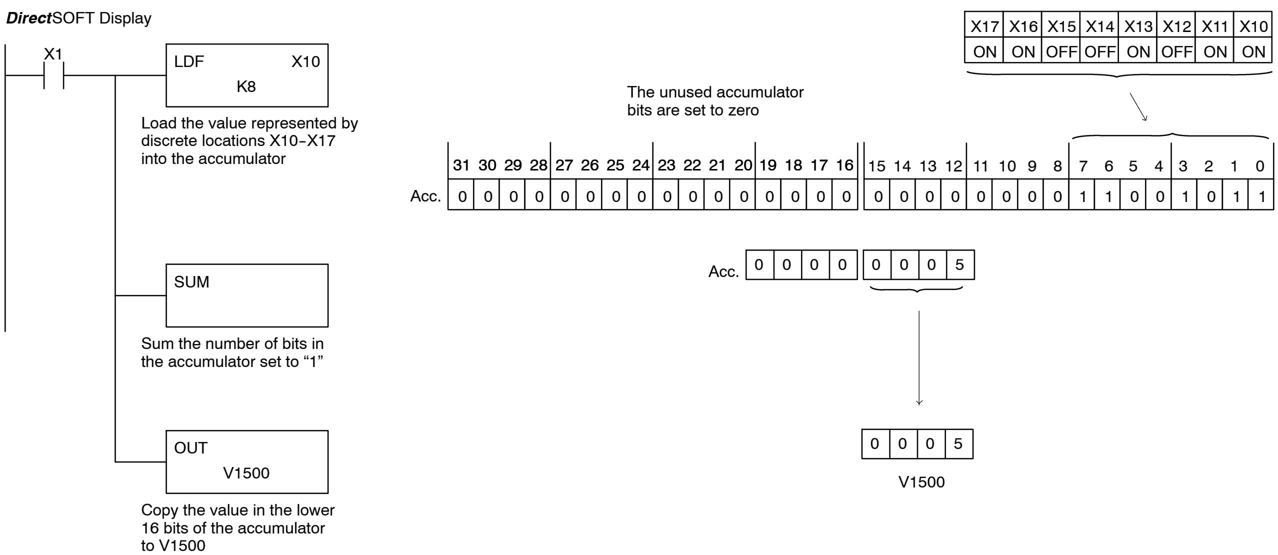
Bit Operation Instructions

Sum (SUM)

The Sum instruction counts number of bits that are set to “1” in the accumulator. The HEX result resides in the accumulator.



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.

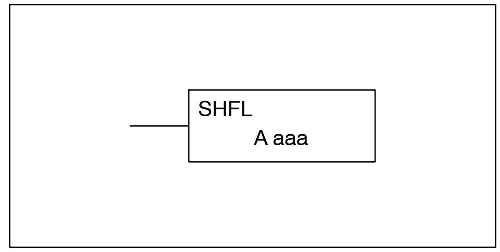


Handheld Programmer Keystrokes

STR	→	X	1	ENT							
SHFT	L	D	F	→	X	1	0	→	K	8	ENT
SHFT	S	SHFT	U	M	ENT						
OUT	→	V	1	5	0	0					

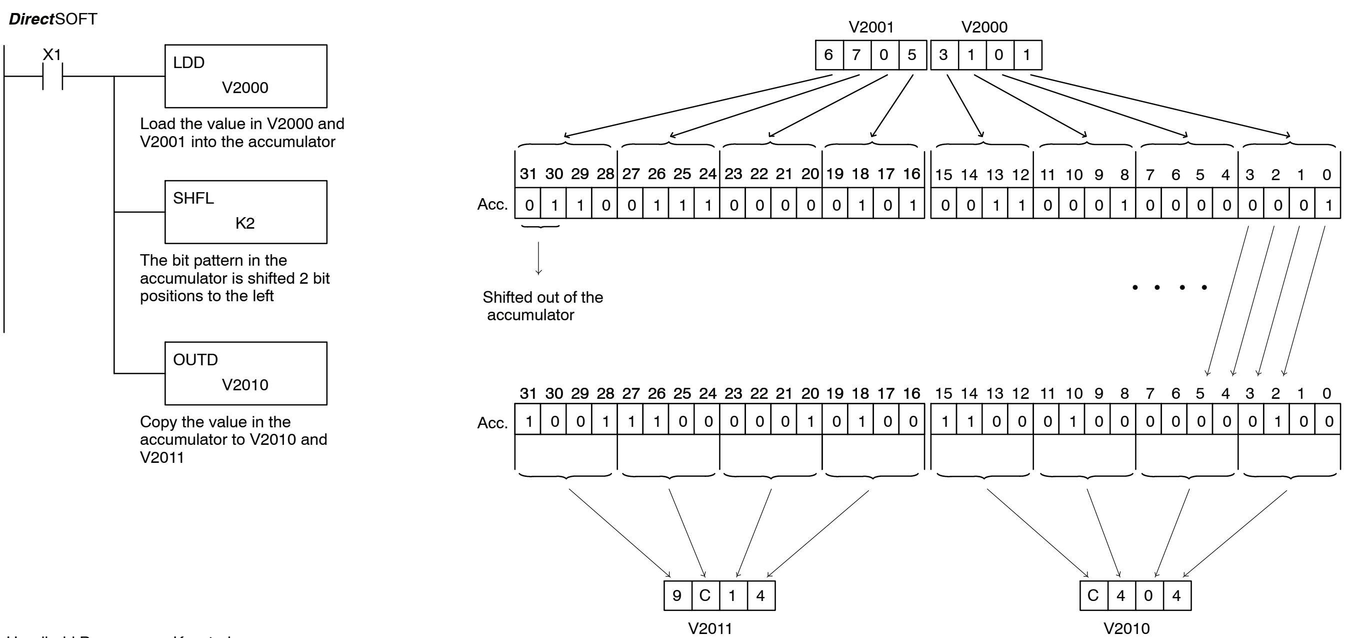
Shift Left (SHFL)

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.



Operand Data Type		DL350 Range
	A	aaa
V–memory	V	All (See page 3–29)
Constant	K	1–32

In the following example, when X1 is on, the value in V2000 and V2001 will be loaded into the accumulator using the Load Double instruction. The bit pattern in the accumulator is shifted 2 bits to the left using the Shift Left instruction. The value in the accumulator is copied to V2010 and V2011 using the Out Double instruction.



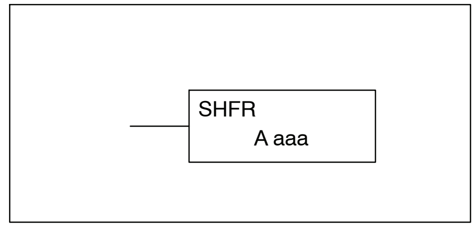
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT					
SHFT	L ANDST	D 3	D 3	→	C 2	A 0	A 0	ENT
SHFT	S RST	SHFT	H 7	F 5	L ANDST	→	C 2	ENT
GX OUT	SHFT	D 3	→	C 2	A 0	B 1	A 0	ENT

Standard RLL Instructions

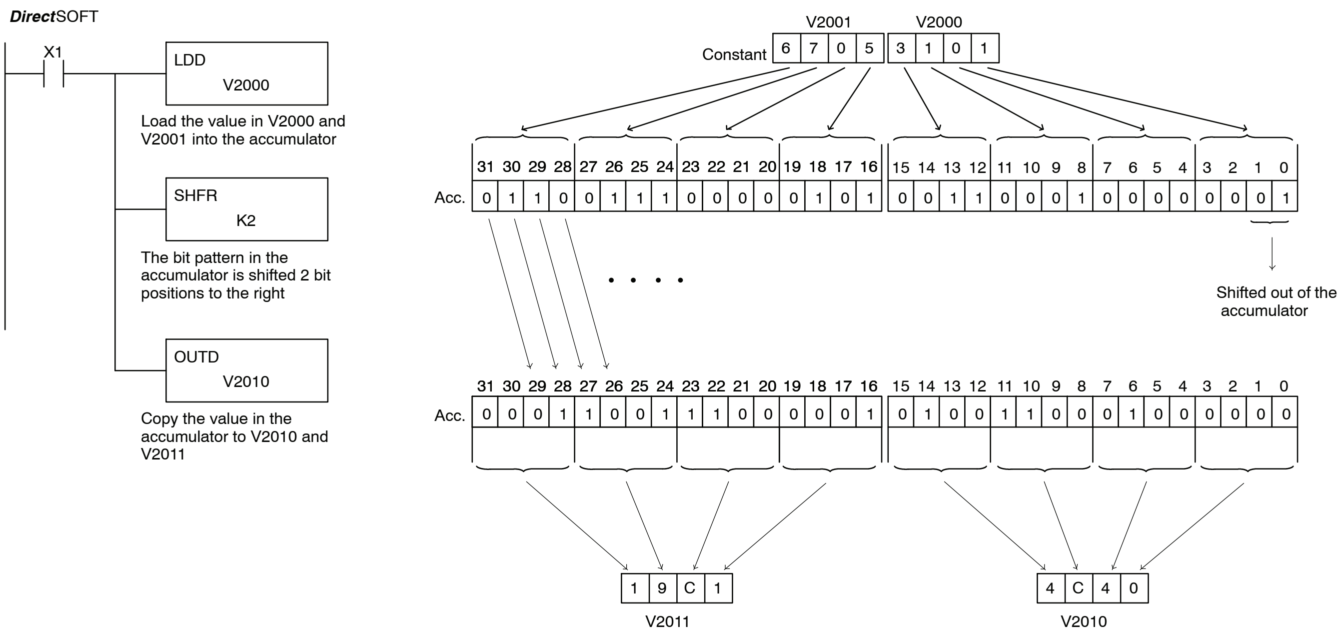
Shift Right (SHFR)

Shift Right is a 32 bit instruction that shifts the bits in the accumulator a specified number (Aaaa) of places to the right. The vacant positions are filled with zeros and the bits shifted out of the accumulator are lost.

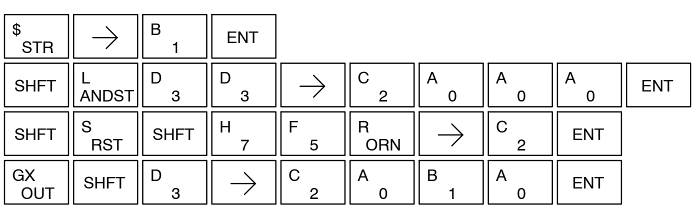


Operand Data Type		DL350 Range
	A	aaa
V–memory	V	All (See page 3–29)
Constant	K	1–32

In the following example, when X1 is on, the value in V2000 and V2001 will be loaded into the accumulator using the Load Double instruction. The bit pattern in the accumulator is shifted 2 bits to the right using the Shift Right instruction. The value in the accumulator is copied to V2010 and V2011 using the Out Double instruction.

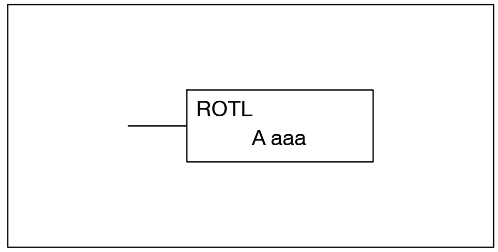


Handheld Programmer Keystrokes



Rotate Left (ROTL)

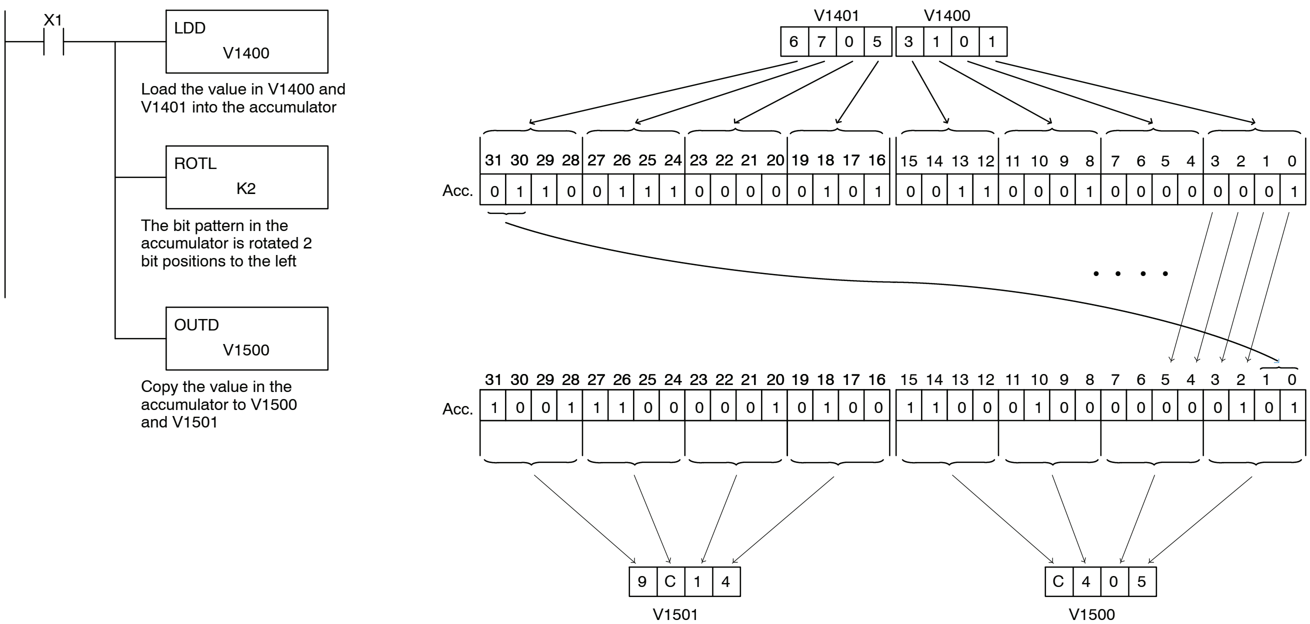
Rotate Left is a 32 bit instruction that rotates the bits in the accumulator a specified number (Aaaa) of places to the left.



Operand Data Type		DL350 Range
	A	aaa
V–memory	V	All (See p. 3–29)
Constant	K	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.

*Direct*SOFT Display

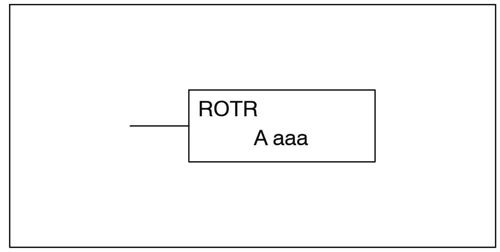


Handheld Programmer Keystrokes

STR	→	X	1	ENT					
SHFT	L	D	D	→	V	1	4	0	0
SHFT	R	O	T	L	→	K	2	ENT	
OUT	SHFT	D	→	V	1	5	0	0	ENT

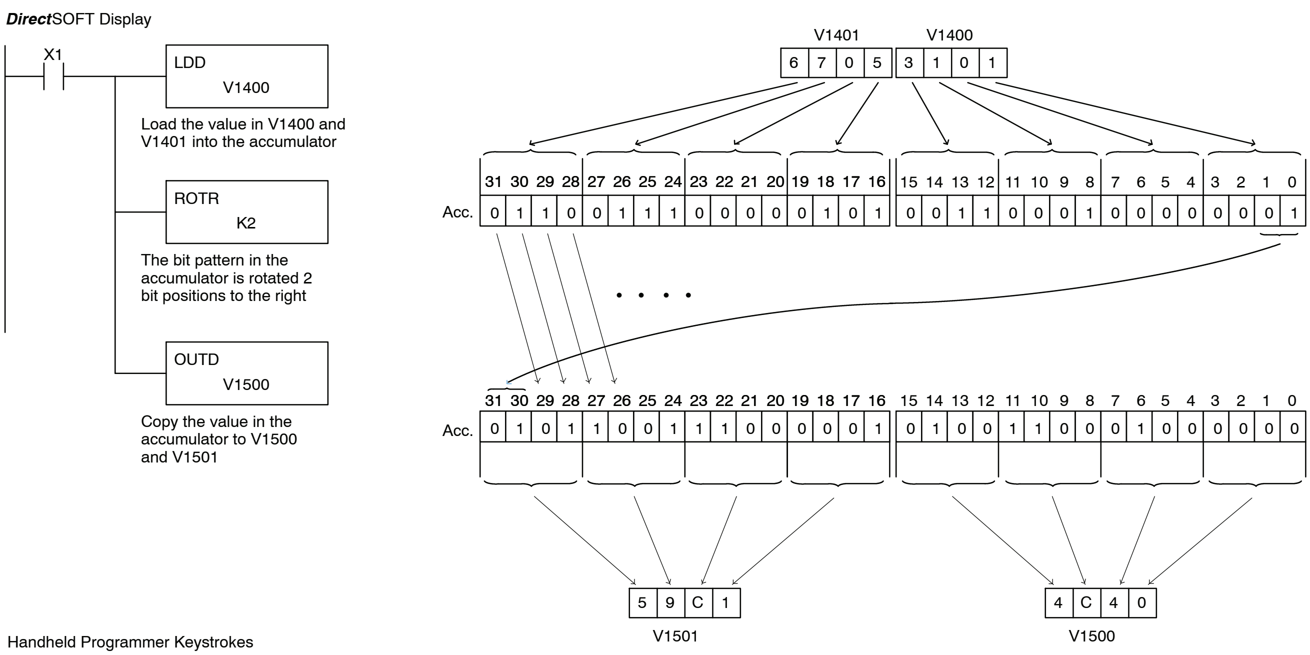
Rotate Right (ROTR)

Rotate Right is a 32 bit instruction that rotates the bits in the accumulator a specified number (Aaaa) of places to the right.



Operand Data Type		DL350 Range
	A	aaa
V–memory	V	All (See p. 3–29)
Constant	K	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.

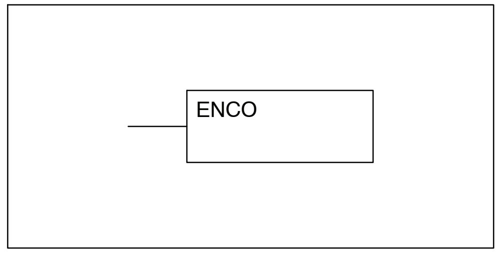


Handheld Programmer Keystrokes

STR	→	X	1	ENT					
SHFT	L	D	D	→	V	1	4	0	0
SHFT	R	O	T	R	→	K	2	ENT	
OUT	SHFT	D	→	V	1	5	0	0	ENT

Encode (ENCO)

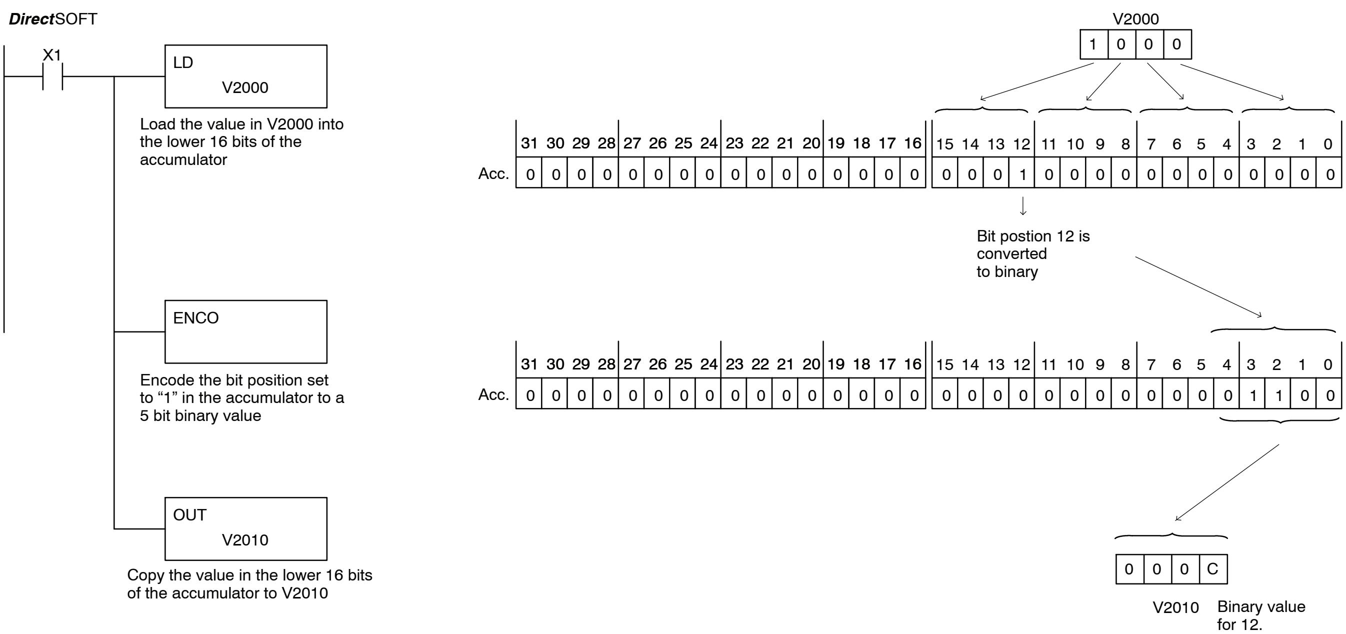
The Encode instruction encodes the bit position in the accumulator having a value of 1, and returns the appropriate binary representation. If the most significant bit is set to 1 (Bit 31), the Encode instruction would place the value HEX 1F (decimal 31) in the accumulator. If the value to be encoded is 0000 or 0001, the instruction will place a zero in the accumulator. If the value to be encoded has more than one bit position set to a "1", the least significant "1" will be encoded and SP53 will be set on.



Discrete Bit Flags	Description
SP53	On when the value of the operand is larger than the accumulator can work with.

NOTE: The status flags are only valid until another instruction that uses the same flags is executed.

In the following example, when X1 is on, The value in V2000 is loaded into the accumulator using the Load instruction. The bit position set to a "1" in the accumulator is encoded to the corresponding 5 bit binary value using the Encode instruction. The value in the lower 16 bits of the accumulator is copied to V2010 using the Out instruction.

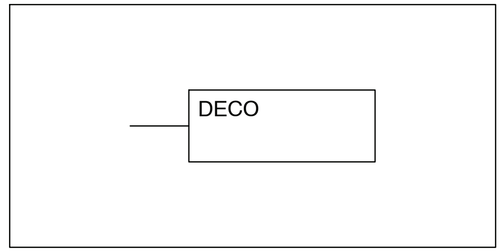


Handheld Programmer Keystrokes

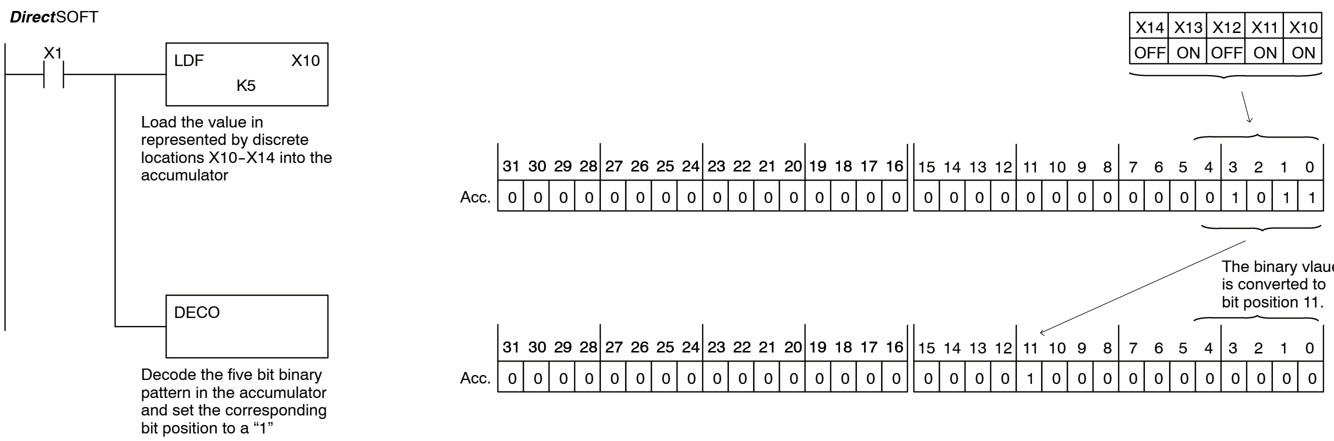
$ STR	→	B 1	ENT					
SHFT	L ANDST	D 3	→	C 2	A 0	A 0	A 0	ENT
SHFT	E 4	N TMR	C 2	O INST#	ENT			
GX OUT	→	SHFT	V AND	C 2	A 0	B 1	A 0	ENT

Decode (DECO)

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.



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.



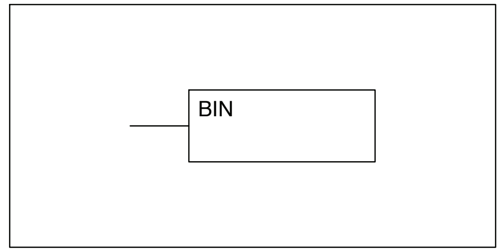
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	F 5	→	B 1	A 0	→	F 5	ENT
SHFT	D 3	E 4	C 2	O INST#	ENT				

Number Conversion Instructions (Accumulator)

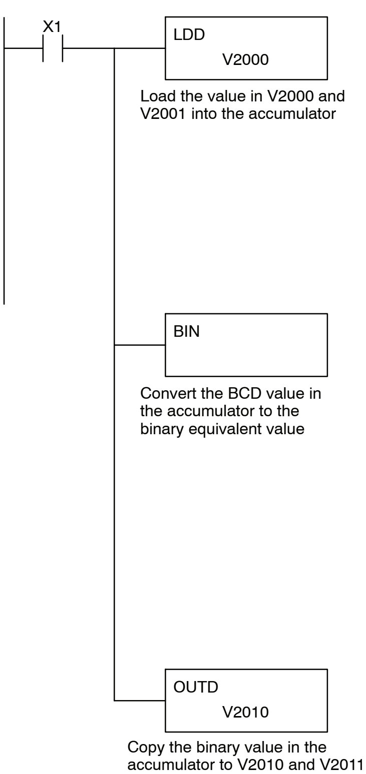
Binary (BIN)

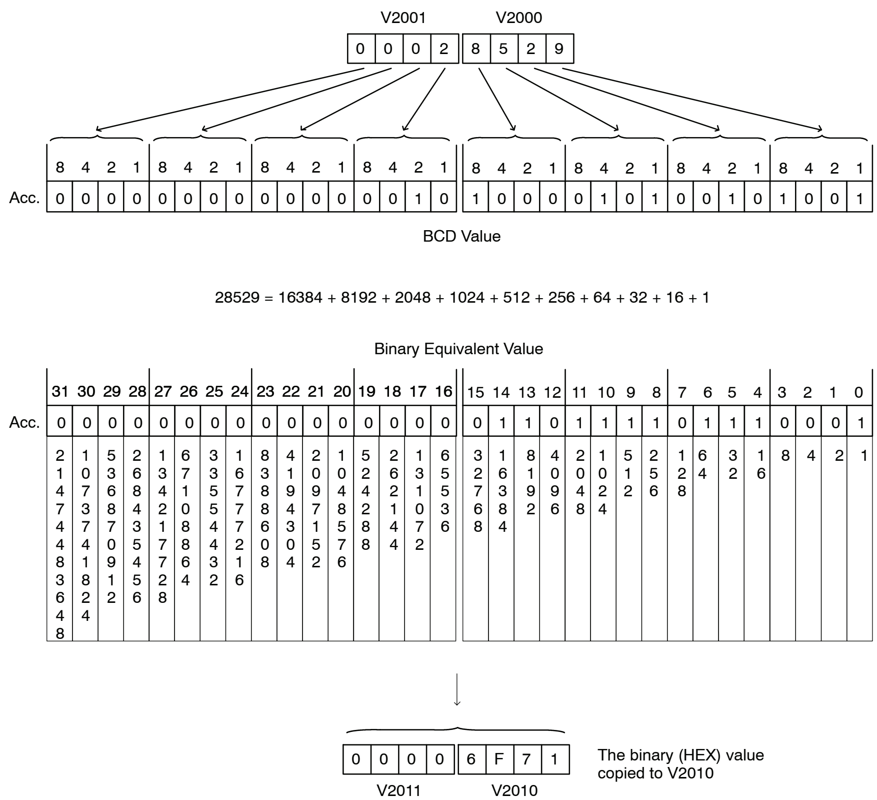
The Binary instruction converts a BCD value in the accumulator to the equivalent binary value. The result resides in the accumulator.



In the following example, when X1 is on, the value in V2000 and V2001 is loaded into the accumulator using the Load Double instruction. The BCD value in the accumulator is converted to the binary (HEX) equivalent using the BIN instruction. The binary value in the accumulator is copied to V2010 and V2011 using the Out Double instruction. (The handheld programmer will display the binary value in V2010 and V2011 as a HEX value.)

*Direct*SOFT



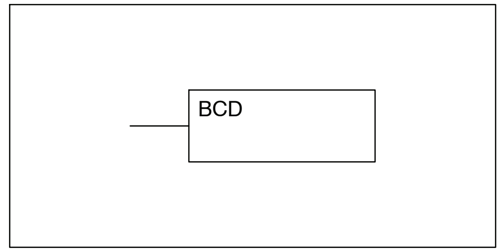


Handheld Programmer Keystrokes

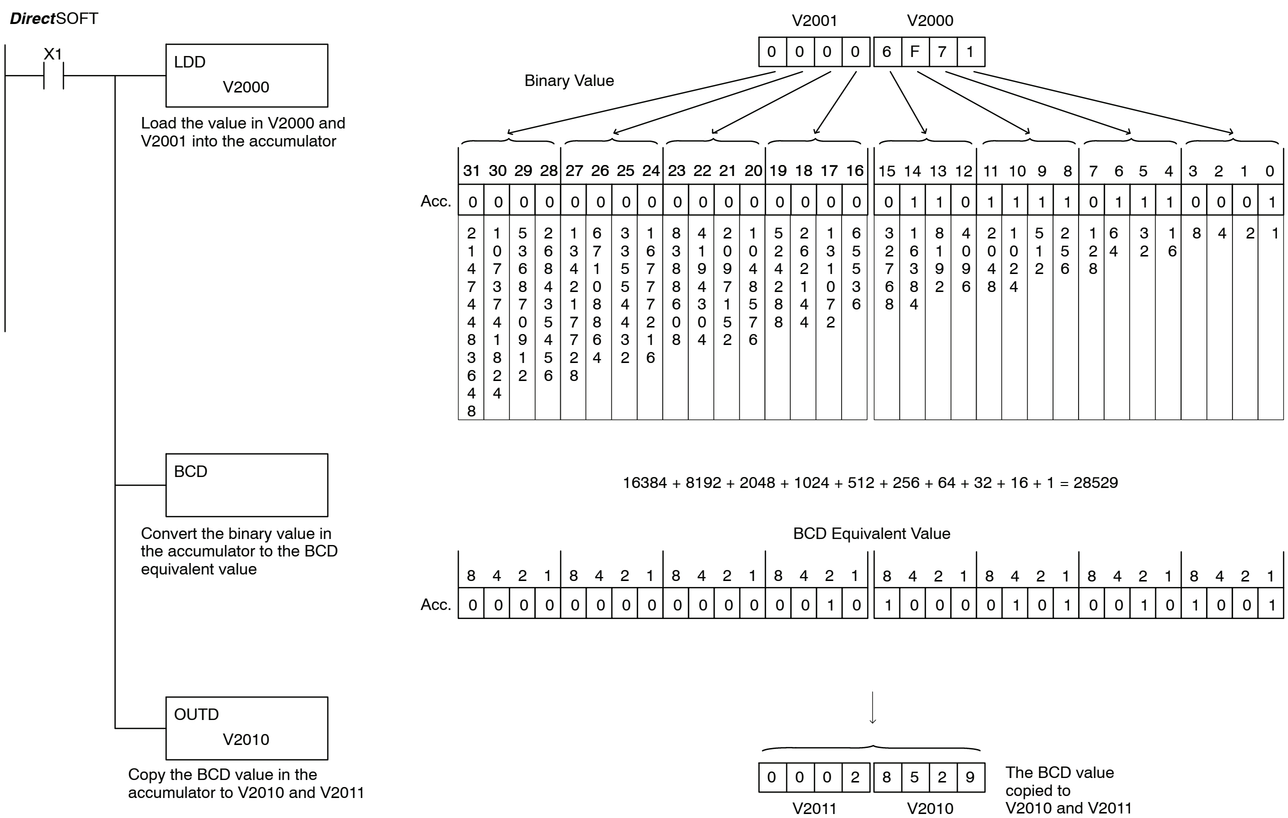
$ STR	→	B 1	ENT					
SHFT	L ANDST	D 3	D 3	→	C 2	A 0	A 0	A 0	ENT
SHFT	B 1	I 8	N TMR	ENT				
GX OUT	SHFT	D 3	→	C 2	A 0	B 1	A 0	ENT

Binary Coded Decimal (BCD)

The Binary Coded Decimal instruction converts a binary value in the accumulator to the equivalent BCD value. The result resides in the accumulator.



In the following example, when X1 is on, the binary (HEX) value in V2000 and V2001 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 V2010 and V2011 using the Out Double instruction.

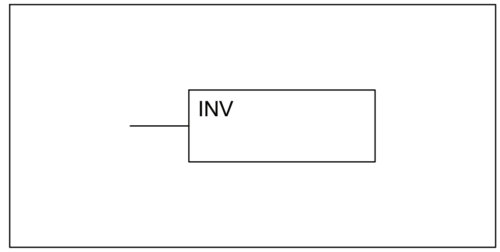


Handheld Programmer Keystrokes

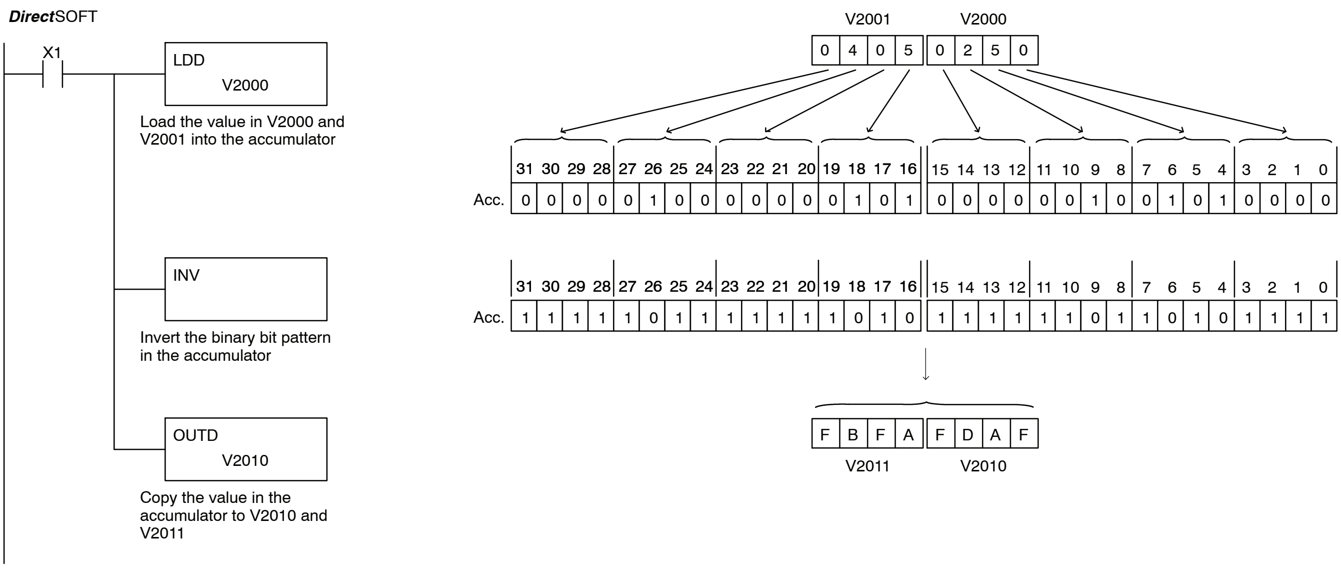
$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	D 3	→	C 2	A 0	A 0	A 0	ENT
SHFT	B 1	C 2	D 3	ENT					
GX OUT	SHFT	D 3	→	C 2	A 0	B 1	A 0	ENT	

Invert (INV)

The Invert instruction inverts or takes the one's complement of the 32 bit value in the accumulator. The result resides in the accumulator.



In the following example, when X1 is on, the value in V2000 and V2001 will be loaded into the accumulator using the Load Double instruction. The value in the accumulator is inverted using the Invert instruction. The value in the accumulator is copied to V2010 and V2011 using the Out Double instruction.



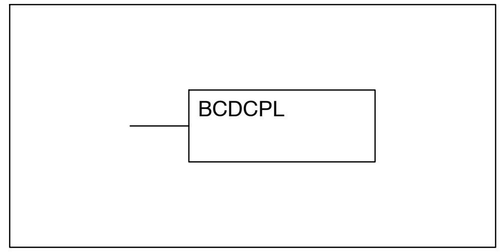
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT					
SHFT	L ANDST	D 3	D 3	→	C 2	A 0	A 0	ENT
SHFT	I 8	N TMR	V AND	ENT				
GX OUT	SHFT	D 3	→	C 2	A 0	B 1	A 0	ENT

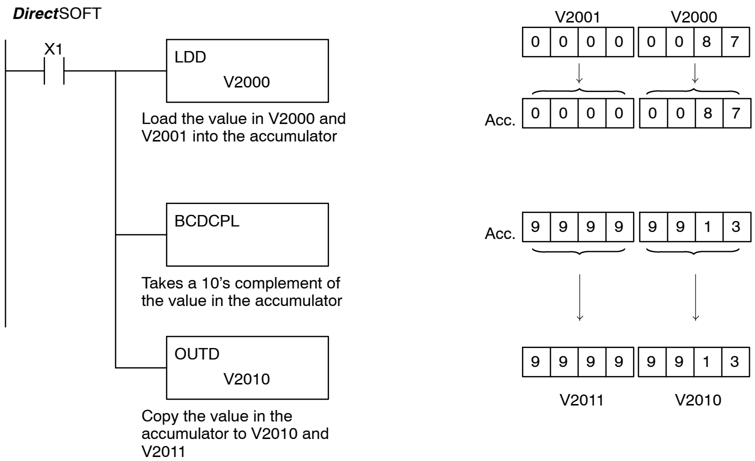
Ten's Complement (BCDCPL)

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 :

$$\begin{array}{r} 100000000 \\ -\ \text{accumulator value} \\ \hline \text{10's complement value} \end{array}$$



In the following example when X1 is on, the value in V2000 and V2001 is loaded into the accumulator. The 10's complement is taken for the 8 digit accumulator using the Ten's Complement instruction. The value in the accumulator is copied to V2010 and V2011 using the Out Double instruction.

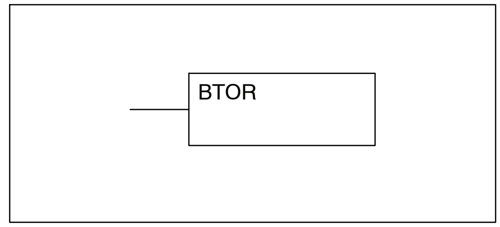


Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	D 3	→	C 2	A 0	A 0	A 0	ENT
SHFT	B 1	C 2	D 3	C 2	P CV	L ANDST	ENT		
GX OUT	SHFT	D 3	→	C 2	A 0	B 1	A 0	ENT	

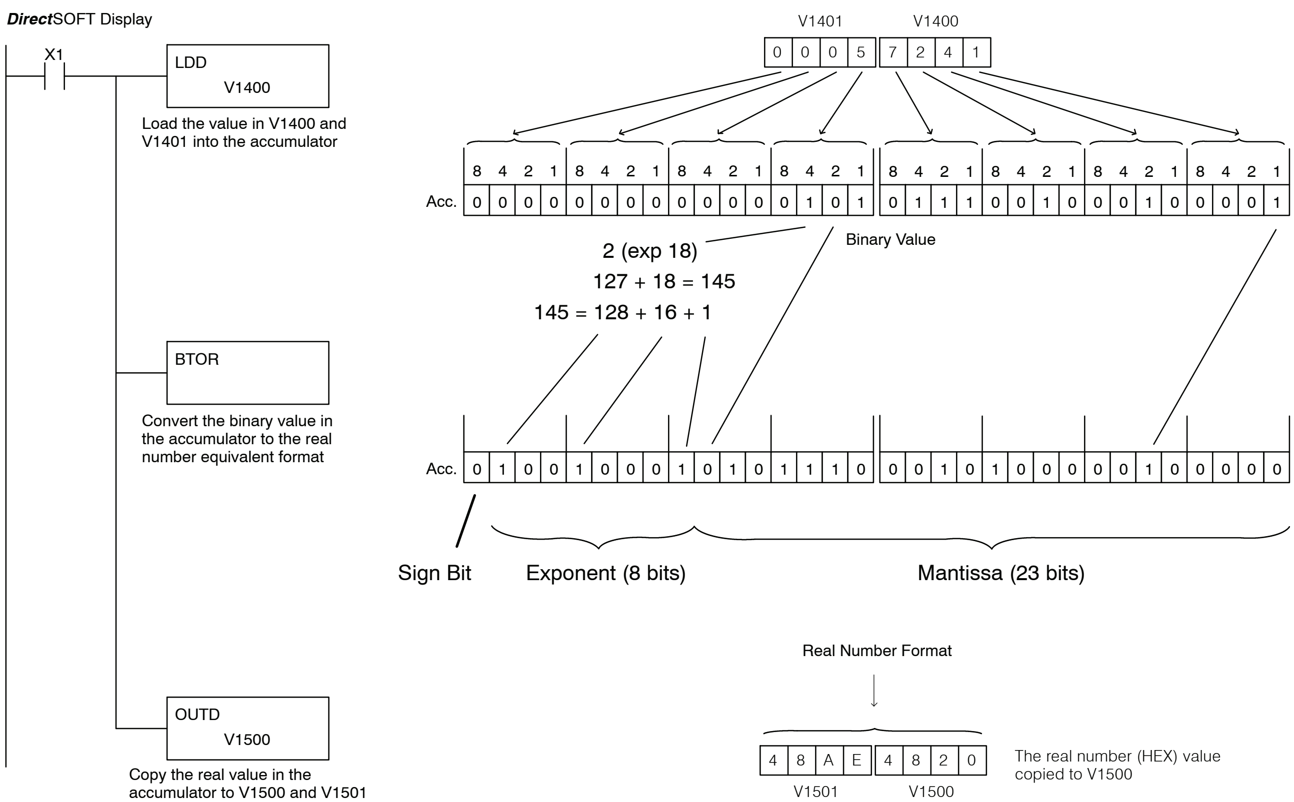
Binary to Real Conversion (BTOR)

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.



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.

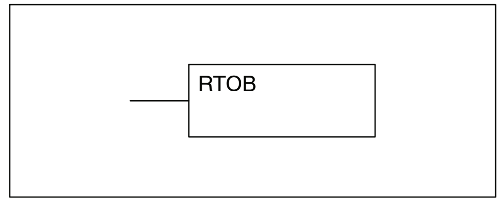


Handheld Programmer Keystrokes

STR	→	X	1	ENT						
SHFT	L	D	D	→	V	1	4	0	0	ENT
SHFT	B	T	O	R	SHFT	ENT				
OUT	SHFT	D	→	V	1	5	0	0	ENT	

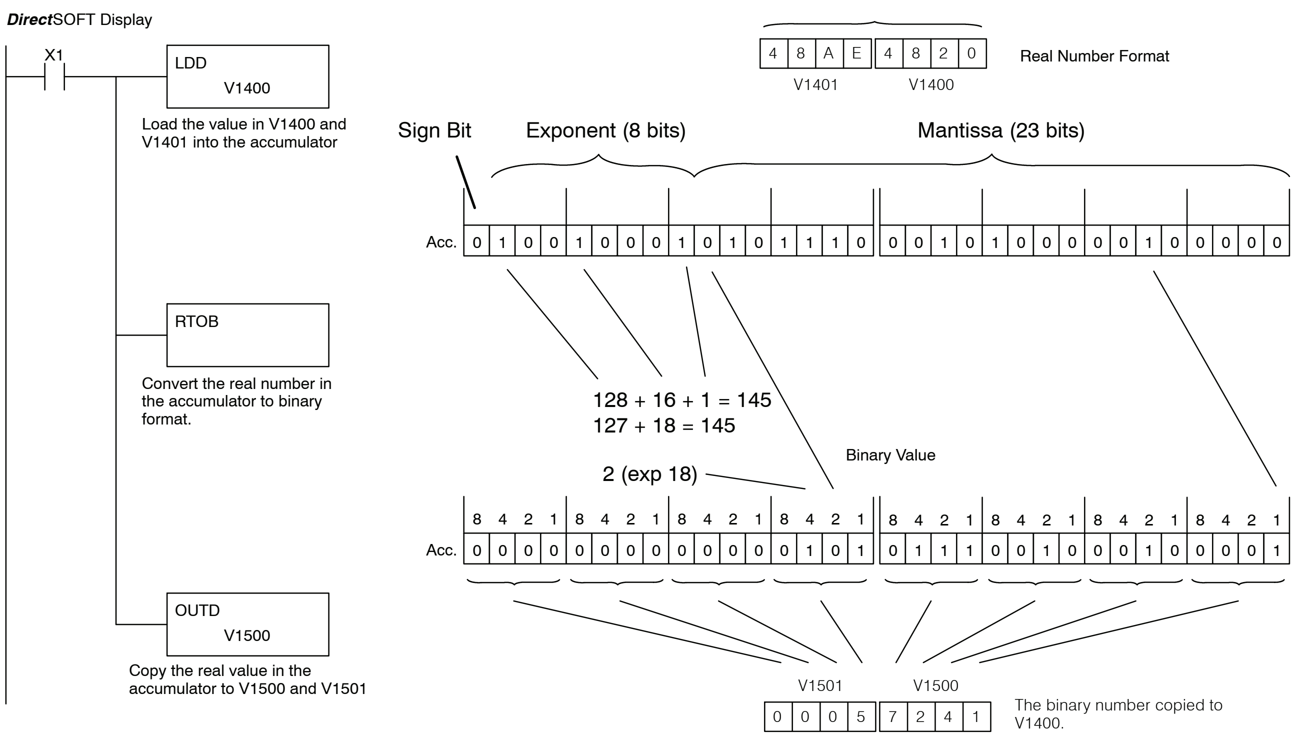
Real to Binary Conversion (RTOB)

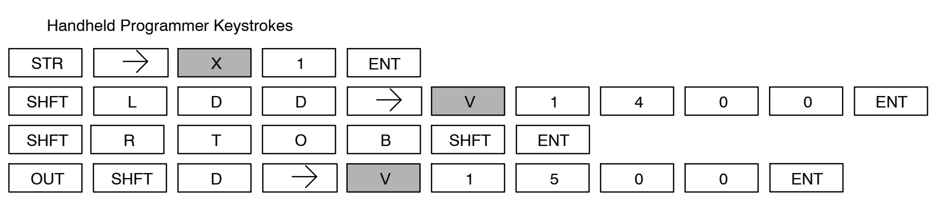
The Real-to-Binary instruction converts the real number in the accumulator to a binary value. The result resides in the accumulator. Both the binary and the real number may use all 32 bits of the accumulator.



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 number cannot be converted to binary.

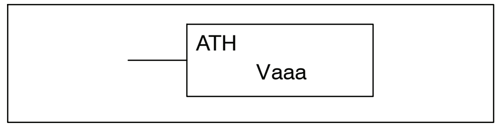
In the following example, when X1 is on, the value in V1400 and V1401 is loaded into the accumulator using the Load Double instruction. The RTOB instruction converts the real value in the accumulator the equivalent binary number format. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction. The handheld programmer would display the binary value in V1500 and V1501 as a HEX value.





ASCII to HEX (ATH)

The ASCII TO HEX instruction converts a table of ASCII values to a specified table of HEX values. ASCII values are two digits and their HEX equivalents are one digit.



This means an ASCII table of four V–memory locations would only require two V–memory locations for the equivalent HEX table. The function parameters are loaded into the accumulator stack and the accumulator by two additional instructions. Listed below are the steps necessary to program an ASCII to HEX table function. The example on the following page shows a program for the ASCII to HEX table function.

Step 1: — Load the number of V–memory locations for the ASCII table into the first level of the accumulator stack.

Step 2: — Load the starting V–memory location for the ASCII table into the accumulator. This parameter must be a HEX value.

Step 3: — Specify the starting V–memory location (Vaaa) for the HEX table in the ATH instruction.

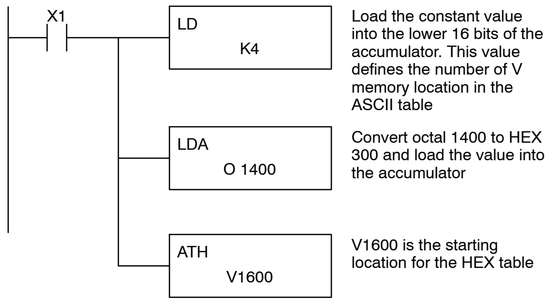
Helpful Hint: — For parameters that require HEX values when referencing memory locations, the LDA instruction can be used to convert an octal address to the HEX equivalent and load the value into the accumulator.

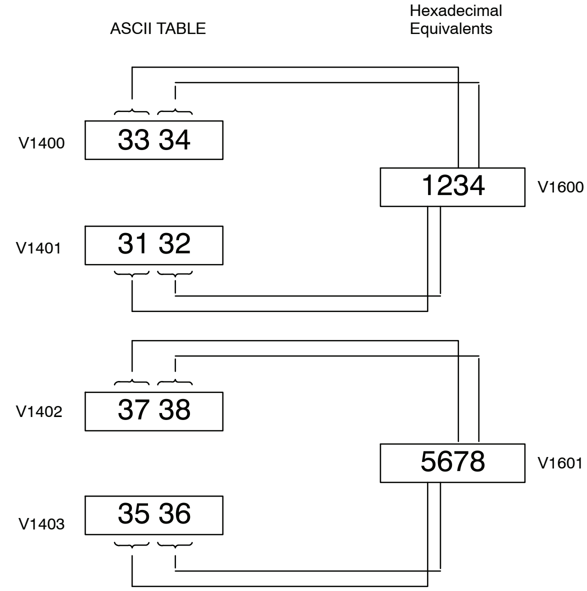
Operand Data Type		DL350 Range
		aaa
V–memory	V	All (See p. 3–29)

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

***Direct*SOFT Display**



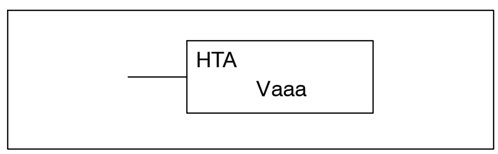


Handheld Programmer Keystrokes

STR	→	X	1	ENT						
SHFT	L	D	→	K	4	ENT				
SHFT	L	D	A	→	O	1	4	0	0	ENT
SHFT	A	T	H	→	V	1	6	0	0	ENT

HEX to ASCII (HTA)

The HEX to ASCII instruction converts a table of HEX values to a specified table of ASCII values. HEX values are one digit and their ASCII equivalents are two digits.



This means a HEX table of two V–memory locations would require four V–memory locations for the equivalent ASCII table. The function parameters are loaded into the accumulator stack and the accumulator by two additional instructions. Listed below are the steps necessary to program a HEX to ASCII table function. The example on the following page shows a program for the HEX to ASCII table function.

Step 1: — Load the number of V–memory locations in the HEX table into the first level of the accumulator stack.

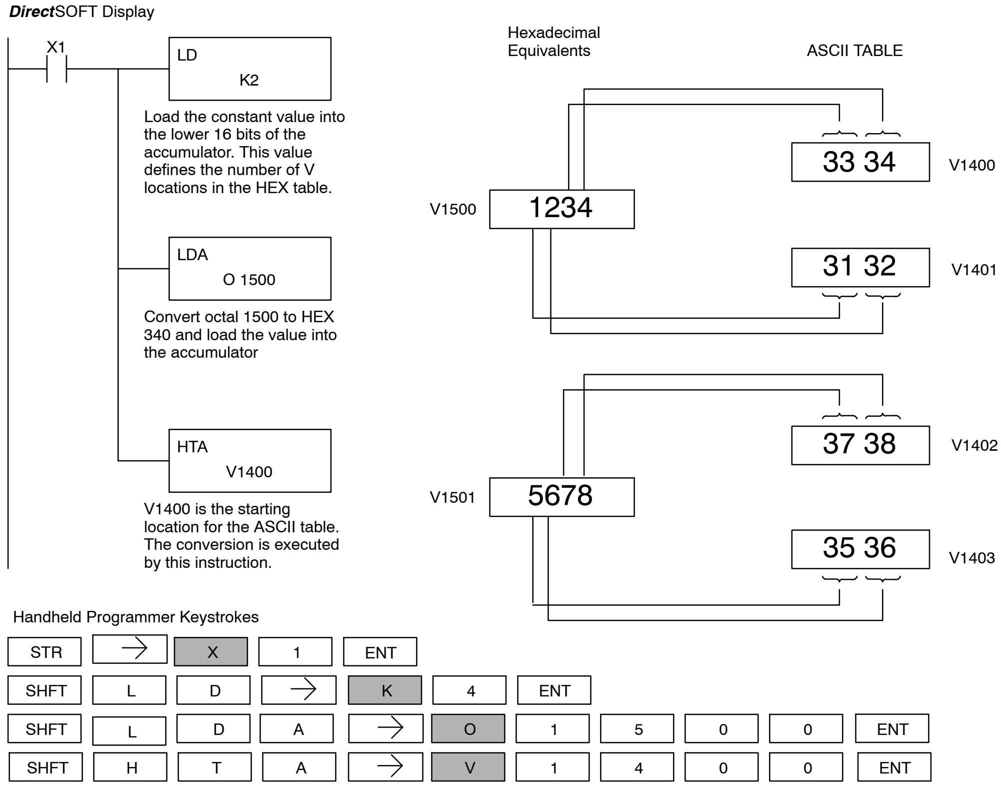
Step 2: — Load the starting V–memory location for the HEX table into the accumulator. This parameter must be a HEX value.

Step 3: — Specify the starting V–memory location (Vaaa) for the ASCII table in the HTA instruction.

Helpful Hint: — For parameters that require HEX values when referencing memory locations, the LDA instruction can be used to convert an octal address to the HEX equivalent and load the value into the accumulator.

Operand Data Type		DL350 Range
		aaa
V–memory	V	All (See p. 3–29)

In the following example, when X1 is ON the constant (K2) is loaded into the accumulator using the Load instruction. The starting location for the HEX table (V1500) is loaded into the accumulator using the Load Address instruction. The starting location for the ASCII table (V1400) is specified in the HEX to ASCII instruction.

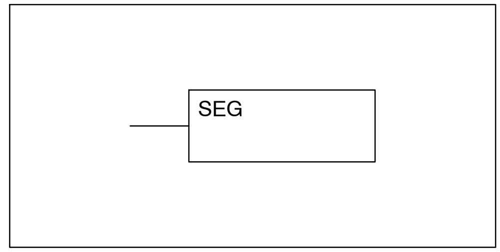


The table below lists valid ASCII values for HTA conversion.

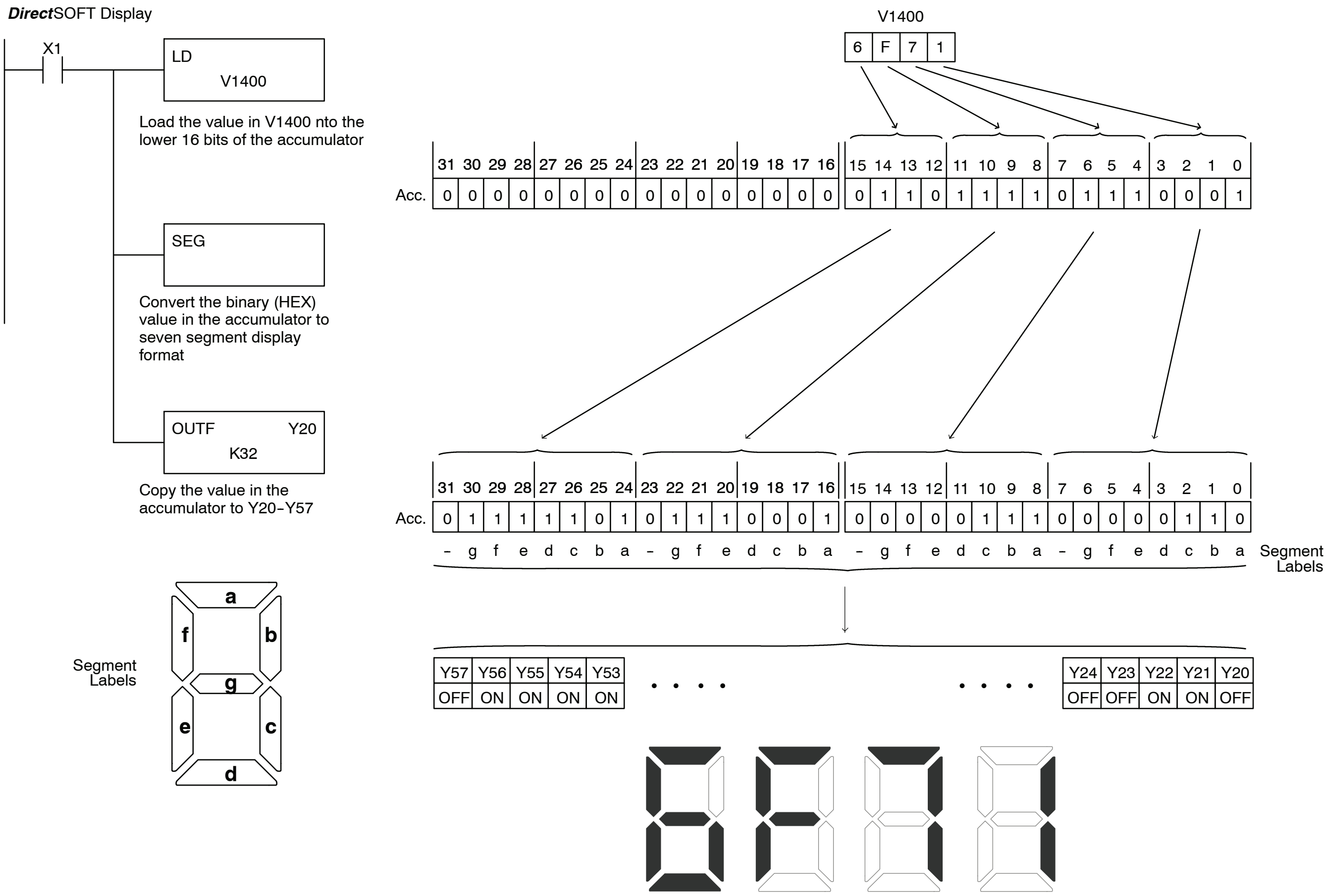
ASCII Values Valid for HTA Conversion			
Hex Value	ASCII Value	Hex Value	ASCII Value
0	30	8	38
1	31	9	39
2	32	A	41
3	33	B	42
4	34	C	43
5	35	D	44
6	36	E	45
7	37	F	46

Segment (SEG)

The BCD / Segment instruction converts a four digit HEX value in the accumulator to seven segment display format. The result resides in the accumulator.



In the following example, when X1 is on, the value in V1400 is loaded into the lower 16 bits of the accumulator using the Load instruction. The 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.



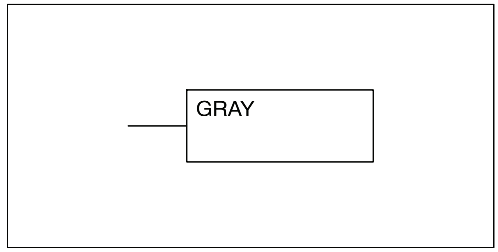
Handheld Programmer Keystrokes

STR	→	X	1	ENT						
L	D	→	V	1	4	0	0	ENT		
SHFT	S	SHFT	E	G	ENT					
OUT	SHFT	F	→	Y	2	0	→	K	3	2

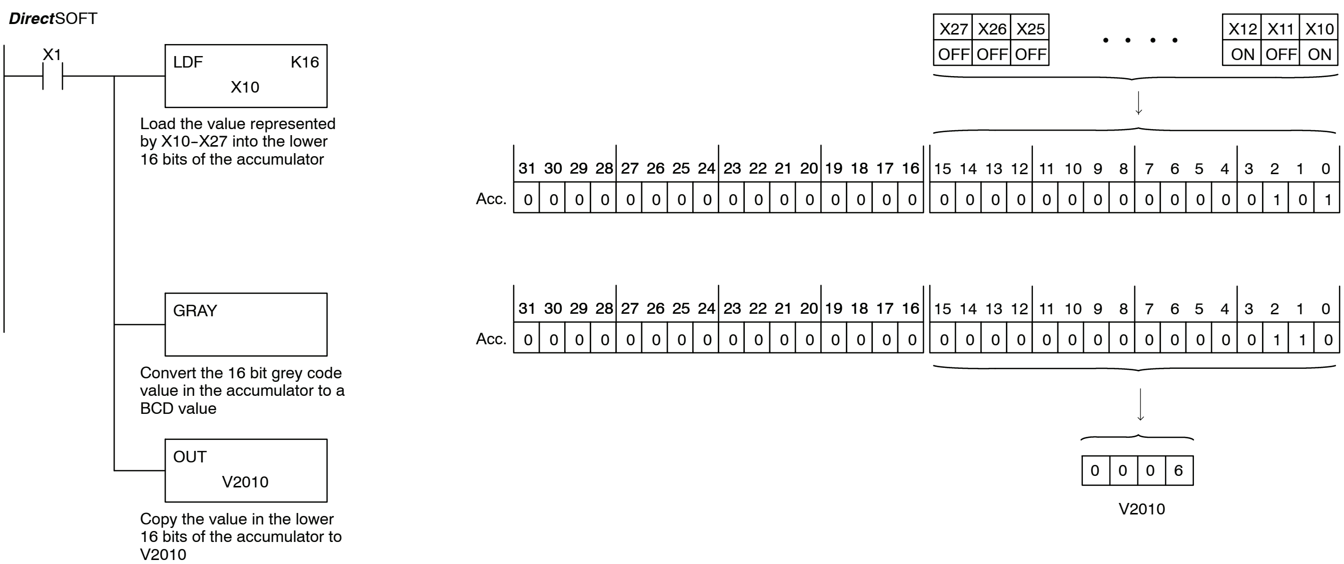
Standard RLL Instructions

Gray Code (GRAY)

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.



In the following example, when X1 is ON the binary value represented by X10–X27 is loaded into the accumulator using the Load Formatted instruction. The gray code value in the accumulator is converted to BCD using the Gray Code instruction. The value in the lower 16 bits of the accumulator is copied to V2010.



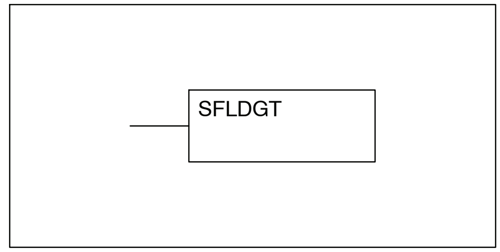
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT							
SHFT	L ANDST	D 3	F 5	→	B 1	A 0	→	B 1	G 6	ENT
SHFT	G 6	R ORN	A 0	Y MLS	ENT					
GX OUT	→	SHFT	V AND	C 2	A 0	B 1	A 0	ENT		

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)

The Shuffle Digits instruction shuffles a maximum of 8 digits rearranging them in a specified order. This function requires parameters to be loaded into the first level of the accumulator stack and the accumulator with two additional instructions. Listed below are the steps necessary to use the shuffle digit function. The example on the following page shows a program for the Shuffle Digits function.



Step 1:— Load the value (digits) to be shuffled into the first level of the accumulator stack.

Step 2:— Load the order that the digits will be shuffled to into the accumulator.

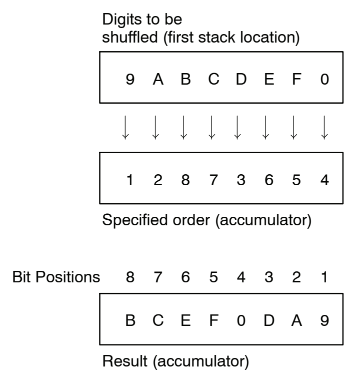
Note:— If the number used to specify the order contains a 0 or 9–F, the corresponding position will be set to 0.
See example on the next page.

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 on the next page.

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.

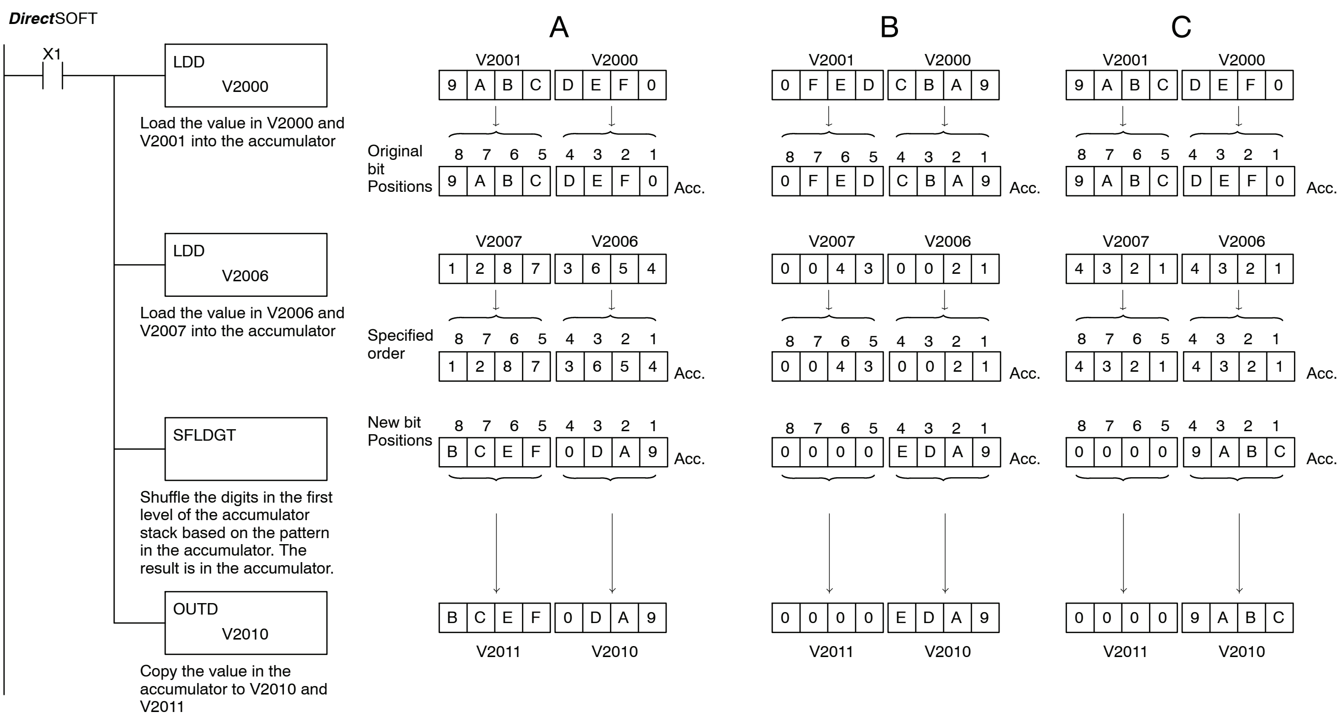


In the following example when X1 is on, The value in the first level of the accumulator stack will be reorganized in the order specified by the value in the accumulator.

Example A shows how the shuffle digits works when 0 or 9 -F is not used when specifying the order the digits are to be shuffled. Also, there are no duplicate numbers in the specified order.

Example B shows how the shuffle digits works when a 0 or 9–F is used when specifying the order the digits are to be shuffled. Notice when the Shuffle Digits instruction is executed, the bit positions in the first stack location that had a corresponding 0 or 9–F in the accumulator (order specified) are set to “0”.

Example C shows how the shuffle digits works when duplicate numbers are used specifying the order the digits are to be shuffled. Notice when the Shuffle Digits instruction is executed, the most significant duplicate number in the order specified is used in the result.



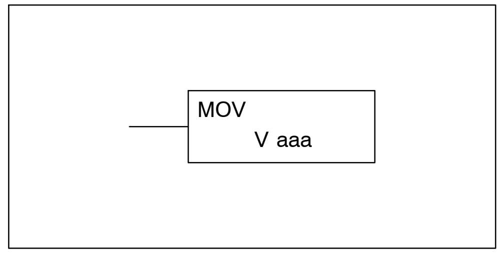
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	D 3	→	C 2	A 0	A 0	A 0	ENT
SHFT	L ANDST	D 3	D 3	→	C 2	A 0	A 0	G 6	ENT
SHFT	S RST	SHFT	F 5	L ANDST	D 3	G 6	T MLR	ENT	
GX OUT	SHFT	D 3	→	C 2	A 0	B 1	A 0	ENT	

Table Instructions

Move (MOV)

The Move instruction moves the 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.



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.

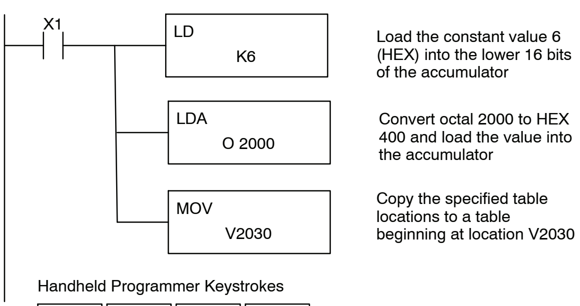
Step 2:— Load the starting V-memory location for the locations to be moved into the accumulator. This parameter must be a HEX value.

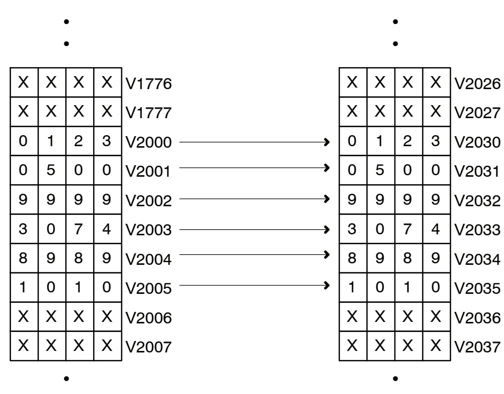
Step 3:— Insert the MOVE instruction which specifies starting V-memory location (Vaaa) for the destination table.

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		DL350 Range
		aaa
V-memory	V	All (See page 3-29)

In the following example, when X1 is on, the constant value (K6) is loaded into the accumulator using the Load instruction. This value specifies the length of the table and is placed in the first stack location after the Load Address instruction is executed. The octal address 2000 (V2000), the starting location for the source table is loaded into the accumulator. The destination table location (V2030) is specified in the Move instruction.



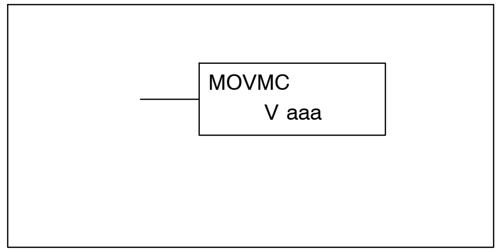


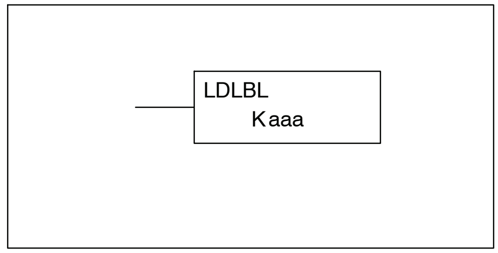
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	→	SHFT	K JMP	G 6	ENT		
SHFT	L ANDST	D 3	A 0	→	C 2	A 0	A 0	A 0	ENT
SHFT	M ORST	O INST#	V AND	→	C 2	A 0	D 3	A 0	ENT

Move Memory Cartridge / Load Label (MOVMC) (LDLBL)

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.

Step 1:— Load the number of words to be copied into the second level of the accumulator stack.

Step 2:— Load the offset for the data label area in the program ladder memory and the beginning of the V–memory block into the first level of the accumulator stack.

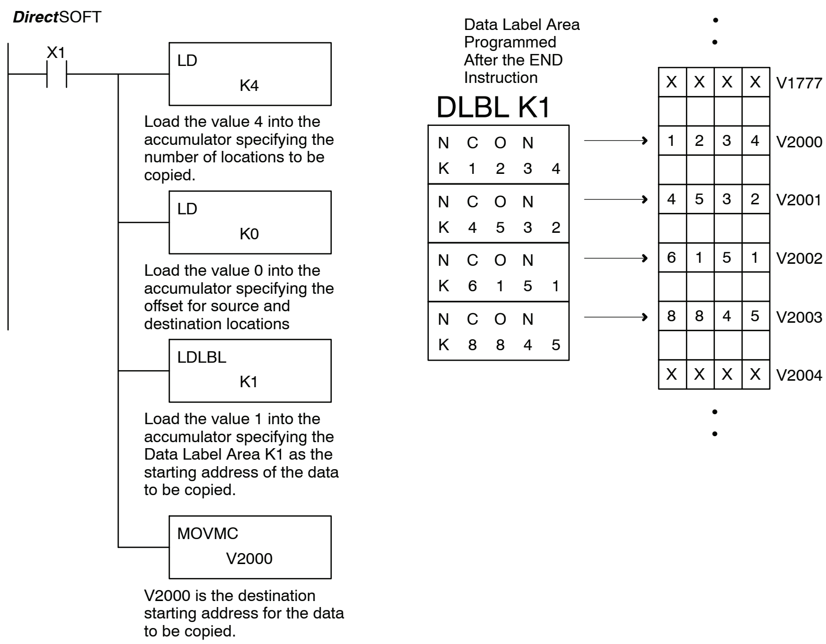
Step 3:— Load the *source data label* (LDLBL Kaaa) into the accumulator when copying data from ladder memory to V–memory. Load the *source address* into the accumulator when copying data from V–memory to ladder memory. This is where the value will be copied from. If the source address is a V–memory location, the value must be entered in HEX.

Step 4:— Insert the MOVMC instruction which specifies destination (Aaaa). This is where the value will be copied to.

Operand Data Type		DL350 Range
	A	aaa
V–memory	V	All (See page 3–29)

Copy Data From a Data Label Area to V-Memory

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 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 destination data, 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 starting location and executes the copying of data from the Data Label Area to V-memory.



Handheld Programmer Keystrokes

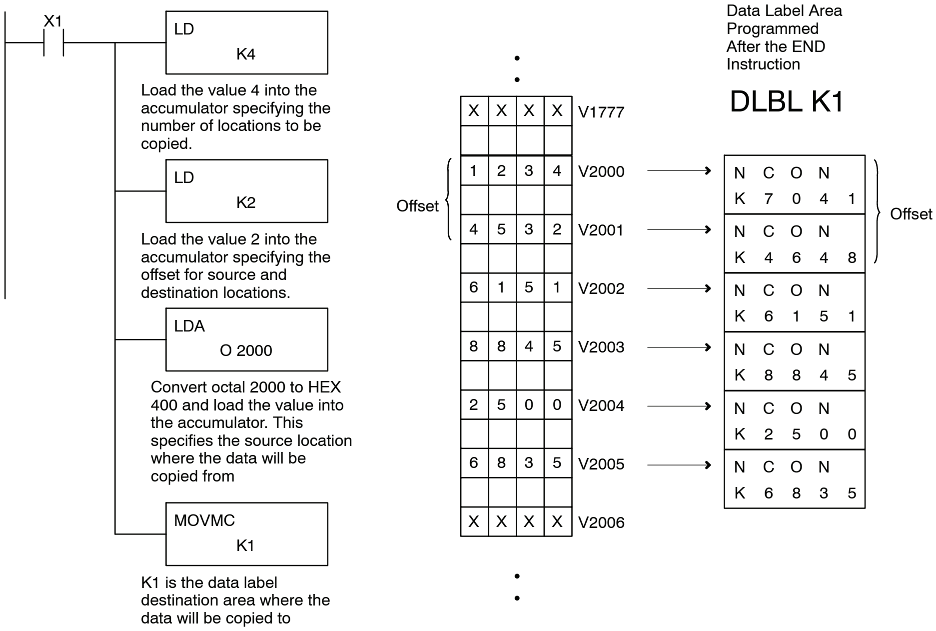
$ STR	→	B 1	ENT								
SHFT	L ANDST	D 3	→	SHFT	K JMP	E 4	ENT				
SHFT	L ANDST	D 3	→	SHFT	K JMP	A 0	ENT				
SHFT	L ANDST	D 3	L ANDST	B 1	L ANDST	→	B 1	ENT			
SHFT	M ORST	O INST#	V AND	M ORST	C 2	→	C 2	A 0	A 0	A 0	ENT

Standard RLL Instructions

Copy Data From V–Memory to a Data Label Area

In the following 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 table and is placed in the second stack location after the next Load and Load Address instructions are executed. The constant value (K2) is loaded into the accumulator using the Load instruction. This value specifies the offset for the source *and* destination data, and is placed in the first stack location after the Load Address instruction is executed. The source address where 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



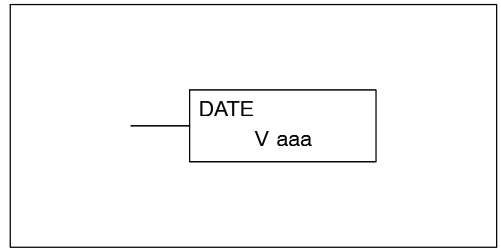
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT							
SHFT	L ANDST	D 3	→	SHFT	K JMP	E 4	ENT			
SHFT	L ANDST	D 3	→	SHFT	K JMP	C 2	ENT			
SHFT	L ANDST	D 3	A 0	→	C 2	A 0	A 0	A 0	ENT	
SHFT	M ORST	O INST#	V AND	M ORST	C 2	→	SHFT	K JMP	B 1	ENT

Clock / Calendar Instructions

Date (DATE)

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).



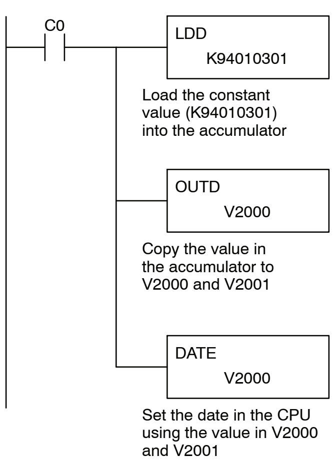
Date	Range	V Memory Location (BCD) (READ Only)
Year	0–99	V7774
Month	1–12	V7773
Day	1–31	V7772
Day of Week	0–06	V7771

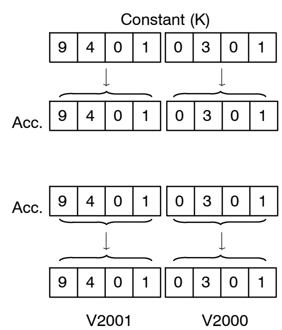
The values entered for the day of week are:
0=Sunday, 1=Monday, 2=Tuesday, 3=Wednesday, 4=Thursday, 5=Friday, 6=Saturday

Operand Data Type		DL350 Range
	A	aaa
V–memory	V	All (See p. 3–29)

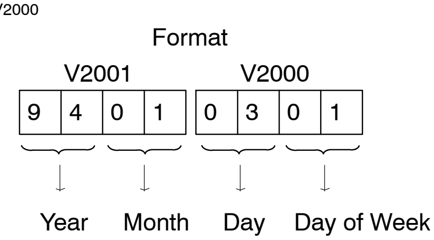
In the following example, when C0 is on, the constant value (K94010301) is loaded into the accumulator using the Load Double instruction (C0 should be a contact from a one shot (PD) instruction). The value in the accumulator is output to V2000 using the Out Double instruction. The Date instruction uses the value in V2000 to set the date in the CPU.

***Direct*SOFT Display**





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)

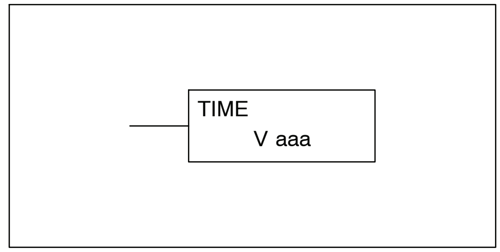


Handheld Programmer Keystrokes

STR	→	C	0	ENT							
SHFT	L	D	D	→	K	9	4	0	1		
0	3	0	1	ENT							
OUT	SHFT	D	→	V	2	0	0	0	ENT		
SHFT	D	A	T	E	→	V	2	0	0	0	ENT

Time (TIME)

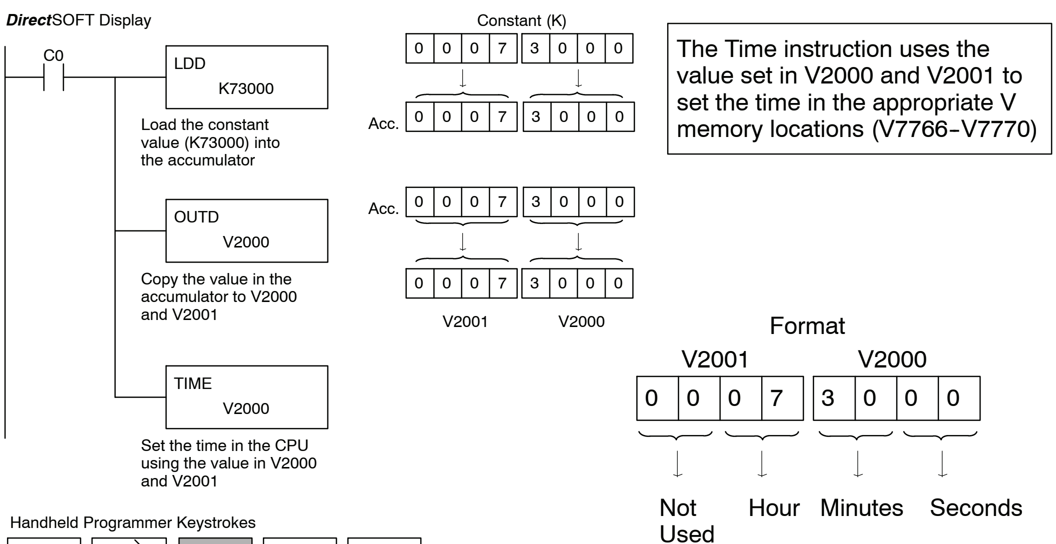
The Time instruction can be used to set the time (24 hour clock) in the CPU. The instruction requires two consecutive V–memory locations (Vaaa) which are used to *set the time*. If the values in the specified locations are not valid, the time will not be set. The current time can be read from memory locations V7747 and V7766–V7770.



Date	Range	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		DL350 Range
	A	aaa
V–memory	V	All (See p. 3–29)

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.



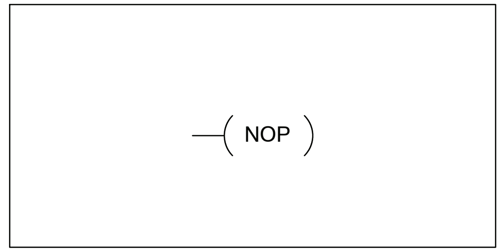
Handheld Programmer Keystrokes

STR	→	C	0	ENT							
SHFT	L	D	D	→	K	7	3	0	0	ENT	
OUT	SHFT	D	→	V	2	0	0	0	ENT		
SHFT	T	I	M	E	→	V	2	0	0	0	ENT

CPU Control Instructions

No Operation (NOP)

The No Operation is an empty (not programmed) memory location.



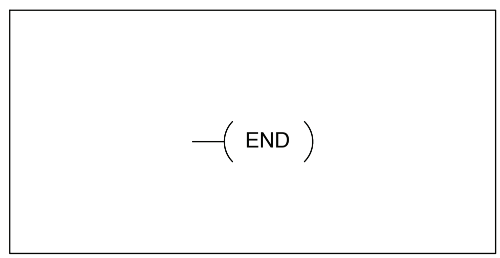
*Direct*SOFT

Handheld Programmer Keystrokes

SHFT	N TMR	O INST#	P CV	ENT

End (END)

The End instruction marks the termination point of the normal program scan. An End instruction is required at the end of the main program body. If the End instruction is omitted an error will occur and the CPU will not enter the Run Mode. Data labels, subroutines and interrupt routines are placed after the End instruction. The End instruction is not conditional; therefore, no input contact is allowed.



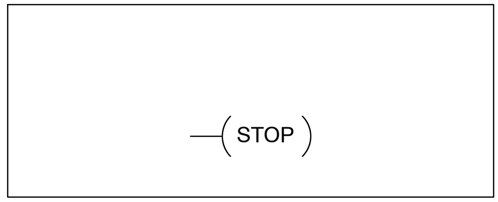
*Direct*SOFT

Handheld Programmer Keystrokes

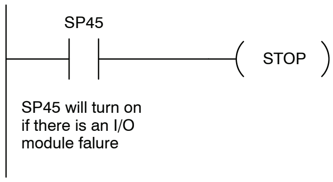
SHFT	E 4	N TMR	D 3	ENT

Stop (STOP)

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.



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.

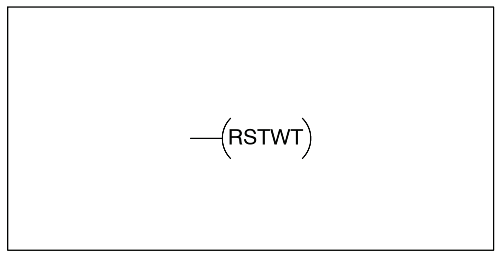


Handheld Programmer Keystrokes

$ STR	→	SHFT	SP STRN	E 4	F 5	ENT
SHFT	S RST	SHFT	T MLR	O INST#	P CV	ENT

Reset Watch Dog Timer (RSTWT)

The Reset Watch Dog Timer instruction resets the CPU scan timer. The default setting for the watch dog timer is 200ms. Scan times very seldom exceed 200ms, but it is possible. For/next loops, subroutines, interrupt routines, and table instructions can be programmed such that the scan becomes longer than 200ms. When instructions are used in a manner that could exceed the watch dog timer setting, this instruction can be used to reset the timer.



A software timeout error (E003) will occur and the CPU will enter the program mode if the scan time exceeds the watch dog timer setting. Placement of the RSTWT instruction in the program is very important. The instruction has to be executed before the scan time exceeds the watch dog timer's setting.

If the scan time is consistently longer than the watch dog timer's setting, the timeout value may be permanently increased from the default value of 200ms by AUX 55 on the HPP or the appropriate auxiliary function in your programming package. This eliminates the need for the RSTWT instruction.

In the following example the CPU scan timer will be reset to 0 when the RSTWT instruction is executed. See the For/Next instruction for a detailed example.

***Direct*SOFT**

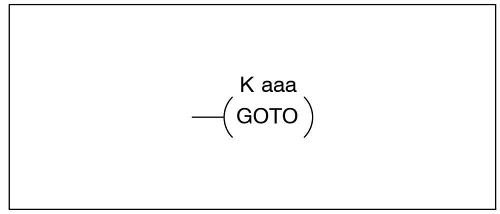
Handheld Programmer Keystrokes

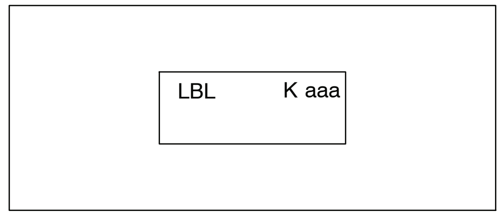
SHFT	R ORN	S RST	T MLR	W ANDN	T MLR	ENT

Program Control Instructions

Goto Label (GOTO) (LBL)

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.

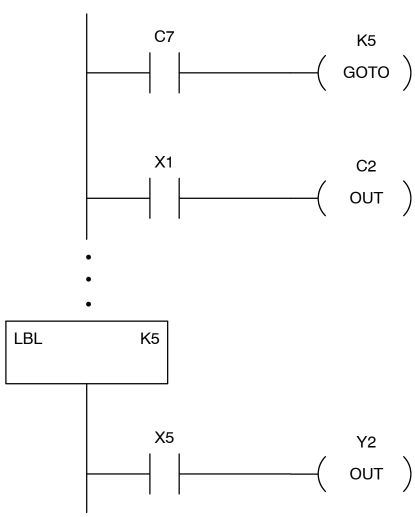




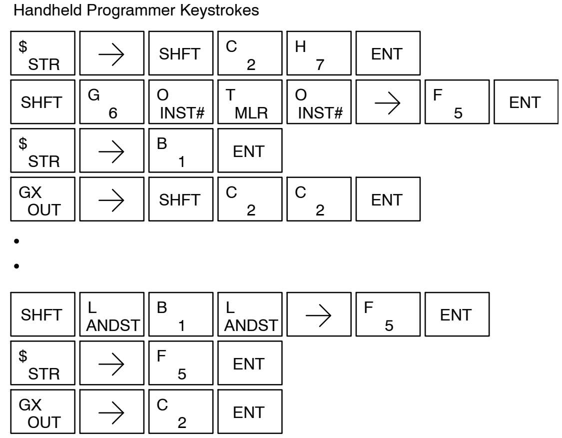
Operand Data Type	DL350 Range
	aaa
Constant K	1–FFFF

In the following example, when C7 is on, all the program logic between the GOTO and the corresponding LBL instruction (designated with the same constant Kaaa value) will be skipped. The instructions being skipped will not be executed by the CPU.

***Direct*SOFT**

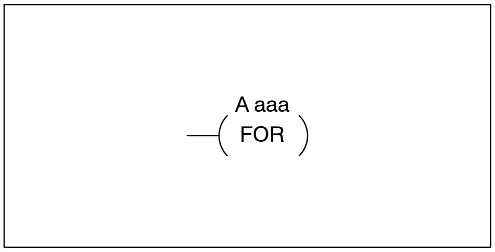


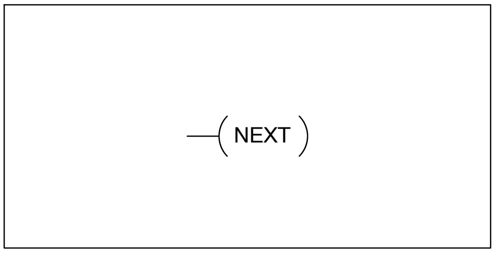
Handheld Programmer Keystrokes



For / Next (FOR) (NEXT)

The For and Next instructions are used to execute a section of ladder logic between the For and Next instruction a specified numbers of times. When the For instruction is enabled, the program will loop the specified number of times. If the For instruction is not energized the section of ladder logic between the For and Next instructions is not executed.



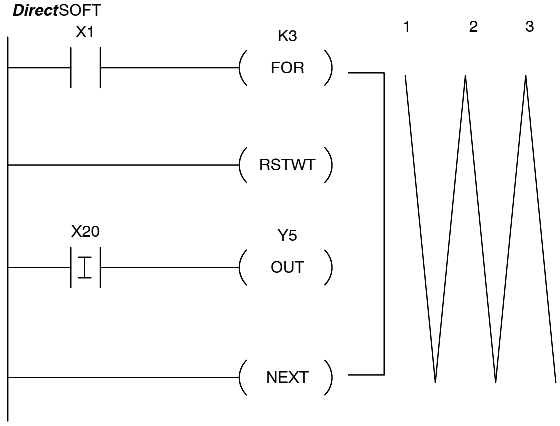


For / Next instructions cannot be nested. Up to 64 For / Next loops may be used in a program. If the maximum number of For / Next loops is exceeded, error E413 will occur. The normal I/O update and CPU housekeeping is suspended while executing the For / Next loop. The program scan can increase significantly, 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		DL350 Range
	A	**aaa**
V–memory	V	All (See page 3–29)
Constant	K	1–9999

In the following example, when X1 is on, the application program inside the For / Next loop will be executed three times. If X1 is off the program inside the loop will not be executed. The immediate instructions may or may not be necessary depending on your application. Also, The RSTWT instruction is not necessary if the For / Next loop does not extend the scan time larger the Watch Dog Timer setting. For more information on the Watch Dog Timer, refer to the RSTWT instruction.

***Direct*SOFT**



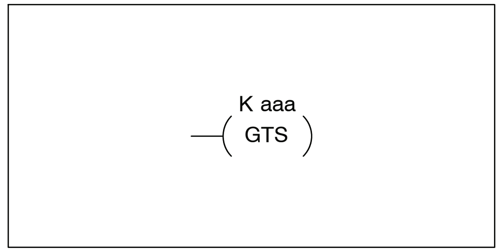
Handheld Programmer Keystrokes

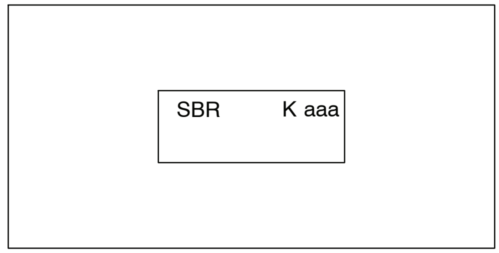
$ STR	→	B 1	ENT			
SHFT	F 5	O INST#	R ORN	→	D 3	ENT
SHFT	R ORN	S RST	T MLR	W ANDN	T MLR	ENT
$ STR	SHFT	I 8	→	C 2	A 0	ENT
GX OUT	→	F 5	ENT			
SHFT	N TMR	E 4	X SET	T MLR	ENT	

Goto Subroutine (GTS) (SBR)

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 128 GTS instructions and 64 SBR instructions used in a program. The GTS instructions can be nested up to 8 levels. An error E412 will occur if the maximum limits are exceeded. Typically this will be used in an application where a block of program logic may be slow to execute and is not required to execute every scan. The subroutine label and all associated logic is placed after the End statement in the program. When the subroutine is called from the main program, the CPU will execute the subroutine (SBR) with the same constant number (K) as the GTS instruction which called the subroutine.

By placing code in a subroutine it is only scanned and executed when needed since it resides after the End instruction. Code which is not scanned does not impact the overall scan time of the program.

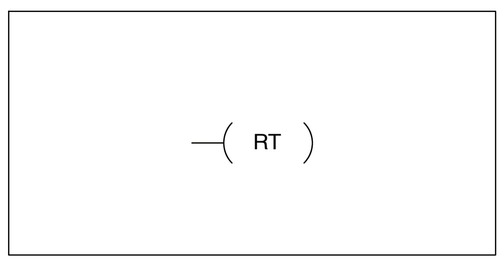




Operand Data Type		DL350 Range
		aaa
Constant	K	1–FFFF

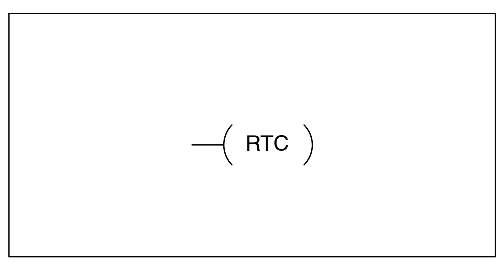
Subroutine Return (RT)

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).

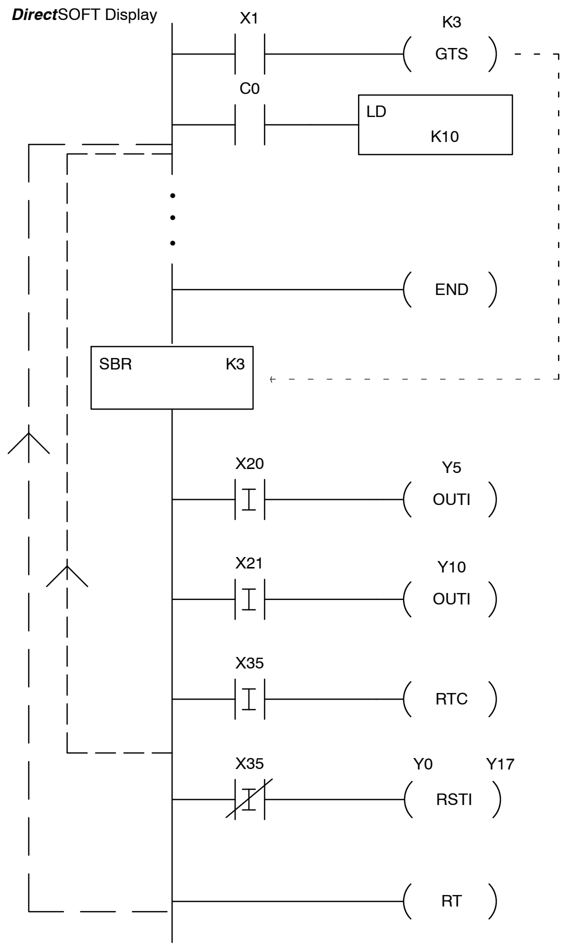


Subroutine Return Conditional (RTC)

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.



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.

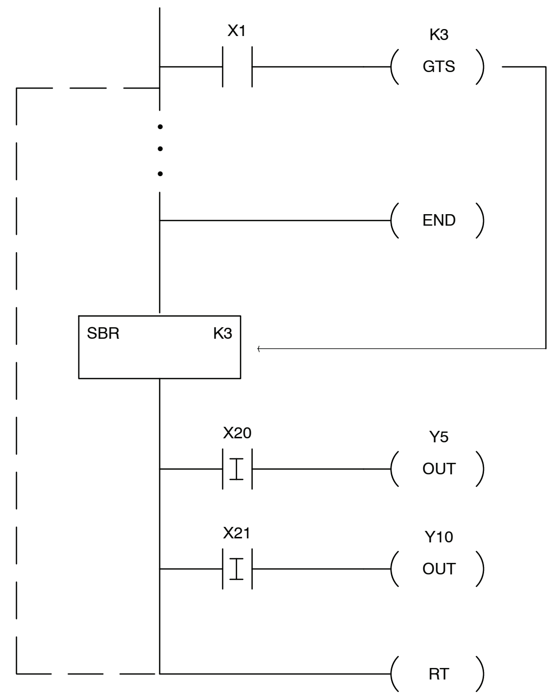


Handheld Programmer Keystrokes

STR	→	X	1	ENT					
SHFT	G	T	S	→	K	3	ENT		
:									
SHFT	E	N	D	ENT					
SHFT	S	SHFT	B	R	→	K	3	ENT	
STR	SHFT	I	→	X	2	0	ENT		
OUT	SHFT	I	→	Y	5	ENT			
STR	SHFT	I	→	X	2	1	ENT		
OUT	SHFT	I	→	Y	1	0	ENT		
STR	SHFT	I	→	X	3	5	ENT		
SHFT	R	T	C	ENT					
STRN	SHFT	I	→	X	3	5	ENT		
RST	SHFT	I	→	Y	0	Y	1	7	ENT
SHFT	R	T	ENT						

In the following example, when X1 is on, Subroutine K3 will be called. The CPU will jump to the Subroutine Label K3 and the ladder logic in the subroutine will be executed. The CPU will return to the main body of the program after the RT instruction is executed.

***Direct*SOFT**

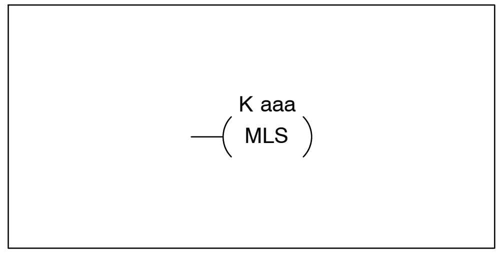


Handheld Programmer Keystrokes

$ STR	→	B 1	ENT				
SHFT	G 6	T MLR	S RST	→	D 3	ENT	
⋮							
SHFT	E 4	N TMR	D 3	ENT			
SHFT	S RST	SHFT	B 1	R ORN	→	D 3	ENT
$ STR	SHFT	I 8	→	C 2	A 0	ENT	
GX OUT	→	F 5	ENT				
$ STR	SHFT	I 8	→	C 2	B 1	ENT	
GX OUT	→	B 1	A 0	ENT			
SHFT	R ORN	T MLR	ENT				

Master Line Set (MLS)

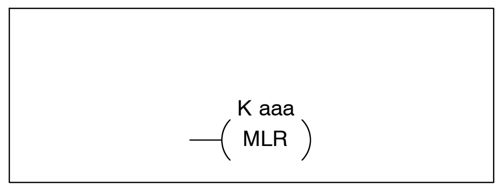
The Master Line Set instruction allows the program to control sections of ladder logic by forming a new power rail controlled by the main left power rail. The main left rail is always master line 0. When a MLS K1 instruction is used, a new power rail is created at level 1. Master Line Sets and Master Line Resets can be used to nest power rails up to seven levels deep. Note that unlike stages in RLL*PLUS*, the logic within the master control relays is still scanned and updated even though it will not function if the MLS is off.



Operand Data Type		DL350 Range
		aaa
Constant	K	1–7

Master Line Reset (MLR)

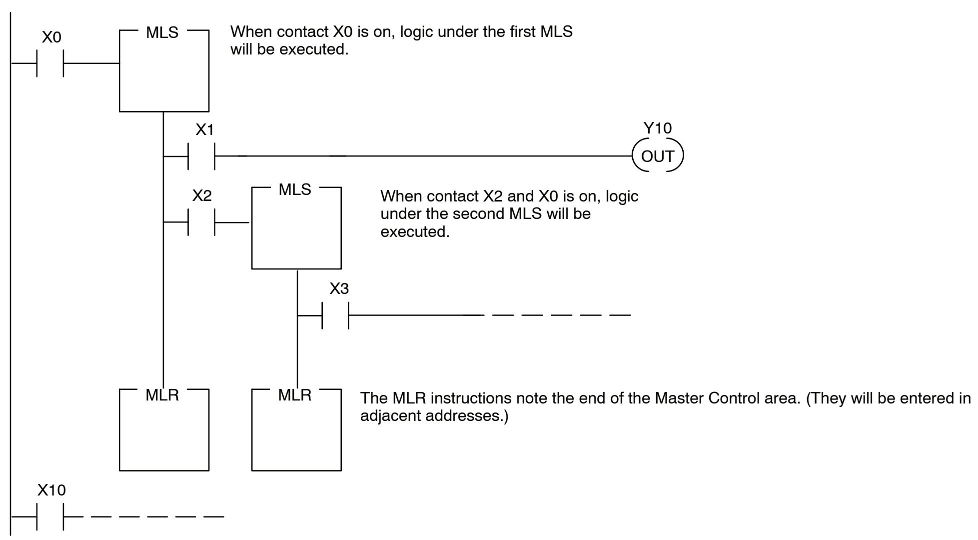
The Master Line Reset instruction marks the end of control for the corresponding MLS instruction. The MLR reference is one less than the corresponding MLS.



Operand Data Type		DL350 Range
		aaa
Constant	K	0–7

Understanding Master Control Relays

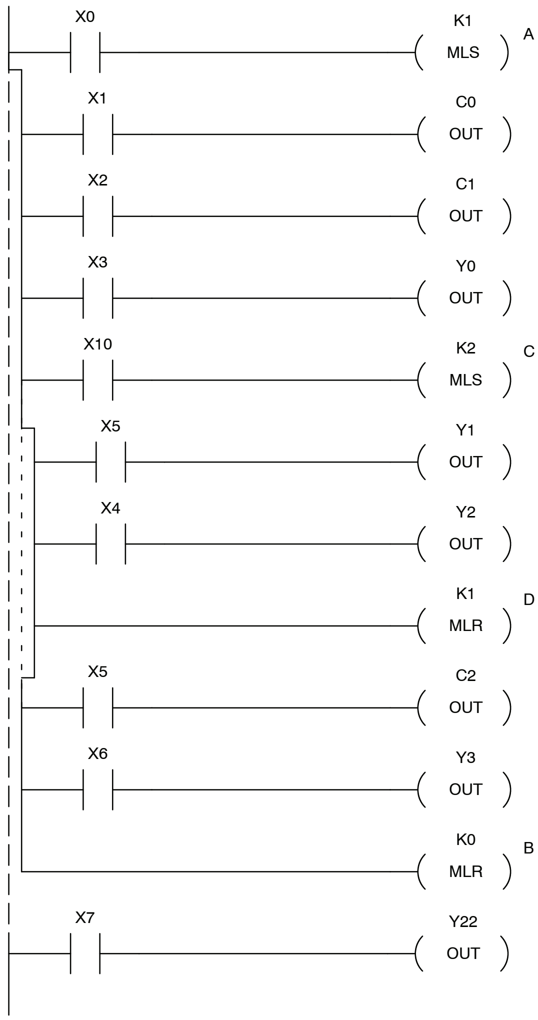
The Master Line Set (MLS) and Master Line Reset (MLR) instructions allow you to quickly enable (or disable) sections of the RLL program. This provides program control flexibility. The following example shows how the MLS and MLR instructions operate by creating a sub power rail for control logic.



MLS/MLR Example

In the following MLS/MLR example logic between the first MLS K1 (A) and MLR K0 (B) will function only if input X0 is on. The logic between the MLS K2 (C) and MLR K1 (D) will function only if input X10 and X0 is on. The last rung is not controlled by either of the MLS coils.

*Direct*SOFT



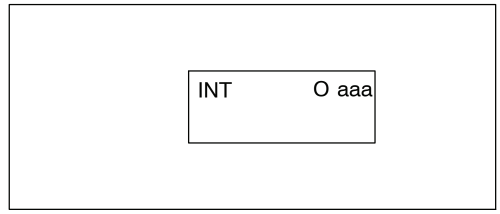
Handheld Programmer Keystrokes

$ STR	→	A 0	ENT		
Y MLS	→	B 1	ENT		
$ STR	→	B 1	ENT		
GX OUT	→	SHFT	C 2	A 0	ENT
$ STR	→	C 2	ENT		
GX OUT	→	SHFT	C 2	B 1	ENT
$ STR	→	D 3	ENT		
GX OUT	→	A 0	ENT		
$ STR	→	B 1	A 0	ENT	
Y MLS	→	C 2	ENT		
$ STR	→	F 5	ENT		
GX OUT	→	B 1	ENT		
$ STR	→	E 4	ENT		
GX OUT	→	C 2	ENT		
T MLR	→	B 1	ENT		
$ STR	→	F 5	ENT		
GX OUT	→	SHFT	C 2	C 2	ENT
$ STR	→	G 6	ENT		
GX OUT	→	D 3	ENT		
T MLR	→	A 0	ENT		
$ STR	→	H 7	ENT		
GX OUT	→	C 2	C 2	ENT	

Interrupt Instructions

Interrupt (INT)

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 by external interrupts via the counter interface module which provides 4 interrupts.



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. Interrupt module interrupts are labeled in octal to correspond with the hardware input signal (X1 will initiate interrupt INT1). There is only one software interrupt and it is labeled INT 0. The program execution will continue from where it was before the interrupt occurred once the interrupt is serviced.

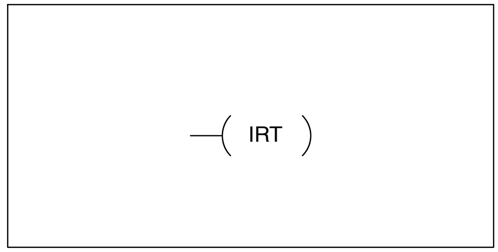
The software interrupt is setup by programming the interrupt time in V7634. The valid range is 3–999 ms. The value must be a BCD value. The interrupt will not execute if the value is out of range.

NOTE: See the example program of a software interrupt.

Operand Data Type		DL350 Range
		aaa
Constant	O	0–3

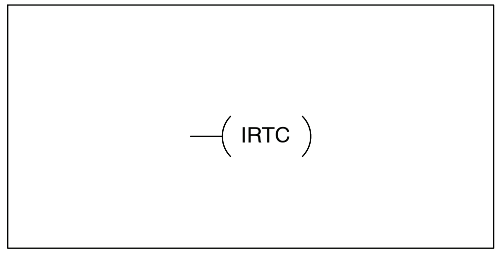
Interrupt Return (IRT)

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).



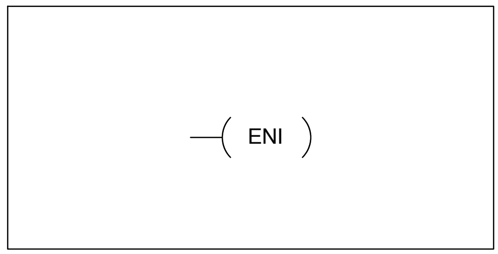
Interrupt Return Conditional (IRTC)

The Interrupt Return Conditional instruction is a optional instruction used with an input contact to implement a condtional return from the interrupt routine. The Interrupt Return is required to terminate the interrupt routine.



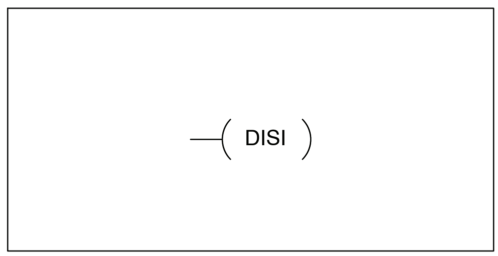
Enable Interrupts (ENI)

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.



Disable Interrupts (DISI)

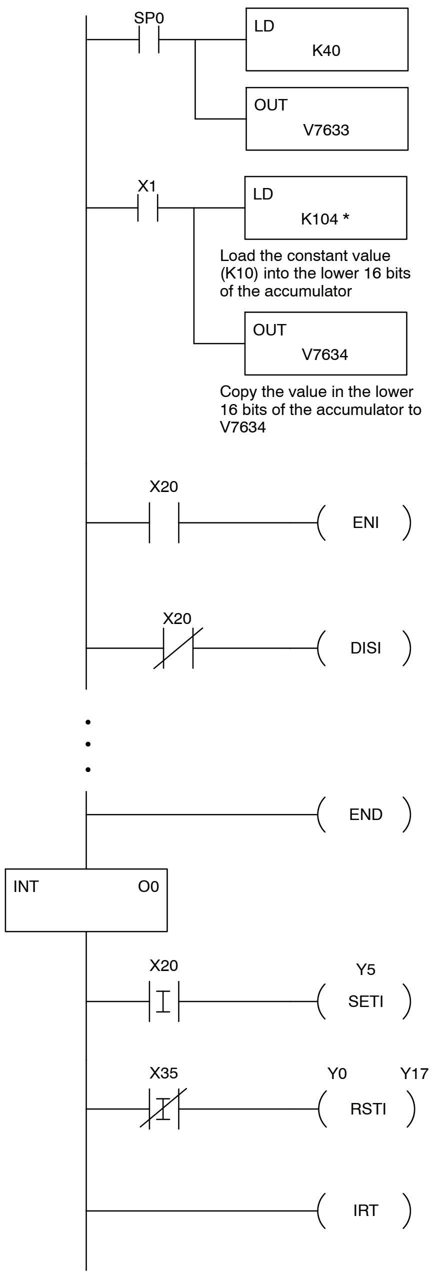
The Disable Interrupt instruction is programmed in the main body of the application program (before the End instruction) to disable both hardware or software interrupts. Once the coil has been energized interrupts will be disabled until the interrupt is enabled by the Enable Interrupt instruction.



Interrupt Example for Software Interrupt

In the following example, when X1 is on, the value 10 is copied to V7634. This value sets the software interrupt to 10 ms. When X20 turns on, the interrupt will be enabled. When X20 turns off, the interrupt will be disabled. Every 10 ms the CPU will jump to the interrupt label INT O 0. The application ladder logic in the interrupt routine will be performed. If X35 is not on Y0–Y17 will be reset to off and then the CPU will return to the main body of the program.

***Direct*SOFT**



Handheld Programmer Keystrokes

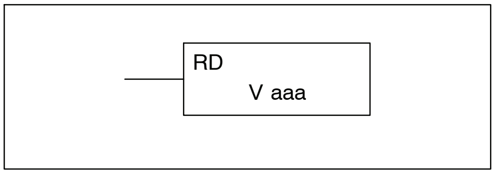
$ STR	→	B 1	ENT						
SHFT	L ANDST	D 3	→	SHFT	K JMP	B 4	A 0	ENT	
GX OUT	→	SHFT	V AND	H 7	G 6	D 3	E 4	ENT	
$ STR	→	C 2	A 0	ENT					
SHFT	E 4	N TMR	I 8	ENT					
SP STRN	→	C 2	A 0	ENT					
SHFT	D 3	I 8	S RST	I 8	ENT				
•									
•									
SHFT	E 4	N TMR	D 3	ENT					
SHFT	I 8	N TMR	T MLR	→	A 0	ENT			
$ STR	SHFT	I 8	→	C 2	A 0	ENT			
X SET	SHFT	I 8	→	F 5	ENT				
SP STRN	SHFT	I 8	→	D 3	F 5	ENT			
X SET	SHFT	I 8	→	B 1	A 0	→	B 1	H 7	ENT
SHFT	I 8	R ORN	T MLR	ENT					

* The value entered, 0–999 must be followed by the digit 4 to complete the instruction.

Intelligent I/O Instructions

Read from Intelligent Module (RD)

The Read from Intelligent Module instruction reads a block of data (1-128 bytes maximum) from an intelligent I/O module into the CPU's V-memory. It loads the function parameters into the first and second level of the accumulator stack, and the accumulator by three additional instructions.



Listed below are the steps to program the Read from Intelligent module function.

Step 1: — Load the base number (0-3) into the first byte and the slot number (0-7) into the second byte of the second level of the accumulator stack.

Step 2: — Load the number of bytes to be transferred into the first level of the accumulator stack. (maximum of 128 bytes)

Step 3: — Load the address from which the data will be read into the accumulator. This parameter must be a HEX value.

Step 4: — Insert the RD instruction which specifies the starting V-memory location (Vaaa) where the data will be read into.

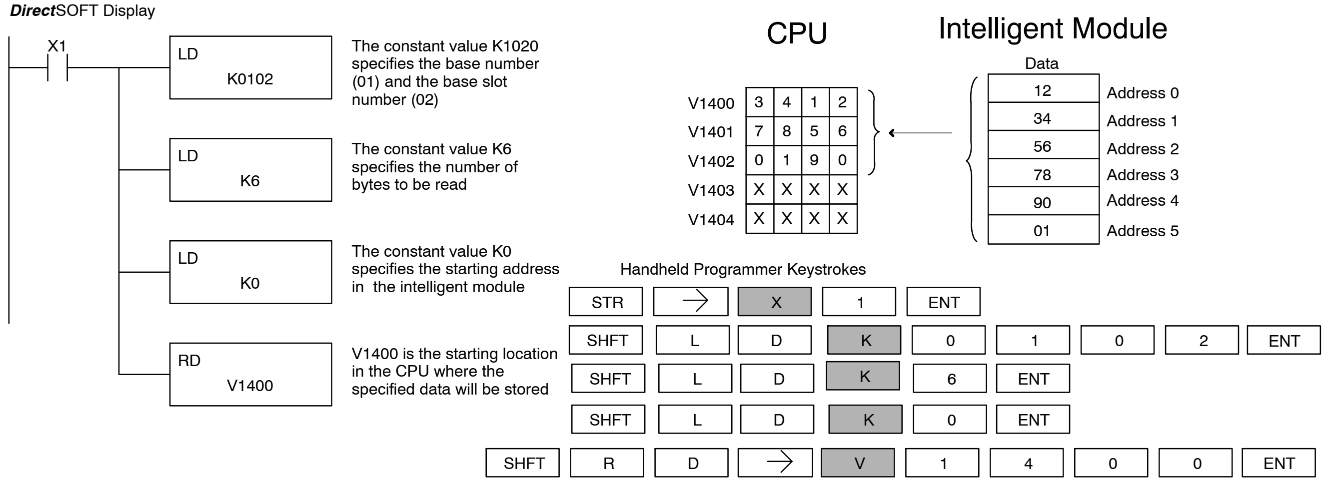
Helpful Hint: —Use the LDA instruction to convert an octal address to its HEX equivalent and load it into the accumulator when the hex format is required.

Operand Data Type		DL350 Range
		aaa
V-memory	V	All (See p. 3-29)

Discrete Bit Flags	Description
SP54	on when RX, WX, RD, WT instructions are executed with the wrong parameters.

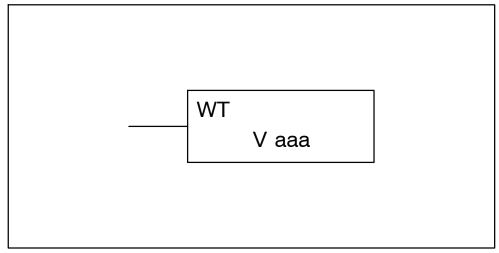
NOTE: Status flags are valid only until another instruction uses the same flag.

In the following example when X1 is on, the RD instruction will read six bytes of data from a intelligent module in base 1, slot 2 starting at address 0 in the intelligent module and copy the information into V-memory locations V1400-V1402.



Write to Intelligent Module (WT)

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.



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.

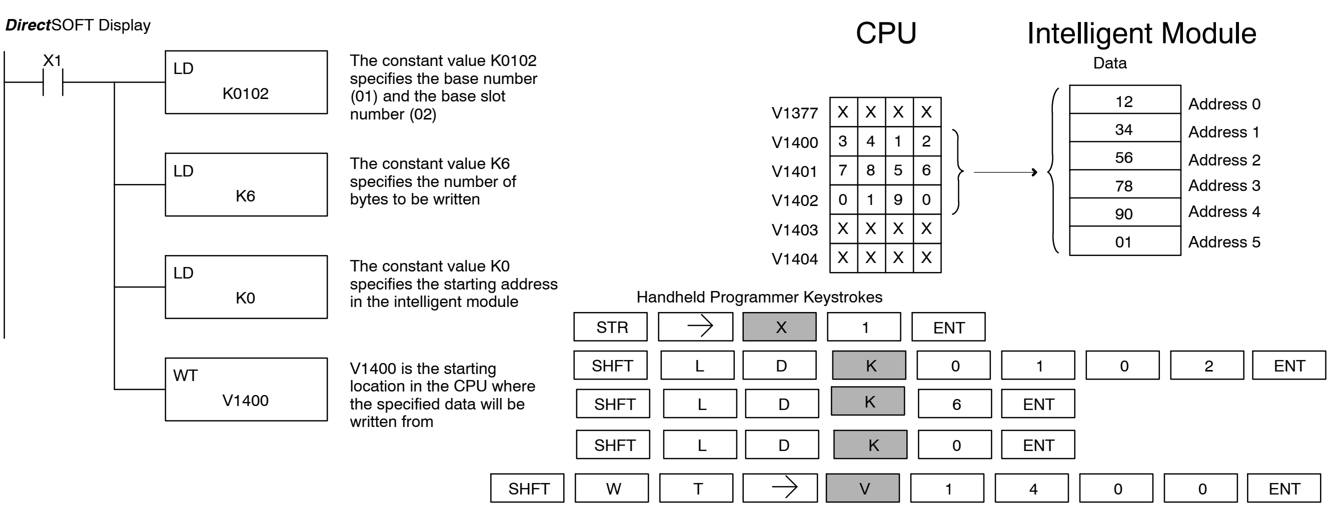
Helpful Hint: —Use the LDA instruction to convert an octal address to its HEX equivalent and load it into the accumulator when the hex format is required.

Operand Data Type		DL350 Range
		aaa
V–memory	V	All (See p. 3–29)

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.

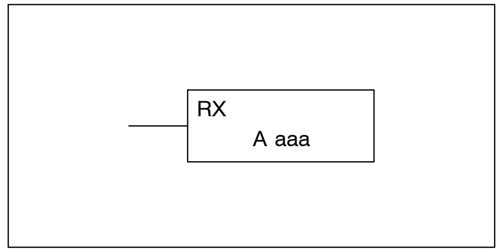
In the following example, when X1 is on, the WT instruction will write six bytes of data to an intelligent module in base 1, slot 2 starting at address 0 in the intelligent module and copy the information from V–memory locations V1400–V1402.



Network Instructions

Read from Network (RX)

The Read from Network instruction is used by the master device 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.



Replace Step 1 with the following: Step 1: - Load the slave address (0-90 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. When using Port 2 of the CPU, the formatting should be Kf1xx where xx is the slave address (0-90 BCD).

Step 1: — Load the slave address (0–90 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.

Step 2: — Load the number of bytes to be transferred into the first level of the accumulator stack.

Step 3: — Load the address of the data to be read into the accumulator. This parameter requires a HEX value.

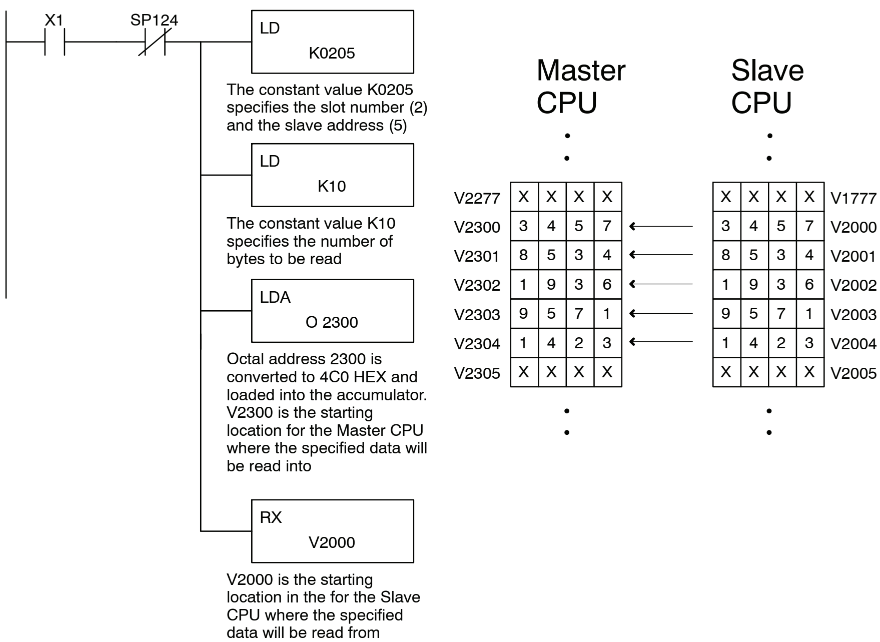
Step 4: — Insert the RX instruction which specifies the starting V memory location (Aaaa) where the data will be read from in the slave.

Helpful Hint: — For parameters that require HEX values, the LDA instruction can be used to convert an octal address to the HEX equivalent and load the value into the accumulator.

Operand Data Type		DL350 Range
	A	aaa
V–memory	V	All (See page 3–29)
Pointer	P	All V mem. (See page 3–29)
Inputs	X	0–777
Outputs	Y	0–777
Control Relays	C	0–1777
Stage	S	0–1777
Timer	T	0–377
Counter	CT	0–177
Special Relay	SP	0–777
Program Memory	$	0–7679 (7.5K program mem.) 0–15873 (15.5K program mem.)

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 (V2000 – V2004) will be read from a CPU at station address 5 and copied into V–memory locations V2300–V2304 in the CPU with the master DCM.

***Direct*SOFT**

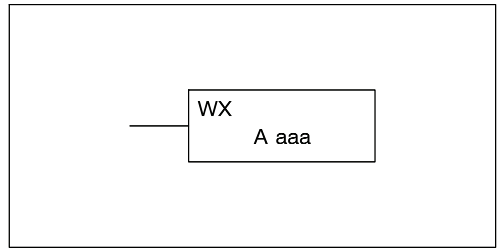


Handheld Programmer Keystrokes

$ STR	→	B 1	ENT						
W ANDN	→	SHFT	SP STRN	B 1	C 2	E 4	ENT		
SHFT	L ANDST	D 3	→	SHFT	K JMP	C 2	A 0	F 5	ENT
SHFT	L ANDST	D 3	→	SHFT	K JMP	B 1	A 0	ENT	
SHFT	L ANDST	D 3	A 0	→	C 2	D 3	A 0	A 0	ENT
SHFT	R ORN	X SET	→	C 2	A 0	A 0	A 0	ENT	

Write to Network (WX)

The Write to Network instruction is used to write a block of data from the master device to a slave device on the same network. The function parameters are loaded into the 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.



Replace Step 1 with the following:
Step 1: - Load the slave address (0-90 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. When using Port 2 of the CPU, the formatting should be Kf1xx where xx is the slave address (0-90 BCD).

Step 1: — Load the slave address (0–90 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.

Step 2: — Load the number of bytes to be transferred into the first level of the accumulator stack.

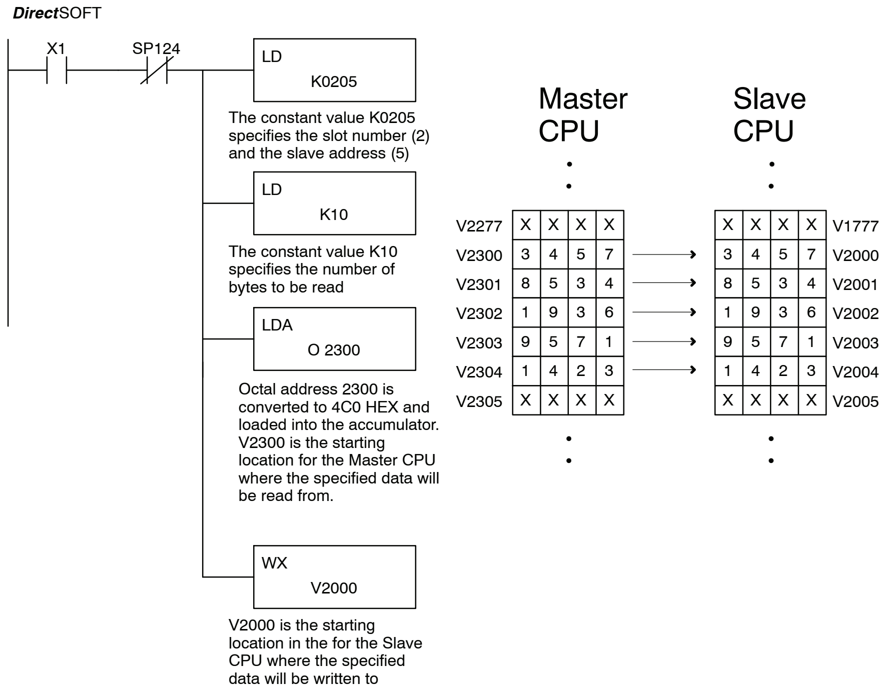
Step 3: — Load the address of the data in the master that is to be written to the network into the accumulator. This parameter requires a HEX value.

Step 4: — Insert the WX instruction which specifies the starting V–memory location (Aaaa) where the data will be written to the slave.

Helpful Hint: — For parameters that require HEX values, the LDA instruction can be used to convert an octal address to the HEX equivalent and load the value into the accumulator.

Operand Data Type		DL350 Range
	A	aaa
V–memory	V	All (See page 3–29)
Pointer	P	All V mem. (See page 3–29)
Inputs	X	0–777
Outputs	Y	0–777
Control Relays	C	0–1777
Stage	S	0–1777
Timer	T	0–377
Counter	CT	0–177
Special Relay	SP	0–777
Program Memory	$	0–7679 (7.5K program mem.) 0–15873 (15.5K program mem.)

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. 10 consecutive bytes of data is read from the CPU at station address 5 and copied to V-memory locations V2000–V2004 in the slave CPU.



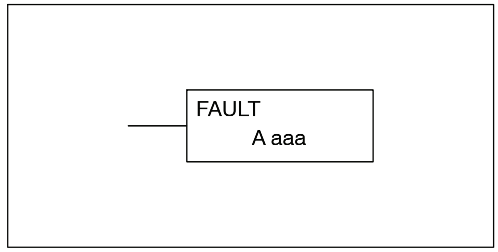
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT								
W ANDN	→	SHFT	SP STRN	B 1	C 2	E 4	ENT				
SHFT	L ANDST	D 3	→	SHFT	K JMP	C 2	A 0	F 5	ENT		
SHFT	L ANDST	D 3	→	SHFT	K JMP	B 1	A 0	ENT			
SHFT	L ANDST	D 3	A 0	→	SHFT	O INST#	C 2	D 3	A 0	A 0	ENT
SHFT	W ANDN	X SET	→	SHFT	V AND	C 2	A 0	A 0	A 0	ENT	

Message Instructions

Fault (FAULT)

The Fault instruction is used to display a message on the handheld programmer or ***Direct***SOFT. The message has a maximum of 23 characters and can be either V–memory data, numerical constant data or ASCII text.



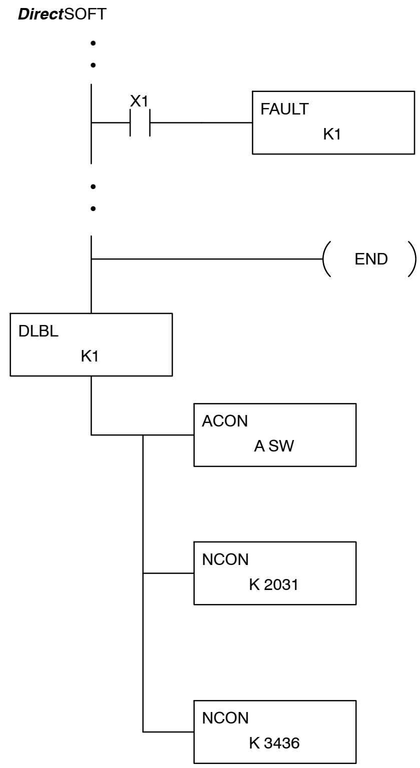
To display the value in a V–memory location, specify the V–memory location in the instruction. To display the data in ACON (ASCII constant) or NCON (Numerical constant) instructions, specify the constant (K) value for the corresponding data label area.

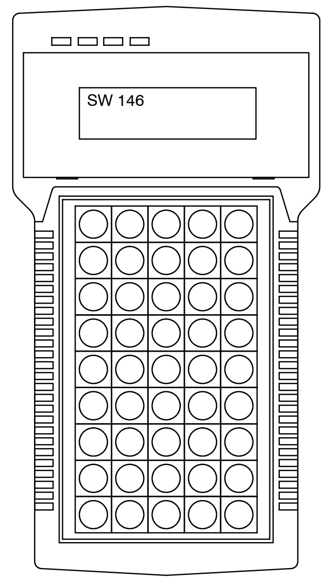
Operand Data Type		DL350 Range
	A	aaa
V–memory	V	All (See page 3–29)
Constant	K	1–FFFF

NOTE: The FAULT instruction takes a considerable amount of time to execute. This is because the FAULT parameters are stored in EEPROM. Make sure you consider the instructions execution times (shown in Appendix C) if you are attempting to use the FAULT instructions in applications that require faster than normal execution cycles.

Fault Example

In the following example when X1 is on, the message SW 146 will display on the handheld programmer. The NCONs use the HEX ASCII equivalent of the text to be displayed. (The HEX ASCII for a blank is 20, a 1 is 31, 4 is 34 ...)





Handheld Programmer Keystrokes

$ STR	→	B 1	ENT							
SHFT	F 5	A 0	U ISG	L ANDST	T MLR	→	B 1	ENT		

•
•

SHFT	E 4	N TMR	D 3	ENT						
SHFT	D 3	L ANDST	B 1	L ANDST	→	B 1	ENT			
SHFT	A 0	C 2	O INST#	N TMR	→	S RST	W ANDN	ENT		
SHFT	N TMR	C 2	O INST#	N TMR	→	C 2	A 0	D 3	B 1	ENT
SHFT	N TMR	C 2	O INST#	N TMR	→	D 3	E 4	D 3	G 6	ENT

Data Label (DLBL)

The Data Label instruction marks the beginning of an ASCII / numeric data area. DLBLs are programmed after the End statement. A maximum of 64 DLBL instructions can be used in a DL350 program. Multiple NCONs and ACONs can be used in a DLBL area.

```
DLBL      K aaa
```

Operand Data Type		DL350 Range
		aaa
Constant	K	1–FFFF

ASCII Constant (ACON)

The ASCII Constant instruction is used with the DLBL instruction to store ASCII text for use with other instructions. Two ASCII characters can be stored in an ACON instruction. If only one character is stored in a ACON a leading space will be printed in the Fault message.

```
ACON
     A aaa
```

Operand Data Type		DL350 Range
		aaa
ASCII	A	0–9 A–Z

Numerical Constant (NCON)

The Numerical Constant instruction is used with the DLBL instruction to store the HEX ASCII equivalent of numerical data for use with other instructions. Two digits can be stored in an NCON instruction.

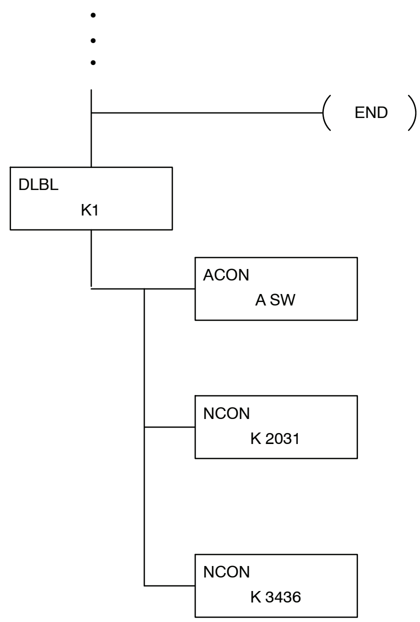
```
NCON
     K aaa
```

Operand Data Type		DL350 Range
		aaa
Constant	K	0–FFFF

Data Label Example

In the following example, an ACON and two NCON instructions are used within a DLBL instruction to build a text message. See the FAULT instruction for information on displaying messages.

***Direct*SOFT**



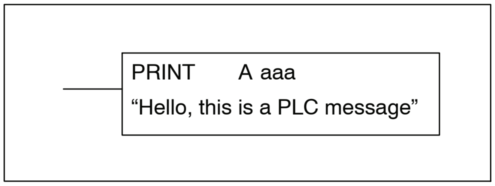
Handheld Programmer Keystrokes

•
•

SHFT	E 4	N TMR	D 3	ENT						
SHFT	D 3	L ANDST	B 1	L ANDST	→	B 1	ENT			
SHFT	A 0	C 2	O INST#	N TMR	→	S RST	W ANDN	ENT		
SHFT	N TMR	C 2	O INST#	N TMR	→	C 2	A 0	D 3	B 1	ENT
SHFT	N TMR	C 2	O INST#	N TMR	→	D 3	E 4	D 3	G 6	ENT

Print Message (PRINT)

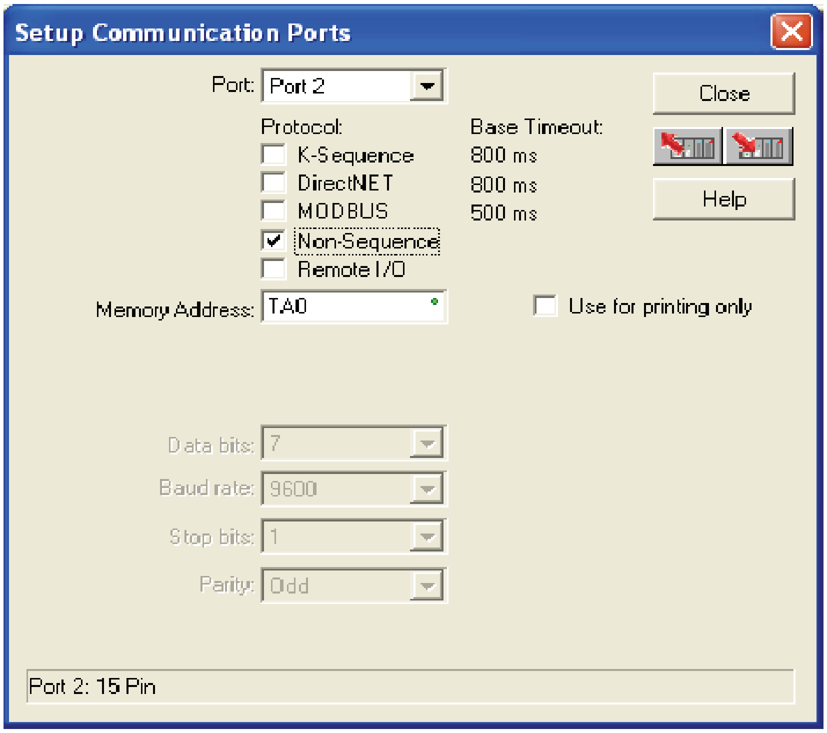
The Print Message instruction prints the embedded text or text/data variable message to Port 2 on the DL350 CPU, which must have the communications port configured.



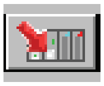
Data Type		DL350 Range
	A	aaa
Constant	K	1

You may recall from the CPU specifications in Chapter 3 that the DL350’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 the PLC menu, then Setup, then Setup Secondary Comm Port.

- **Port:** From the port number list box at the top, choose “Port 2”.
- **Protocol:** Click the check box to the left of “Non-sequence”, and then you’ll see the dialog box shown below.



- **Memory Address:** Choose a V-memory address for ***Direct***SOFT to use to store the port setup information. You will need to reserve 9 words in V-memory for this purpose. Select “Use for printing” if you are only using the port for print instructions output.
- **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.



Then click the button indicated to send the Port 2 configuration to the CPU, and click Close. Then see Chapter 3 for port wiring information, in order to connect your printer to the DL350.

Port 2 on the DL350 has standard RS232 levels, and should work with most printer serial input connections.

Text element – this is used for printing character strings. The character strings are defined as the character (more than 0) ranged by the double quotation marks. Two hex numbers preceded by the dollar sign means an 8-bit ASCII character code. Also, two characters preceded by the dollar sign is interpreted according to the following table:

#	Character code	Description
1	$$	Dollar sign ($)
2	$”	Double quotation (”)
3	$L or $l	Line feed (LF)
4	$N or $n	Carriage return line feed (CRLF)
5	$P or $p	Form feed
6	$R or $r	Carriage return (CR)
7	$T or $t	Tab

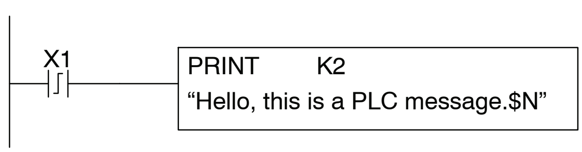
The following examples show various syntax conventions and the length of the output to the printer.

Example:

” ”	Length 0 without character
”A”	Length 1 with character A
” ”	Length 1 with blank
” $” ”	Length 1 with double quotation mark
” $ R $ L ”	Length 2 with one CR and one LF
” $ 0 D $ 0 A ”	Length 2 with one CR and one LF
” $ $ ”	Length 1 with one $ mark

In printing an ordinary line of text, you will need to include **double quotation** marks before and after the text string. Error code 499 will occur in the CPU when the print instruction contains invalid text or no quotations. It is important to test your PRINT instruction data during the application development.

The following example prints the message to port 2. We use a PD contact, which causes the message instruction to be active for just one scan. Note the $N at the end of the message, which produces a carriage return / line feed on the printer. This prepares the printer to print the next line, starting from the left margin.



Print the message to Port 2 when X1 makes an off-to-on transition.

V-memory element – this is used for printing V-memory contents in the integer format or real format. Use V-memory number or V-memory number with ":" and data type. The data types are shown in the table below. The Character code must be all capital letters.

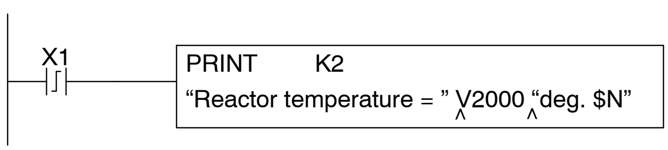
NOTE: There must be a space entered before and after the V-memory address to separate it from the text string. Failure to do this will result in an error code 499.

#	Character code	Description
1	none	16-bit binary (decimal number)
2	: B	4 digit BCD
3	: D	32-bit binary (decimal number)
4	: D B	8 digit BCD
5	: R	Floating point number (real number)
6	: E	Floating point number (real number with exponent)

Example:

V2000	Print binary data in V2000 for decimal number
V2000 : B	Print BCD data in V2000
V2000 : D	Print binary number in V2000 and V2001 for decimal number
V2000 : D B	Print BCD data in V2000 and V2001
V2000 : R	Print floating point number in V2000/V2001 as real number
V2000 : E	Print floating point number in V2000/V2001 as real number with exponent

Example: The following example prints a message containing text and a variable. The "reactor temperature" labels the data, which is at V2000. You can use the : B qualifier after the V2000 if the data is in BCD format, for example. The final string adds the units of degrees to the line of text, and the $N adds a carriage return / line feed.



Print the message to Port 2 when X1 makes an off-to-on transition.

^ represents a space

Message will read:
Reactor temperature = 0156 deg

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 SP116 and SP117 indicate the status of the DL350 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.