

Standard RLL Instructions

In This Chapter. . . .

- Introduction
 - Boolean Instructions
 - Comparative Boolean Instructions
 - Immediate Instructions
 - Timer, Counter, and Shift Register Instructions
 - Accumulator / Data Stack and Output Instructions
 - Accumulator Logic Instructions
 - Math Instructions
 - Bit Operation Instructions
 - Number Conversion Instructions
 - Table Instructions
 - Clock / Calender Instructions
 - CPU Control Instructions
 - Program Control Instructions
 - Interrupt Instructions
 - Intelligent I/O Instructions
 - Network Instructions
 - Message Instructions
-

Introduction

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

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

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

Example 1

		
430	440	450

Example 2

		
430	440	450

Instruction	Page
ACON	5-195
ACOSR	5-118
ADD	5-86
ADDB	5-98
ADDBD	5-99
ADDBS	5-112
ADDD	5-87
ADDF	5-104
ADDR	5-88
ADDS	5-108
AND	5-13, 5-69, 5-30,
ANDD	5-70
ANDND	5-72
ANDS	5-72
ANDSTR	5-15
ANDB	5-14
ANDD	5-70
ANDE	5-27
ANDF	5-71
ANDI	5-33
ANDMOV	5-170
ANDN	5-13, 5-30
ANDNB	5-14
ANDNE	5-27
ANDNI	5-33
ANDPD	5-22
ANDS	5-72

Instruction	Page
ANDSTR	5-117
ASINR	5-117
ATANR	5-118
ATH	5-134
ATT	5-158
BCALL	7-27
BCD	5-128
BCDCPL	5-130
BEND	7-27
BIN	5-127
BLK	7-27
BREAK	5-177
BTOR	5-131
CMP	5-81
CMPD	5-82
CMPF	5-83
CMPR	5-85
CMPS	5-84
CNT	5-46
COSR	5-117
CV	7-25
CVJMP	7-25
DATE	5-174
DEC	5-116
DECB	5-119
DECO	5-126

Instruction	Page
DEGR	5-133
DISI	5-185
DIV	5-95
DIVB	5-103
DIVBS	5-115
DIVD	5-96
DIVF	5-107
DIVR	5-97
DIVS	5-111
DLBL	5-195
DRUM	6-16
EDRUM	6-19
ENCO	5-125
END	5-176
ENI	5-185
FAULT	5-194
FDGT	5-143
FILL	5-141
FIND	5-142
FINDB	5-172
FOR	5-179
GOTO	5-178
GRAY	5-138
GTS	5-180
HISTORY	5-196
HTA	5-135
INC	5-116

Instruction	Page
INCB	5-119
INT	5-184
INV	5-129
IRT	5-185
IRTC	5-185
ISG	7-24
JMP	7-24
LBL	5-178
LD	5-58
LDA	5-60
LDD	5-58
LDF	5-59
LDI	5-36
LDIF	5-37
LDLBL	5-161
LDR	5-63
LDSX	5-62
LDX	5-61
MDRMD	6-22
MDRMW	6-25
MLR	5-182
MLS	5-182
MOV	5-145
MOVMC	5-161
MUL	5-92
MULB	5-102
MULBS	5-114
MULD	5-93
MULF	5-106
MULR	5-94
MULS	5-110
NCON	5-196
NEXT	5-179
NJMP	7-24
NOP	5-176
NOT	5-18
OR	5-11, 5-73 5-29,
ORB	5-12
ORD	5-74
ORE	5-26
ORF	5-75
ORI	5-32
ORMOV	5-170
ORN	5-11, 5-29
ORNB	5-12

Instruction	Page
ORND	5-21
ORNE	5-26
ORNI	5-32
OROUT	5-18
OROUTI	5-34
ORPD	5-21
ORS	5-76
ORSTR	5-15
OUT	5-16, 5-64
OUTB	5-17
OUTD	5-64
OUTF	5-65
OUTI	5-34
OUTIF	5-39
OUTL	5-67
OUTM	5-67
OUTX	5-66
PAUSE	5-19
PD	5-19
POP	5-68
PRINT	5-200
RADR	5-133
RD	5-188
RDF	NO TAG
RFB	5-149
RFT	5-155
ROTL	5-123
ROTR	5-124
RST	5-23
RSTB	5-24
RSTBIT	5-166
RSTI	5-35
RSTWT	5-177
RT	5-180
RTC	5-180
RTOB	5-132
RX	5-190
SBR	NO TAG
SEG	5-137
SET	5-23
SETB	5-24
SETBIT	5-166
SETI	5-35
SFLDGT	5-139
SG	7-23

Instruction	Page
SGCNT	5-48
SHFL	5-121
SHFR	5-122
SINR	5-117
SQRTR	5-118
SR	5-52
STOP	5-176
STR	5-9, 5-28
STRB	5-10
STRE	5-25
STRI	5-31
STRN	5-9, 5-28
STRNB	5-10
STRND	5-20
STRNE	5-25
STRNI	5-31
STRPD	5-20
STT	5-152
SUB	5-89
SUBB	5-100
SUBBD	5-101
SUBBS	5-113
SUBD	5-90
SUBF	5-105
SUBR	5-91
SUBS	5-109
SUM	5-120
SWAP	5-173
TANR	5-117
TIME	5-175
TMR	5-41
TMRA	5-43
TMRAF	5-43
TMRF	5-41
TSHFL	5-168
TSHFR	5-168
TTD	5-146
UDC	5-50
WT	5-189
WX	5-191
XOR	5-77
XORD	5-78
XORF	5-79
XORMOV	5-170
XORS	5-80

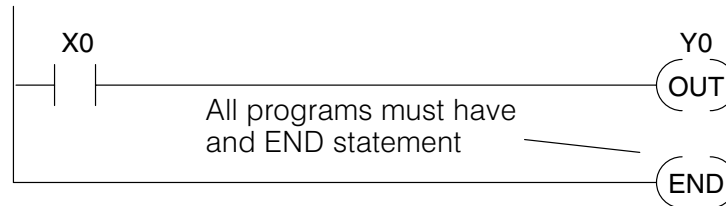
Using Boolean Instructions

Do you ever wonder why so many PLC manufacturers always quote the scan time for a 1K boolean program? ... simple: 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 **DirectSOFT32** package allows you to use graphic symbols to build the program, you don't absolutely *have* to know the boolean equivalents of the instructions. However, it may be helpful at some point, especially if you ever have to troubleshoot the program with a Handheld Programmer.

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

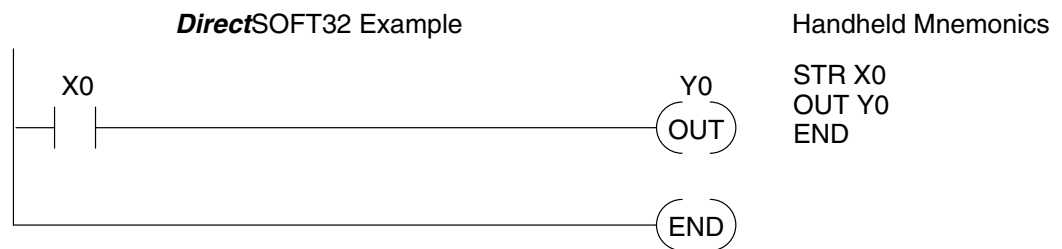
END Statement

DL405 programs require an END statement (coil) as the last instruction. This tells the CPU this is the end of the program. Normally, any instructions placed after the END statement will not be executed. There are exceptions to this such as interrupt routines, etc. Chapter 5 discusses the instruction set in detail.



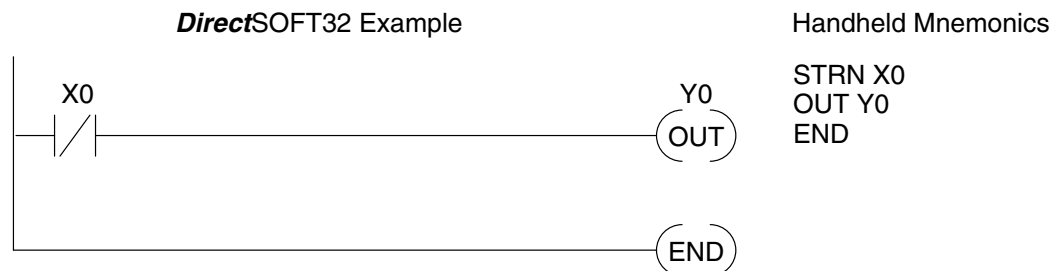
Simple Rungs

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

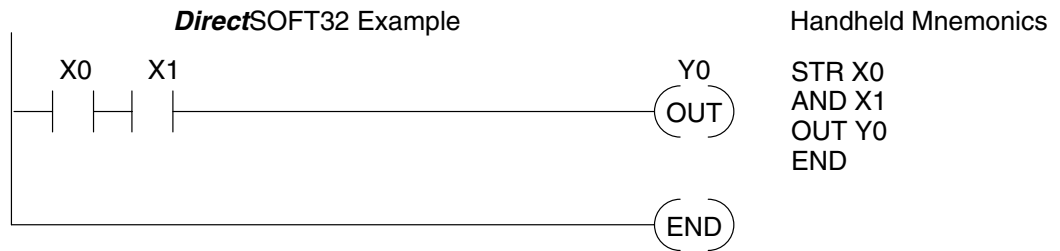


Normally Closed Contact

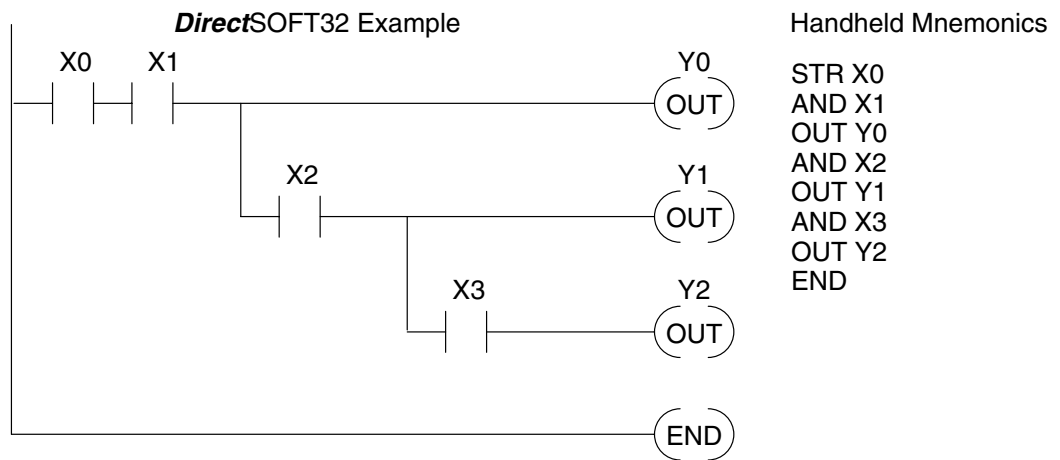
Normally closed contacts are also very common. This is accomplished with the Store Not or, STRN instruction. The following example shows a simple rung with a normally closed contact.



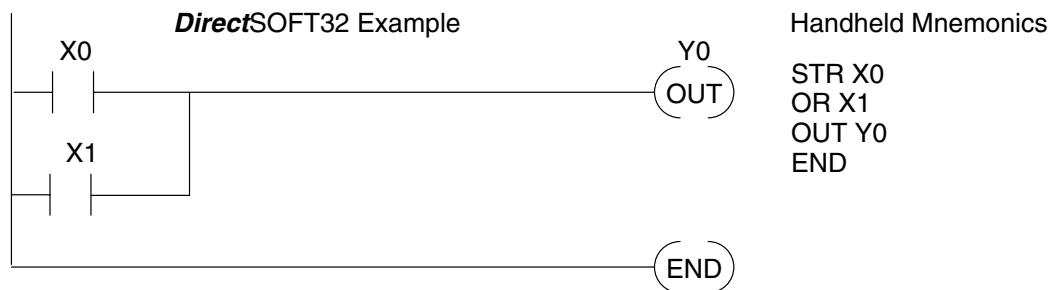
Contacts in Series Use the AND instruction to join two or more contacts in series. The following example shows two contacts in series and a single output coil. The instructions used would be STR X0, AND X1, followed by OUT Y0.



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

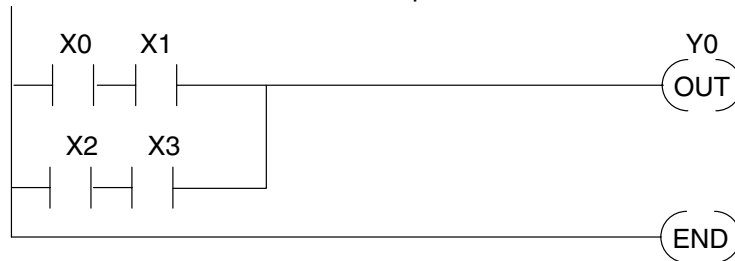


Parallel Elements You also have to join contacts in parallel. The OR instruction allows you to do this. The following example shows two contacts in parallel and a single output coil. The instructions would be STR X0, OR X1, followed by OUT Y0.



**Joining Series
Branches in
Parallel**

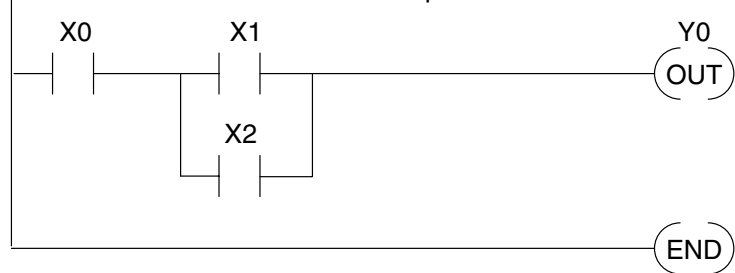
Quite often it is necessary to join several groups of series elements in parallel. The Or Store (ORSTR) instruction allows this operation. The following example shows a simple network consisting of series elements joined in parallel.

DirectSOFT32 Example**Handheld Mnemonics**

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

**Joining Parallel
Branches in Series**

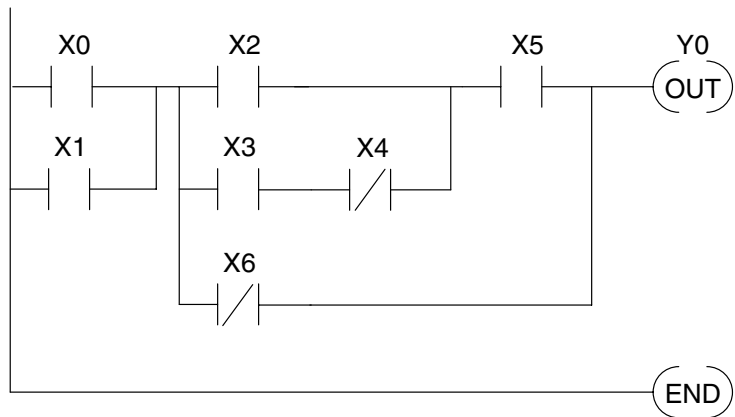
Quite often it is also necessary to join one or more parallel branches in series. The And Store (ANDSTR) instruction allows this operation. The following example shows a simple network with contact branches in series with parallel contacts.

DirectSOFT32 Example**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.

**Handheld Mnemonics**

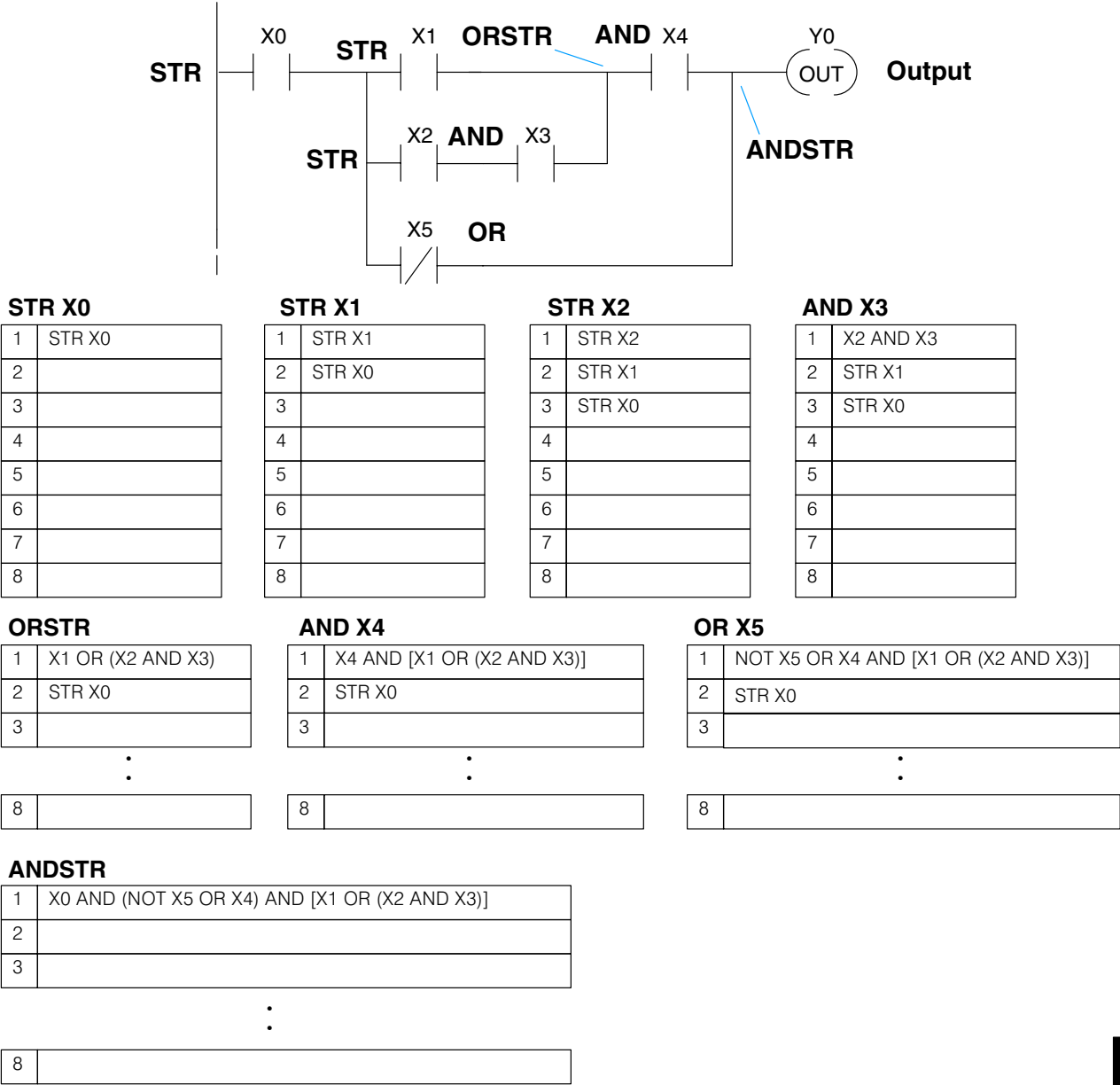
```
STR X0
OR X1
STR X2
STR X3
ANDN X4
ORSTR
AND X5
ORN X6
ANDSTR
OUT Y0
```

Boolean Stack

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

All of you software programmers may be saying, “I use **DirectSOFT32**, so I don’t need to know how the stack works.” Not quite true. Even though you can build the network with the graphic symbols, the limits of the CPU are still the same. If the stack limit is exceed when the program is compiled and error will occur.

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



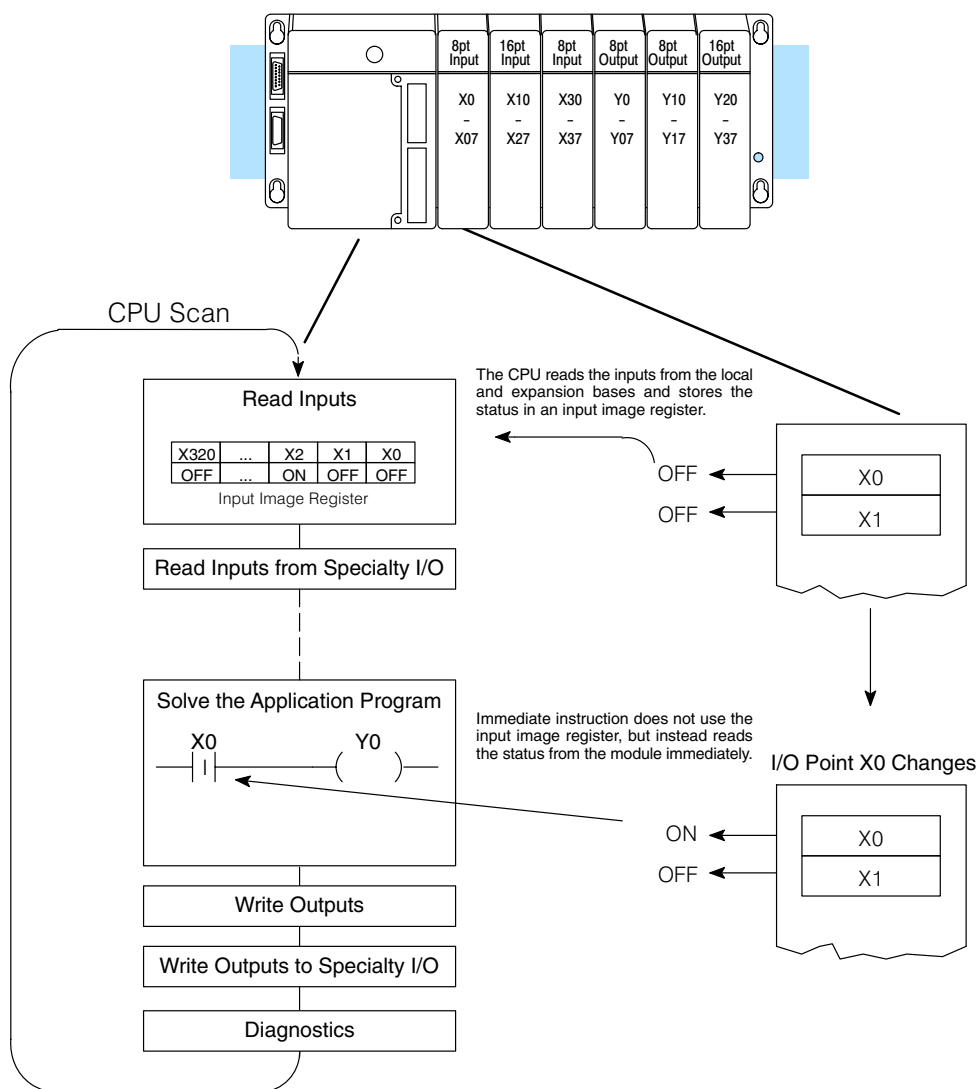
Comparative Boolean

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

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

Immediate Boolean The DL405 CPUs usually can complete an operation cycle in a matter of milliseconds. However, in some applications you may not be able to wait a few milliseconds until the next I/O update occurs. The DL405 CPUs offer Immediate input and outputs which are special boolean instructions that allow reading directly from inputs and writing directly to outputs during the program execution portion of the CPU cycle. You may recall this is normally done during the input or output update portion of the CPU cycle. The immediate instructions take longer to execute because the program execution is interrupted while the CPU reads or writes the module. This function is not normally done until the read inputs or the write outputs portion of the CPU cycle.

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

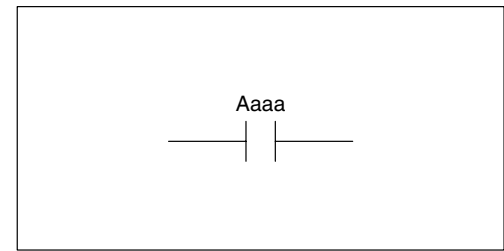


Boolean Instructions

Store (STR)



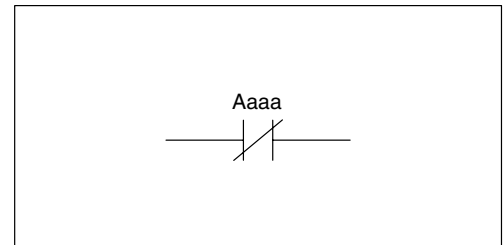
The Store instruction begins a new rung or an additional branch in a rung with a normally open contact. Status of the contact will be the same state as the associated image register point or memory location.



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.



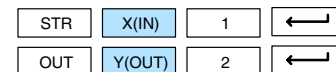
Operand Data Type		DL430 Range	DL440 Range	DL450 Range
A		aaa	aaa	aaa
Inputs	X	0-477	0-477	0-1777
Outputs	Y	0-477	0-477	0-1777
Control Relays	C	0-737	0-1777	0-3777
Stage	S	0-577	0-1777	0-1777
Timer	T	0-177	0-377	0-377
Counter	CT	0-177	0-177	0-377
Special Relay	SP	0-137, 320-617	0-137 320-717	0-137, 320-717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

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

DirectSOFT32



Handheld Programmer Keystrokes

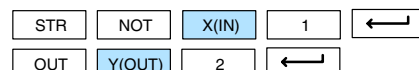


In the following Store Not example, when input X1 is off output Y2 will energize.

DirectSOFT32



Handheld Programmer Keystrokes



Store Bit-of-Word (STRB)

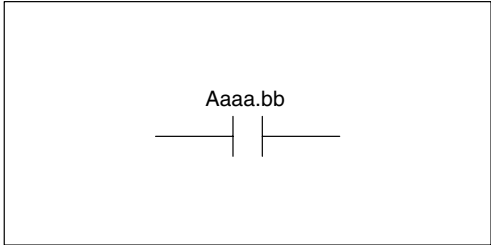
✕

✕

✓

430440450

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)

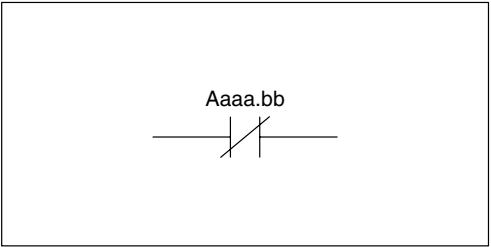
✕

✕

✓

430440450

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		DL450 Range	
A		aaa	bb
Vmemory	B	All (See p. 3-42)	BCD, 0 to 15
Pointer	PB	All (See p. 3-42)	BCD, 0 to 15

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

DirectSOFT32



Handheld Programmer Keystrokes

STR

SHFT

B

SHFT

V

1

4

0

0

K(con)

1

2

←

OUT

Y(OUT)

2

←

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

DirectSOFT32



Handheld Programmer Keystrokes

STR

STR

SHFT

B

SHFT

V

1

4

0

0

K(con)

1

2

←

OUT

Y(OUT)

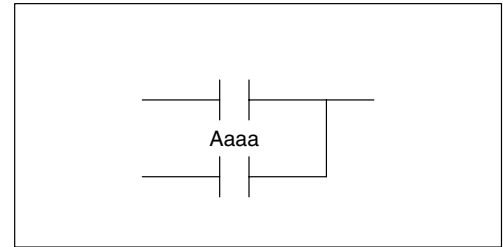
2

←

Or (OR)



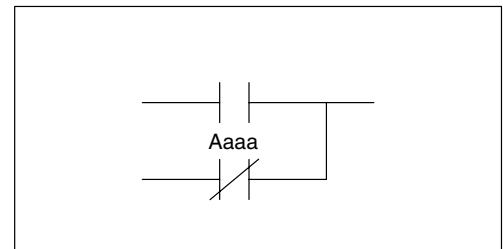
The Or instruction logically ors a normally open contact in parallel with another contact in a rung. The status of the contact will be the same state as the associated image register point or memory location.



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

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

DirectSOFT32

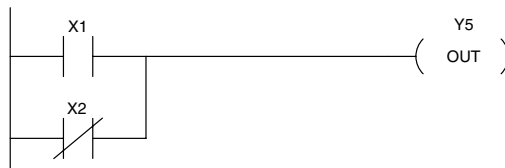


Handheld Programmer Keystrokes

STR	X(IN)	1	←
OR	X(IN)	2	←
OUT	Y(OUT)	5	←

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

DirectSOFT32



Handheld Programmer Keystrokes

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

Or Bit-of-Word (ORB)

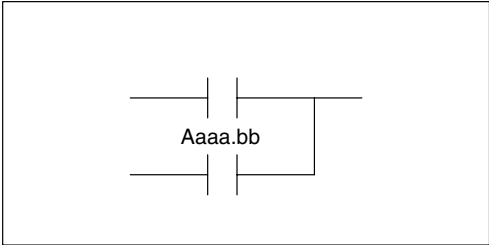
✕

✕

✓

430440450

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)

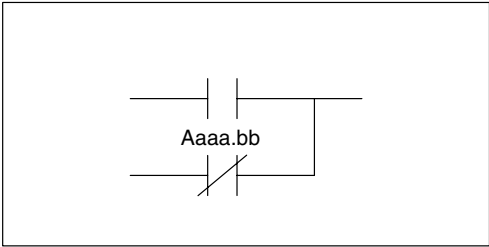
✕

✕

✓

430440450

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		DL450 Range	
	A	aaa	bb
Vmemory	B	All (See p. 3-42)	BCD, 0 to 15
Pointer	PB	All (See p. 3-42)	BCD

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

DirectSOFT32

Handheld Programmer Keystrokes

STR

X(IN)

1

←

OR

SHFT

B

SHFT

V

1

4

0

0

K(con)

7

←

OUT

Y(OUT)

5

←

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

DirectSOFT32

Handheld Programmer Keystrokes

STR

X(IN)

1

←

OR

SHFT

N

B

SHFT

V

1

4

0

0

K(con)

7

←

OUT

Y(OUT)

5

←

And (AND)



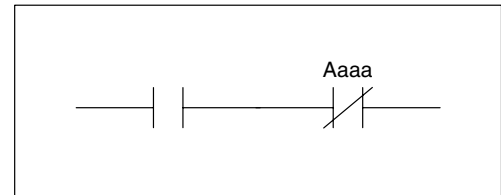
The And instruction logically ands a normally open contact in series with another contact in a rung. The status of the contact will be the same state as the associated image register point or memory location.



And Not (ANDN)



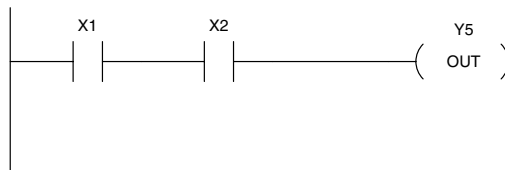
The And Not Bit-of-Word instruction logically ands a normally closed contact in series with another contact in a rung. The status of the contact will be opposite the state of the associated image register point or memory location.



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

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

DirectSOFT32

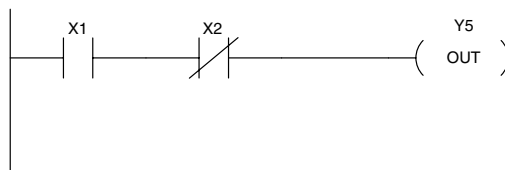


Handheld Programmer Keystrokes

STR	X(IN)	1	←
AND	X(IN)	2	←
OUT	Y(OUT)	5	←

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

DirectSOFT32



Handheld Programmer Keystrokes

STR	X(IN)	1	←
AND	NOT	X(IN)	2 ←
OUT	Y(OUT)	5	←

And Bit-of-Word
(ANDB)

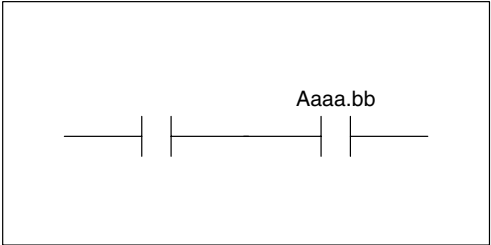
✕

✕

✓

430 440 450

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)

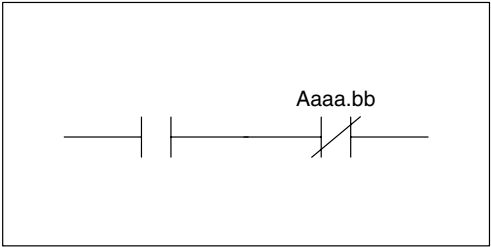
✕

✕

✓

430 440 450

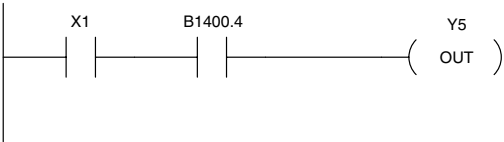
The And Not Bit-of-Word instruction logically ands a normally closed contact in series with another contact in a rung. The status of the contact will be opposite the state of the bit referenced in the associated memory location.



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

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

DirectSOFT32



Handheld Programmer Keystrokes

STRX(IN)1←

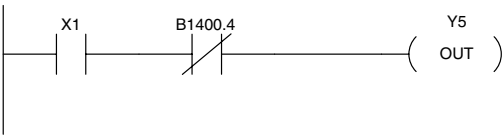
ANDSHFTBSHFTV140000

K(con)4←

OUTY(OUT)5←

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

DirectSOFT32



Handheld Programmer Keystrokes

STRX(IN)1←

ANDSHFTNBSHFTV140000

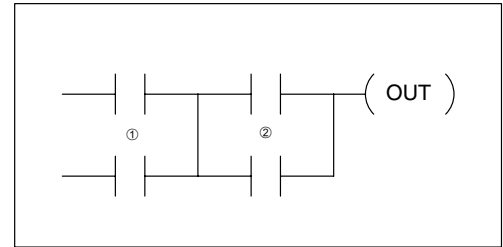
K(con)4←

OUTY(OUT)5←

And Store (ANDSTR)

✓ ✓ ✓
430 440 450

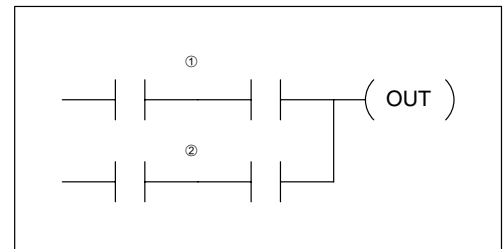
The And Store instruction logically ands two branches of a rung in series. Both branches must begin with the Store instruction.



Or Store (ORSTR)

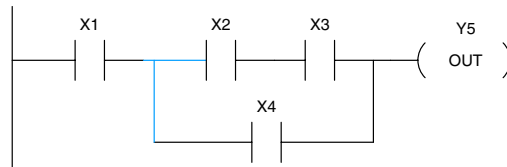
✓ ✓ ✓
430 440 450

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.

DirectSOFT32

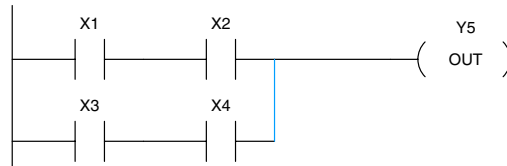


Handheld Programmer Keystrokes

STR	X(IN)	1	←
STR	X(IN)	2	←
AND	X(IN)	3	←
OR	X(IN)	4	←
AND	STR	←	
OUT	Y(OUT)	5	←

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

DirectSOFT32



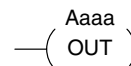
Handheld Programmer Keystrokes

STR	X(IN)	1	←
AND	X(IN)	2	←
STR	X(IN)	3	←
AND	X(IN)	4	←
OR	STR	←	
OUT	Y(OUT)	5	←

**Out
(OUT)**


430 440 450

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

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

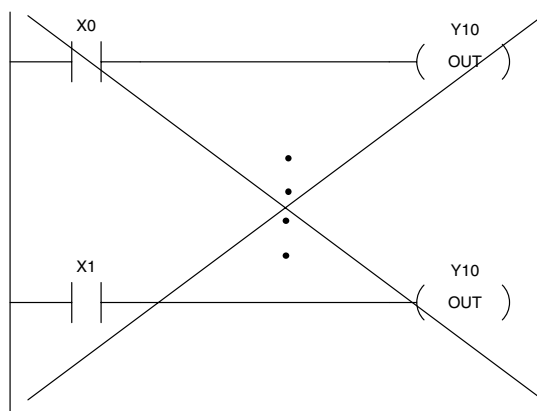
DirectSOFT32



Handheld Programmer Keystrokes

STR	X(IN)	1	←
OUT	Y(OUT)	2	←
OUT	Y(OUT)	5	←

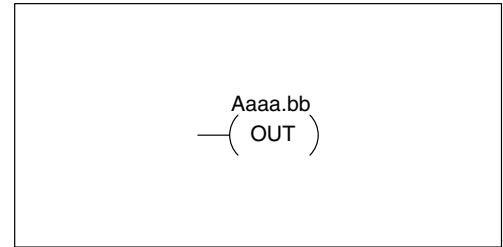
In the following Out example the program contains two Out instructions using the same location (Y10). The physical output of Y10 is ultimately controlled by the last rung of logic referencing Y10. X1 will override the Y10 output being controlled by X0. To avoid this situation, multiple outputs using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



Out Bit-of-Word (OUTB)

✕ ✕ ✓
430 440 450

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.



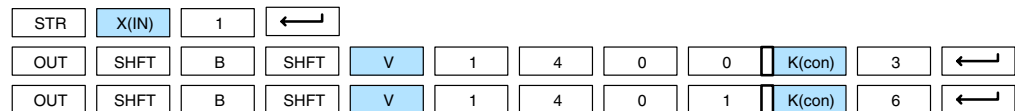
Operand Data Type		DL450 Range	
A		aaa	bb
Vmemory	B	All (See p. 3-42)	BCD, 0 to 15
Pointer	PB	All (See p. 3-42)	BCD

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

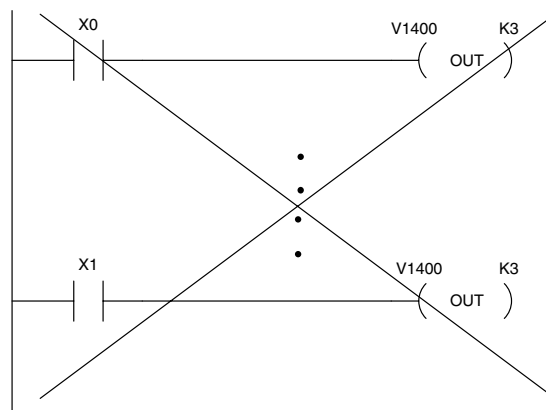
DirectSOFT32



Handheld Programmer Keystrokes



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
(OROUT)**

✓ ✓ ✓
430 440 450

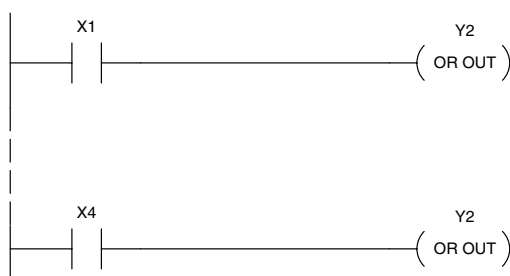
The Or Out instruction has been designed to use more than 1 rung of discrete logic to control a single output. Multiple Or Out instructions referencing the same output coil may be used, since *all* contacts controlling the output are ored together. If the status of *any* rung is on, the output will also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

A aaa
(OROUT)

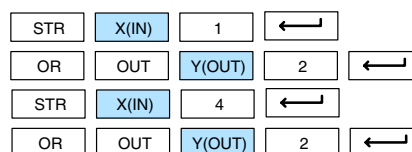
Operand Data Type		DL430 Range	DL440 Range	DL450 Range
A		aaa	aaa	aaa
Inputs	X	0-477	0-477	0-1777
Outputs	Y	0-477	0-477	0-1777
Control Relays	C	0-737	0-1777	0-3777
Global I/O	GX	0-777	0-1777	0-2777 (GX + GY)

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

DirectSOFT32

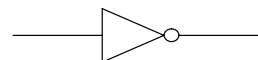


Handheld Programmer Keystrokes

**Not
(NOT)**

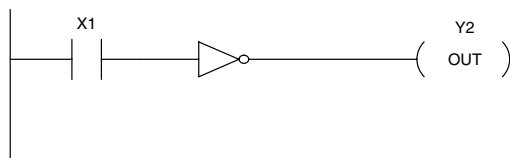
✓ ✓ ✓
430 440 450

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

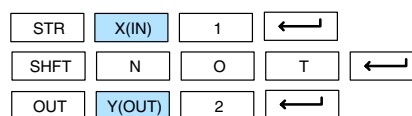


In the following example when X1 is off, Y2 will energize. This is because the Not instruction inverts the status of the rung at the Not instruction.

DirectSOFT32



Handheld Programmer Keystrokes



NOTE: *DirectSOFT32* 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 *DirectSOFT32* versions earlier than 1.1i.

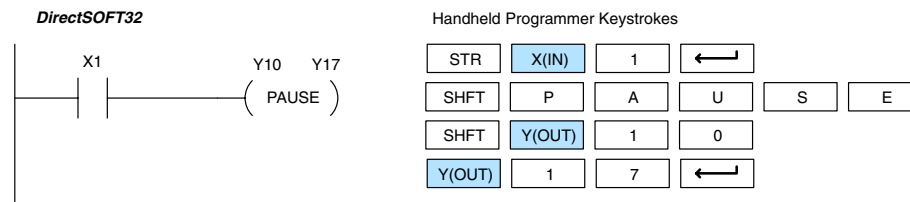
**Pause
(PAUSE)**

✓	✓	✓
430	440	450

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

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

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

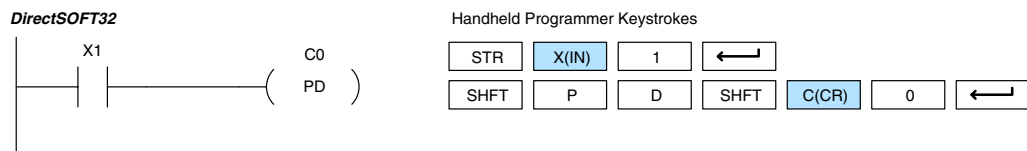
**Positive
Differential
(PD)**

✓	✓	✓
430	440	450

The Positive Differential instruction is typically known as a one-shot. When the input logic produces an Off-to-On transition, the output will energize for one CPU scan. Thereafter, it remains off until its input makes another Off-to-On transition.

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

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

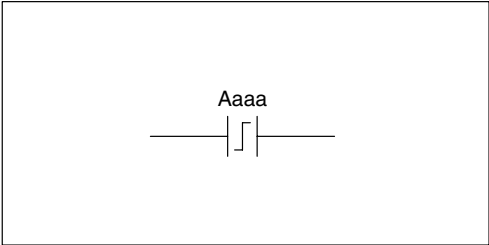


Note that you can place a “Not” instruction immediately before the PD instruction to generate a “one-shot” pulse on an on-to-off transition.

Store Positive
Differential
(STRPD)

X X ✓
430 440 450

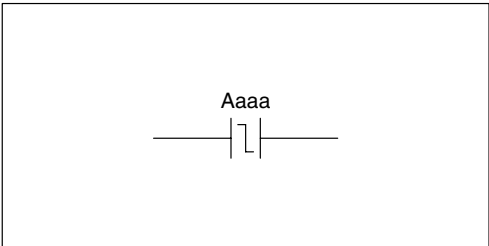
The Store Positive Differential instruction begins a new rung or an additional branch in a rung with a normally open contact. The contact closes for one CPU scan when the state of the associated image register point makes an Off-to-On transition. Thereafter, the contact remains open until the next Off-to-On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



Store Negative
Differential
(STRND)

X X ✓
430 440 450

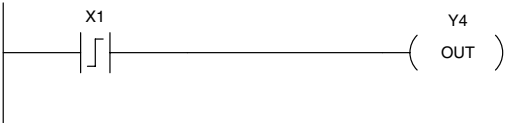
The Store Negative Differential instruction begins a new rung or an additional branch in a rung with a normally closed contact. The contact closes for one CPU scan when the state of the associated image register point makes an On-to-Off transition. Thereafter, the contact remains open until the next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



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

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

DirectSOFT32



Handheld Programmer Keystrokes

STR	SHFT	P	D	SHFT
X(IN)	1	←		
OUT	Y(OUT)	4	←	

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

DirectSOFT32



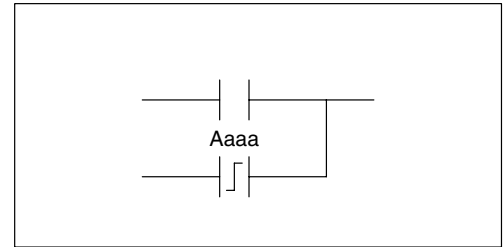
Handheld Programmer Keystrokes

STR	SHFT	N	D	SHFT
X(IN)	1	←		
OUT	Y(OUT)	4	←	

Or Positive Differential (ORPD)

✗ ✗ ✓
430 440 450

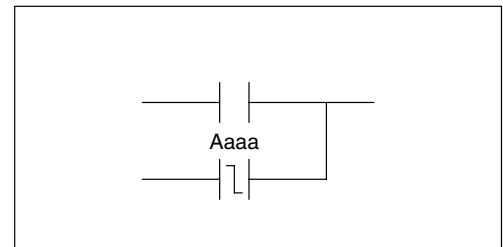
The Or Positive Differential instruction logically ors a normally open contact in parallel with another contact in a rung. The status of the contact will be open until the associated image register point makes an Off-to-On transition, closing it for one CPU scan. Thereafter, it remains open until another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



Or Negative Differential (ORND)

✗ ✗ ✓
430 440 450

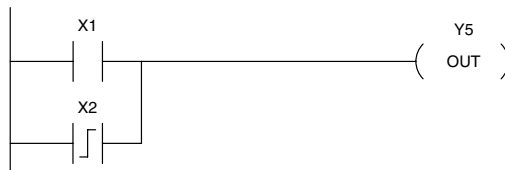
The Or Negative Differential instruction logically ors a normally open contact in parallel with another contact in a rung. The status of the contact will be open until the associated image register point makes an On-to-Off transition, closing it for one CPU scan. Thereafter, it remains open until another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



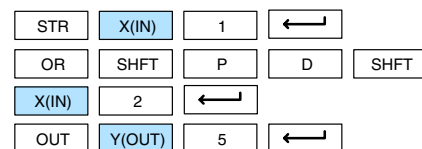
Operand Data Type		DL450 Range
	A	aaa
Inputs	X	0-1777
Outputs	Y	0-1777
Control Relays	C	0-3777
Stage	S	0-1777
Timer	T	0-377
Counter	CT	0-377
Global	GX	0-2777 (GX + GY)

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

DirectSOFT32

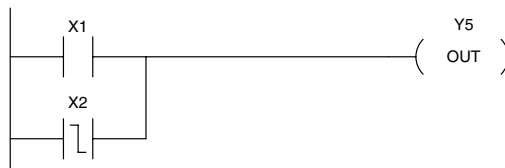


Handheld Programmer Keystrokes

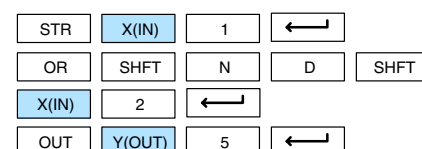


In the following example, Y 5 will energize whenever X1 is on, or for one CPU scan when X2 transitions from On to Off.

DirectSOFT32



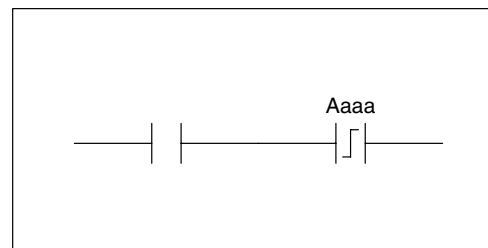
Handheld Programmer Keystrokes



And Positive Differential (ANDPD)

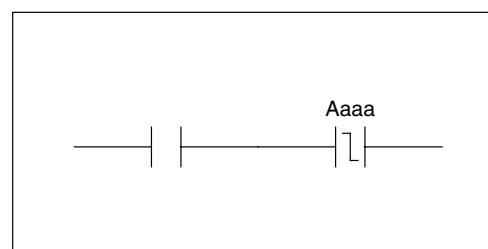
✗	✗	✓
430	440	450

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

**And Negative Differential (ANDND)**

✗	✗	✓
430	440	450

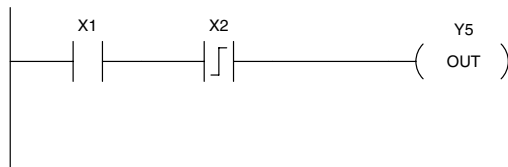
The And Negative Differential instruction logically ands a normally open contact in series with another contact in a rung. The status of the contact will be open until the associated image register point makes an On-to-Off transition, closing it for one CPU scan. Thereafter, it remains open until another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



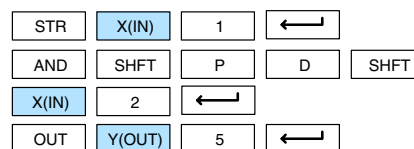
Operand Data Type		DL450 Range
	A	aaa
Inputs	X	0-1777
Outputs	Y	0-1777
Control Relays	C	0-3777
Stage	S	0-1777
Timer	T	0-377
Counter	CT	0-377
Global	GX	0-2777 (GX + GY)

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

DirectSOFT32

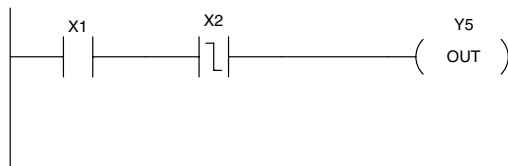


Handheld Programmer Keystrokes

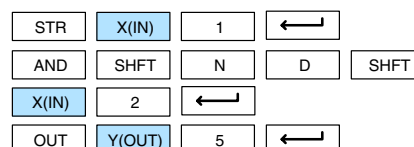


In the following example, Y5 will energize for one CPU scan whenever X1 is on and X2 transitions from On to Off.

DirectSOFT32



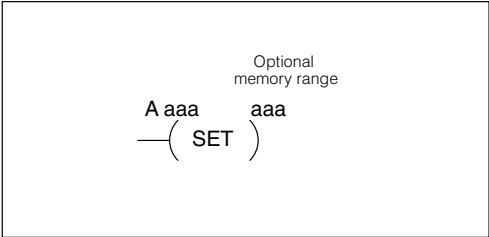
Handheld Programmer Keystrokes



Set
(SET)



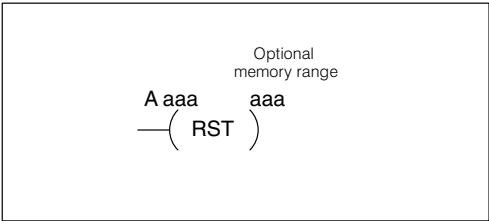
The Set instruction sets or turns on an image register point/memory location or a consecutive range of image register points/memory locations. Once the point/location is set it will remain on until it is reset using the Reset instruction. It is not necessary for the input controlling the Set instruction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

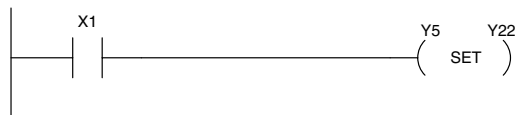


Operand Data Type		DL430 Range	DL440 Range	DL450 Range
A	aaa	aaa	aaa	aaa
Inputs	X	0-477	0-477	0-1777
Outputs	Y	0-477	0-477	0-1777
Control Relays	C	0-737	0-1777	0-3777
Stages	S	0-577	0-1777	0-1777
Timers*	T	0-177	0-377	0-377
Counters*	CT	0-177	0-177	0-377
Global	GX	0-777	0-1777	0-2777 (GX + GY)

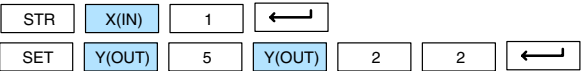
* Timer and counter operand data types are not valid using the Set instruction.

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

DirectSOFT32



Handheld Programmer Keystrokes

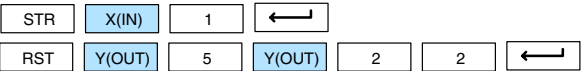


In the following example when X1 turns on, Y5 through Y22 will be reset to the off state.

DirectSOFT32



Handheld Programmer Keystrokes



Set Bit-of-Word
(SETB)

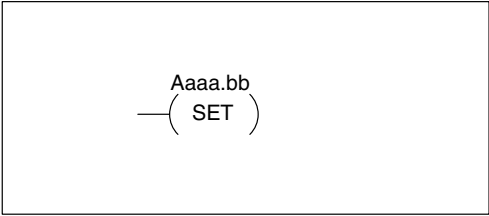
✕

✕

✓

430 440 450

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)

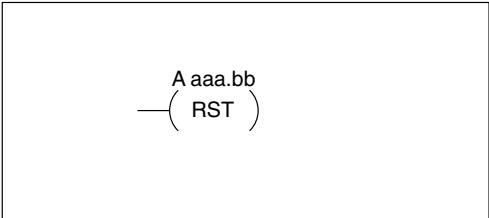
✕

✕

✓

430 440 450

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		DL450 Range	
	A	aaa	bb
Vmemory	B	All (See p. 3–42)	0 to 15
Pointer	PB	All (See p. 3–42)	0 to 15

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

DirectSOFT32



Handheld Programmer Keystrokes

STRX(IN)1←

SETSHFTBSHFTV14000

K(con)0←

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

DirectSOFT32



Handheld Programmer Keystrokes

STRX(IN)1←

RSTSHFTBSHFTV14000

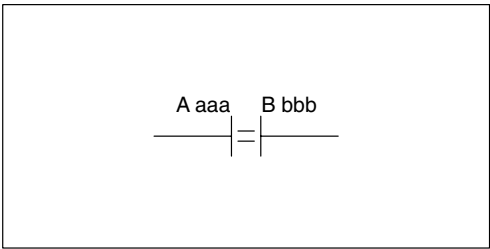
K(con)15←

Comparative Boolean Instructions

Store If Equal (STRE)

✓ ✓ ✓
430 440 450

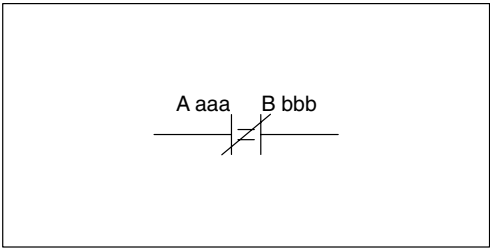
The Store If Equal instruction begins a new rung or additional branch in a rung with a normally open comparative contact. The contact will be on when $Aaaa = Bbbb$.



Store If Not Equal (STRNE)

✓ ✓ ✓
430 440 450

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



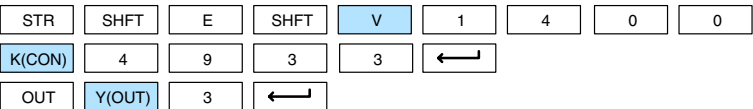
Operand Data Type		DL430 Range		DL440 Range		DL450 Range	
A/B		aaa	bbb	aaa	bbb	aaa	bbb
Vmemory	V	All (See p. 3-40)	All (See p. 3-40)	All (See p. 3-41)	All (See p. 3-41)	All (See p. 3-42)	All (See p. 3-42)
Pointer	P	—	—	All (See p. 3-41)	All (See p. 3-41)	All (See p. 3-42)	All (See p. 3-42)
Constant	K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

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

DirectSOFT32



Handheld Programmer Keystrokes

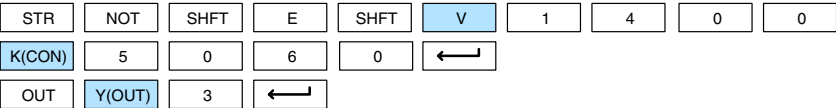


In the following example, when the value in Vmemory location V1400 \neq 5060, Y3 will energize.

DirectSOFT32



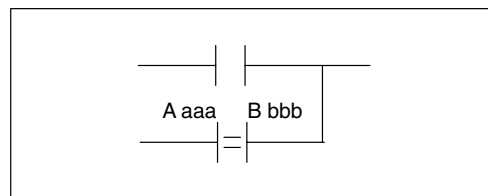
Handheld Programmer Keystrokes



**Or If Equal
(ORE)**

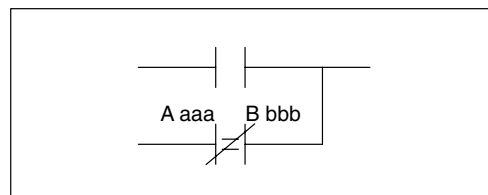
✓ ✓ ✓
430 440 450

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

**Or If Not Equal
(ORNE)**

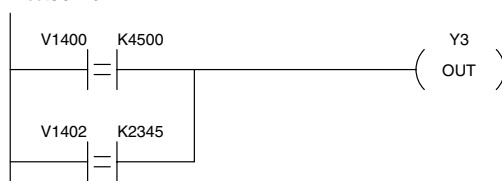
✓ ✓ ✓
430 440 450

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



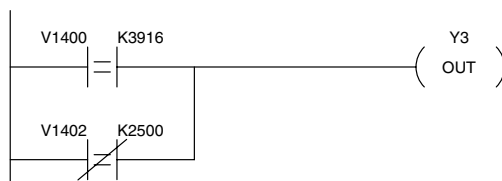
Operand Data Type	A/B	DL430 Range		DL440 Range		DL450 Range	
		aaa	bbb	aaa	bbb	aaa	bbb
Vmemory	V	All (See p. 3-40)	All (See p. 3-40)	All (See p. 3-41)	All (See p. 3-41)	All (See p. 3-42)	All (See p. 3-42)
Pointer	P	—	—	All (See p. 3-41)	All (See p. 3-41)	All (See p. 3-42)	All (See p. 3-42)
Constant	K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

In the following example, when the value in Vmemory location V1400 = 4500 or V1402 = 2345, Y3 will energize.

DirectSOFT32**Handheld Programmer Keystrokes**

STR	SHFT	E	SHFT	V	1	4	0	0
K(CON)	4	5	0	0	←			
OR	SHFT	E	SHFT	V	1	4	0	2
K(CON)	2	3	4	5	←			
OUT	Y(OUT)	3	←					

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

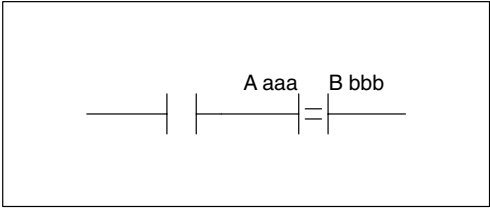
DirectSOFT32**Handheld Programmer Keystrokes**

STR	SHFT	E	SHFT	V	1	4	0	0
K(CON)	3	9	1	6	←			
OR	NOT	SHFT	E	SHFT	V	1	4	0
K(CON)	2	5	0	0	←			
OUT	Y(OUT)	3	←					

And If Equal (ANDE)

✓ ✓ ✓
430 440 450

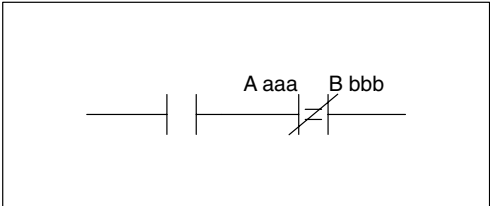
The And If Equal instruction connects a normally open comparative contact in series with another contact. The contact will be on when Aaaa = Bbbb.



And If Not Equal (ANDNE)

✓ ✓ ✓
430 440 450

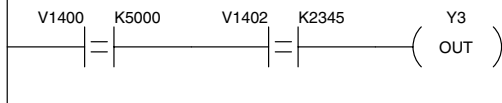
The And If Not Equal instruction connects a normally closed comparative contact in series with another contact. The contact will be on when Aaaa ≠ Bbbb.



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

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

DirectSOFT32

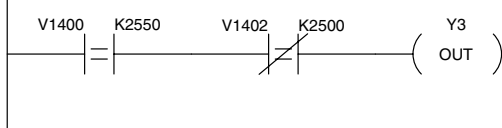


Handheld Programmer Keystrokes

STR	SHFT	E	SHFT	V	1	4	0	0
K(CON)	5	0	0	0	←			
AND	SHFT	E	SHFT	V	1	4	0	2
K(CON)	2	3	4	5	←			
OUT	Y(OUT)	3	←					

In the following example, when the value in Vmemory location V1400 = 2550 and V1402 ≠ 2500, Y3 will energize.

DirectSOFT32



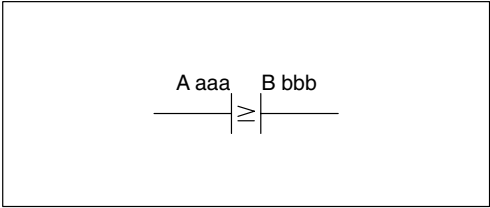
Handheld Programmer Keystrokes

STR	SHFT	E	SHFT	V	1	4	0	0
K(CON)	2	5	5	0	←			
AND	NOT	SHFT	E	SHFT	V	1	4	0
K(CON)	2	5	0	0	←			
OUT	Y(OUT)	3	←					

Store
(STR)

✓ ✓ ✓
430 440 450

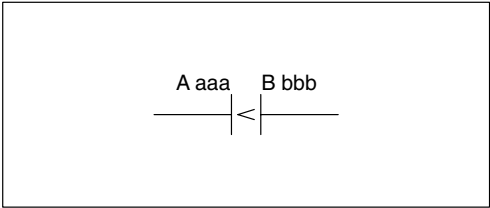
The Comparative Store instruction begins a new rung or additional branch in a rung with a normally open comparative contact. The contact will be on when $Aaaa \geq Bbbb$.



Store Not
(STRN)

✓ ✓ ✓
430 440 450

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

In the following example, when the value in Vmemory location V1400 \geq 1000, Y3 will energize.

DirectSOFT32

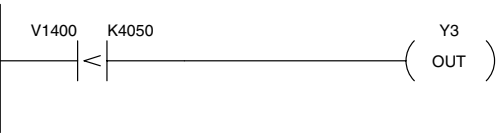


Handheld Programmer Keystrokes

STR V 1 4 0 0
K(CON) 1 0 0 0
OUT Y(OUT) 3

In the following example, when the value in Vmemory location V1400 $<$ 4050, Y3 will energize.

DirectSOFT32



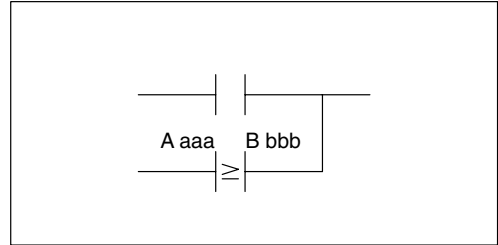
Handheld Programmer Keystrokes

STR NOT V 1 4 0 5
K(CON) 4 0 5 0
OUT Y(OUT) 3

Or (OR)



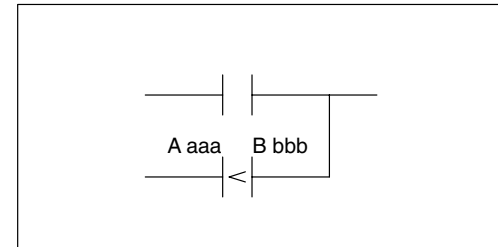
The Comparative Or instruction connects a normally open comparative contact in parallel with another contact. The contact will be on when $Aaaa \geq Bbbb$.



Or Not (ORN)



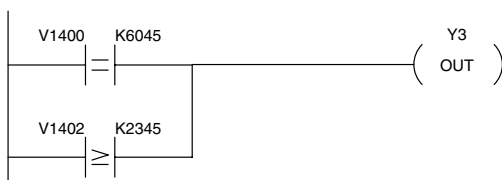
The Comparative Or Not instruction connects a normally open comparative contact in parallel with another contact. The contact will be on when $Aaaa < Bbbb$.



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

In the following example, when the value in Vmemory location V1400 = 6045 or V1402 \geq 2345, Y3 will energize.

DirectSOFT32

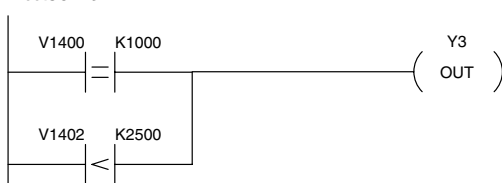


Handheld Programmer Keystrokes

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

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

DirectSOFT32



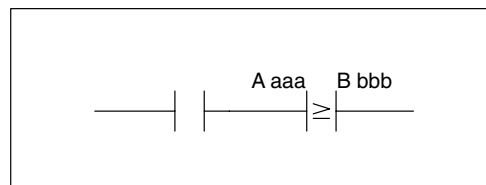
Handheld Programmer Keystrokes

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

And (AND)



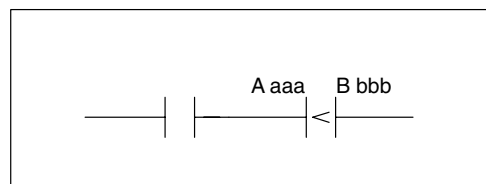
The Comparative And instruction connects a normally open comparative contact in series with another contact. The contact will be on when $Aaaa \geq Bbbb$.



And Not (ANDN)



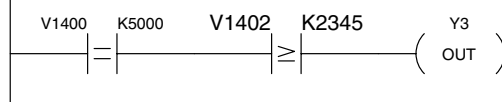
The Comparative And Not instruction connects a normally open comparative contact in series with another contact. The contact will be on when $Aaaa < Bbbb$.



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

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

DirectSOFT32

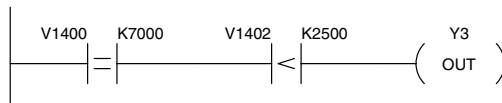


Handheld Programmer Keystrokes

STR	SHFT	E	SHFT	V	1	4	0	0
K(CON)	5	0	0	0	←			
AND	V	1	4	0	2			
K(CON)	2	3	4	5	←			
OUT	Y(OUT)	3	←					

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

DirectSOFT32



Handheld Programmer Keystrokes

STR	SHFT	E	SHFT	V	1	4	0	0
K(CON)	7	0	0	0	←			
AND	NOT	V	1	4	0	2		
K(CON)	2	5	0	0	←			
OUT	Y(OUT)	3	←					

Immediate Instructions

Store Immediate (STRI)

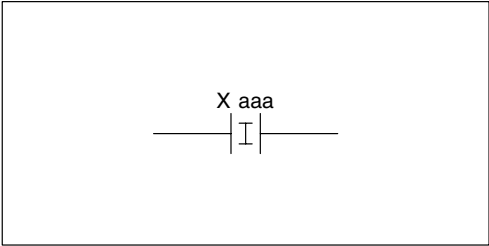
✓

✓

✓

430 440 450

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)

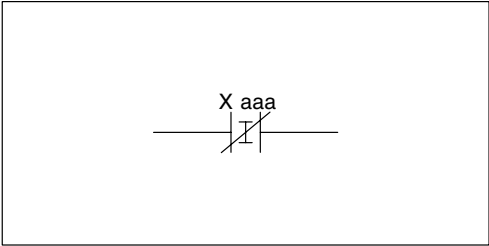
✓

✓

✓

430 440 450

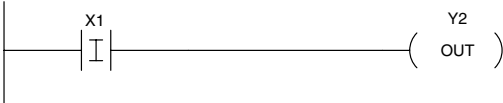
The Store Not Immediate instruction begins a new rung or additional branch in a rung. The status of the contact will be opposite the status of the associated input point on the module *at the time the instruction is executed*. The image register is not updated.



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

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

DirectSOFT32



Handheld Programmer Keystrokes

STR

SHFT

I

SHFT

X(IN)

1

←

OUT

Y(OUT)

2

←

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

DirectSOFT32



Handheld Programmer Keystrokes

STR

NOT

SHFT

I

SHFT

X(IN)

1

←

OUT

Y(OUT)

2

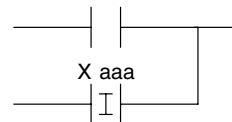
←

Or Immediate (ORI)



430 440 450

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

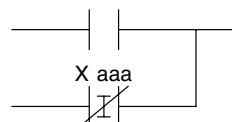


Or Not Immediate (ORNI)



430 440 450

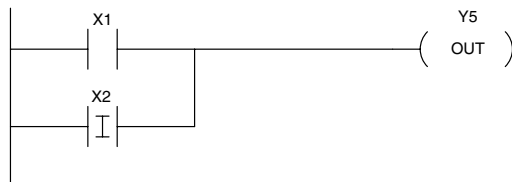
The Or Not Immediate connects two contacts in parallel. The status of the contact will be opposite the status of the associated input point on the module *at the time the instruction is executed*. The image register is not updated.



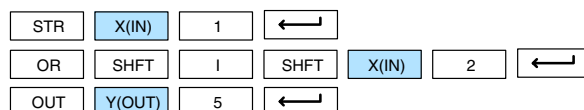
Operand Data Type	DL430 Range	DL440 Range	DL450 Range
	aaa	aaa	aaa
Inputs X	0-477	0-477	0-1777

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

DirectSOFT32

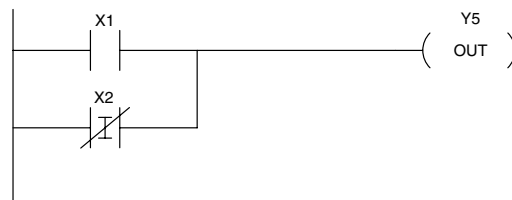


Handheld Programmer Keystrokes

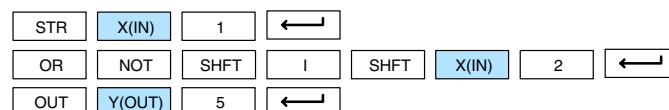


In the following example, when X1 is on or X2 is off, Y5 will energize.

DirectSOFT32



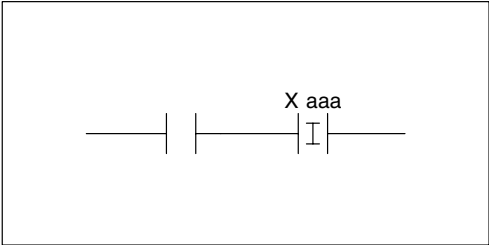
Handheld Programmer Keystrokes



And Immediate
(ANDI)

✓✓✓
430 440 450

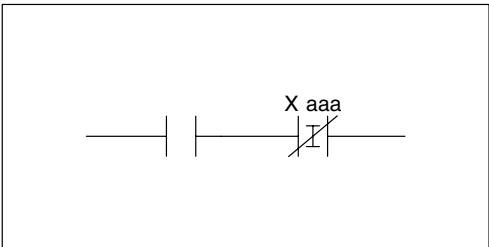
The And Immediate connects two contacts in series. The status of the contact will be the same as the status of the associated input point on the module *at the time the instruction is executed*. The image register is not updated.



And Not Immediate
(ANDNI)

✓✓✓
430 440 450

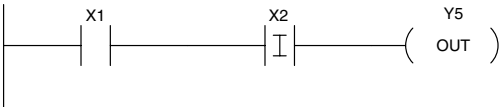
The And Not Immediate connects two contacts in series. The status of the contact will be opposite the status of the associated input point on the module *at the time the instruction is executed*. The image register is not updated.



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

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

DirectSOFT32

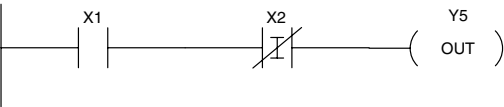


Handheld Programmer Keystrokes

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

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

DirectSOFT32



Handheld Programmer Keystrokes

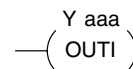
STR	X(IN)	1	←
AND	NOT	SHFT	I SHFT X(IN) 2 ←
OUT	Y(OUT)	5	←

Out Immediate (OUTI)



430 440 450

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.

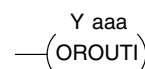


Or Out Immediate (OROUTI)



430 440 450

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.



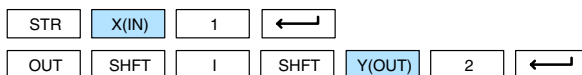
Operand Data Type	DL430 Range	DL440 Range	DL450 Range
	aaa	aaa	aaa
Outputs Y	0-477	0-477	0-1777

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

DirectSOFT32

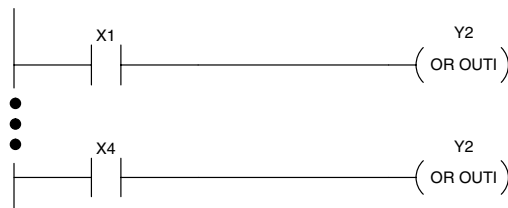


Handheld Programmer Keystrokes

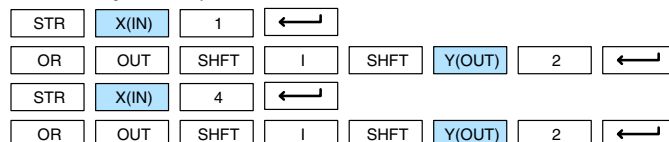


In the following example, when X1 or X4 is on, Y2 will energize.

DirectSOFT32



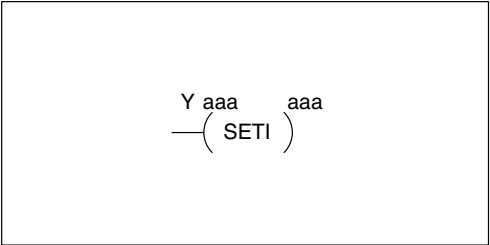
Handheld Programmer Keystrokes



Set Immediate
(SETI)

✓✓✓
430 440 450

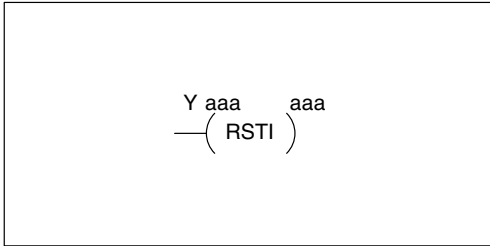
The Set Immediate instruction immediately sets or turns on an output or a range of outputs and the corresponding output module(s) *at the time the instruction is executed*. The image register is not updated. Once the outputs are set it is not necessary for the input to remain on. The Reset Immediate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



Reset
Immediate
(RSTI)

✓✓✓
430 440 450

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



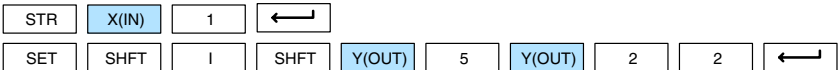
Operand Data Type	DL430 Range	DL440 Range	DL450 Range
	aaa	aaa	aaa
Outputs Y	0–477	0–477	0–1777

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

DirectSOFT32



Handheld Programmer Keystrokes

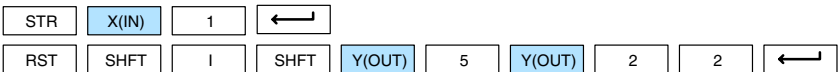


In the following example, when X1 is on, Y5 through Y22 will be reset (off) for the corresponding output module(s).

DirectSOFT32



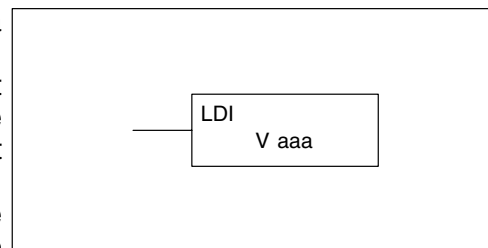
Handheld Programmer Keystrokes



Load Immediate (LDI)

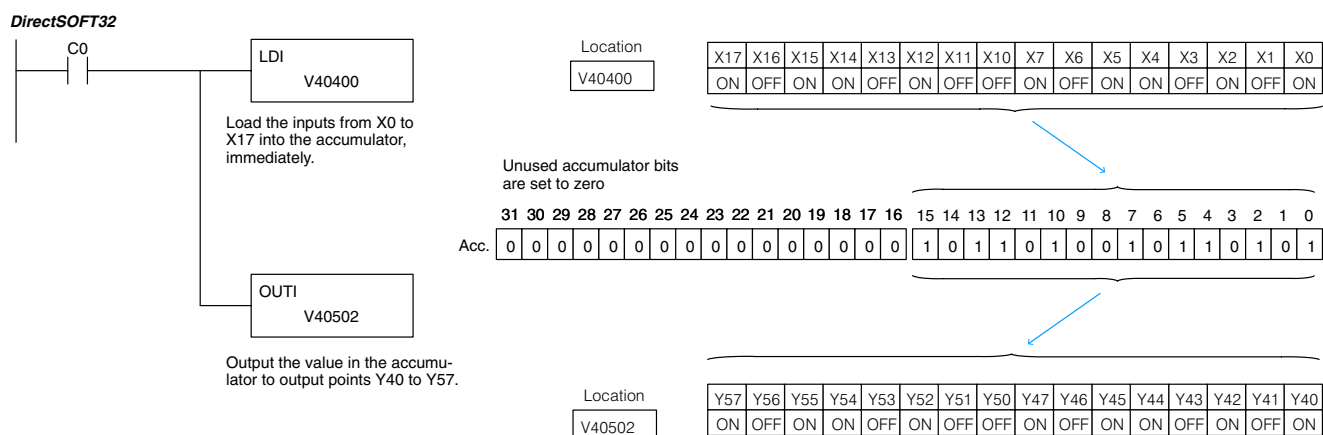


The Load Immediate instruction loads a 16-bit V-memory value into the accumulator. The valid address range includes all input point addresses on the local base. The value reflects the current status of the input points *at the time the instruction is executed*. This instruction may be used instead of the LDIF instruction which requires you to specify the number of input points.



Operand Data Type	DL450 Range
	aaaaa
Inputs V	40400 – 40477

In the following example, when C0 is on, the binary pattern of X10–X17 will be loaded into the accumulator using the Load Immediate instruction. The Out Immediate instruction could be used to copy the 16 bits in the accumulator to output points, such as Y40–Y57. This technique is useful to quickly copy an input pattern to output points (without waiting on a full CPU scan to occur).



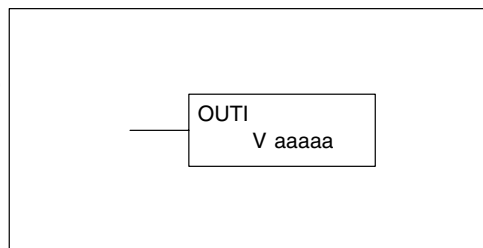
Handheld Programmer Keystrokes

STR	C(CR)	0	←								
LD	SHFT	I	SHFT	V	4	0	4	0	0	←	
OUT	SHFT	I	SHFT	V	4	0	5	4	0	←	

Out Immediate (OUTI)

X	X	✓
430	440	450

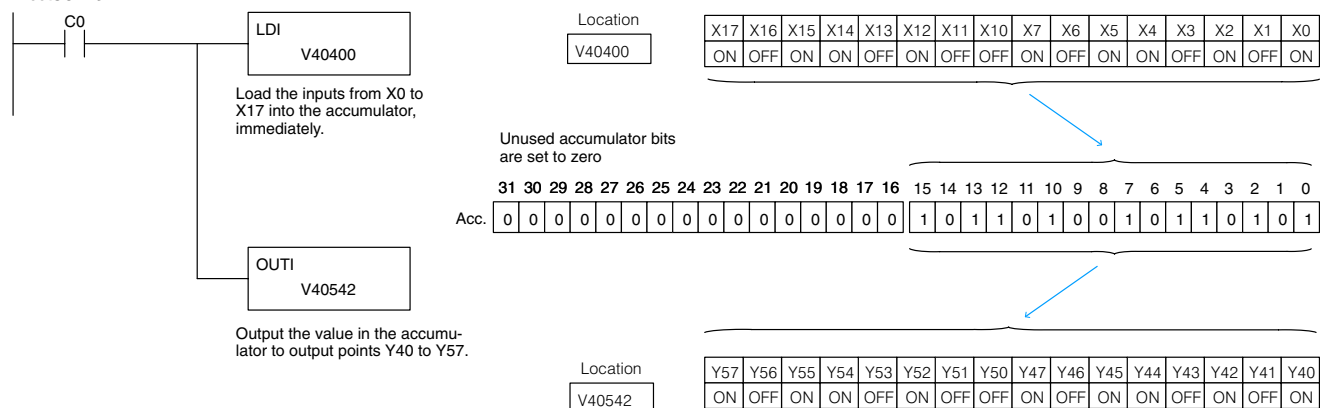
The Out Immediate instruction outputs a 16-bit binary value from the accumulator to a V-memory address *at the time the instruction is executed*. The valid address range includes all output point addresses on the local base. This instruction may be used instead of the OUTIF instruction which requires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



Operand Data Type	DL450 Range
	aaaaa
Inputs V	40400 – 40477

In the following example, when C0 is on, the binary pattern of X10–X17 will be loaded into the accumulator using the Load Immediate instruction. The Out Immediate instruction could be used to copy the 16 bits in the accumulator to output points, such as Y40–Y57. This technique is useful to quickly copy an input pattern to outputs (without waiting on the CPU scan).

DirectSOFT32



Handheld Programmer Keystrokes

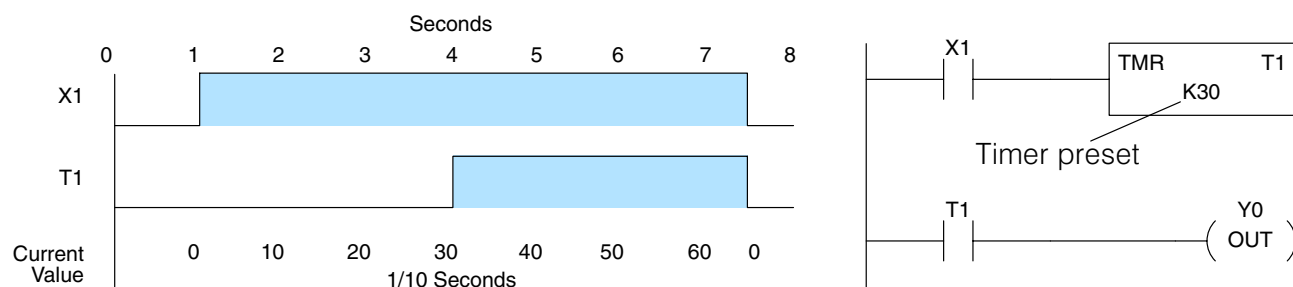
STR	C(CR)	0	←
LD	SHFT	I	SHFT
OUT	SHFT	I	SHFT

Timer, Counter, and Shift Register Instructions

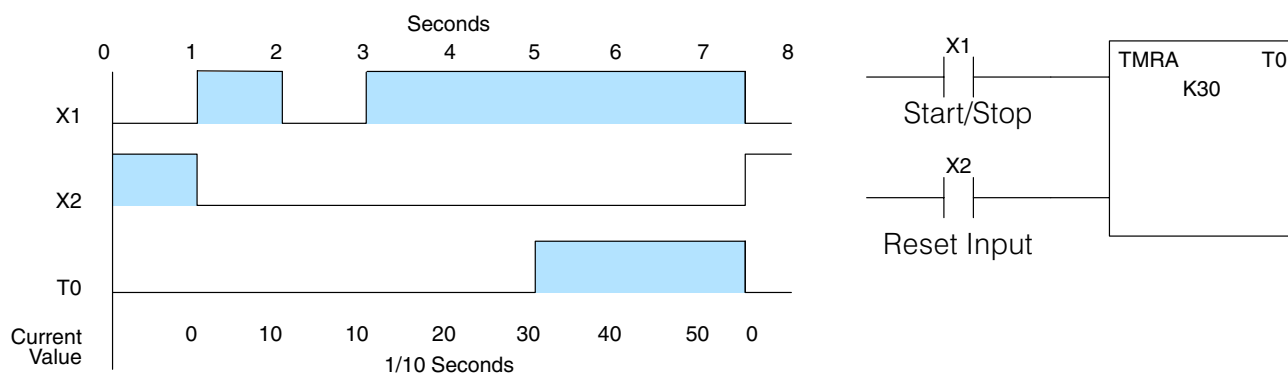
Using Timers

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)**

✓ ✓ ✓
430 440 450

The Timer instruction is a 0.1 second single input timer that times to a maximum of 999.9 seconds. The Timer Fast instruction is a 0.01 second single input timer that times up to a maximum of 99.99 seconds. These timers will be enabled if the input logic is true (on) and will be reset to 0 if the input logic is false (off).

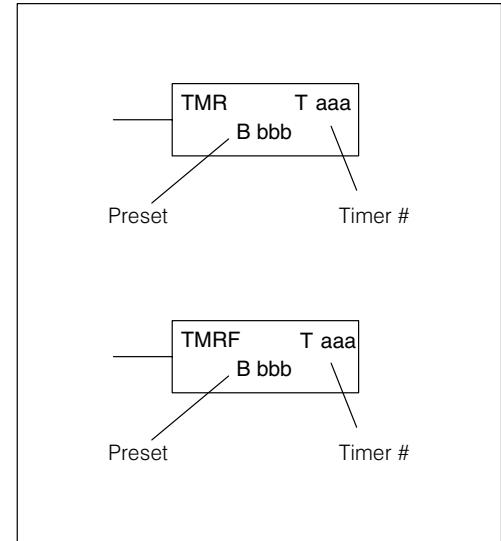
Instruction Specifications

Timer Reference (Taaa): Specifies the timer number.

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

Current Value: Timer current values are accessed by referencing the associated V or T memory location*. For example, the timer current value for T3 physically resides in V-memory location V3. (V locations are 16-bit words.)

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.



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

Operand Data Type	DL430 Range		DL440 Range		DL450 Range	
B	aaa	bbb	aaa	bbb	aaa	bbb
Timers	T	0-177	—	0-377	—	0-377
Vmemory for preset values	V	—	1400-7377	—	1400-7377 10000-17777	—
Pointers (preset only)	P	—	—	1400-7377 10000-17777	—	1400-7377 10000-17777
Constants (preset only)	K	—	0-9999	—	0-9999	—
Timer discrete status bits	T	0-177		0-377		0-377
Timer current values	V / T*	0-177		0-377		0-377

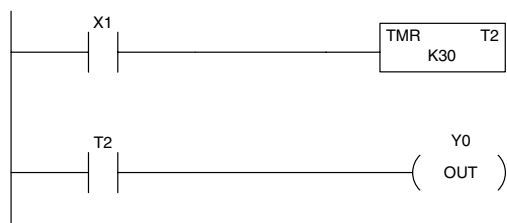
There are two methods of programming timers. You can perform functions when the timer reaches the specified preset using the discrete status bit, or use the comparative contacts to perform functions at different time intervals based on one timer. The following examples show each method of using timers.

NOTE: * For the Handheld Programmer, both the Timer discrete status bits and current value can be accessed with the same data type (example T2). The way the data type is used determines if it is a status bit or a current value. Any comparative instruction using T2 will access the current value, all other instructions using T2 will access the status bit. Current values may also be accessed by the V-memory location. For **DirectSOFT32**, the use of T2 will refer to the timer's discrete status bit. You should use V2 (or the alias TA2) to refer to the current value.

Timer Example Using Discrete Status Bits

In the following example, a single input timer is used with a preset of 3 seconds. The timer discrete status bit (T2) will turn on when the timer has timed for 3 seconds. The timer is reset when X1 turns off after 7.5 seconds turning the discrete status bit off and resetting the timer current value to 0.

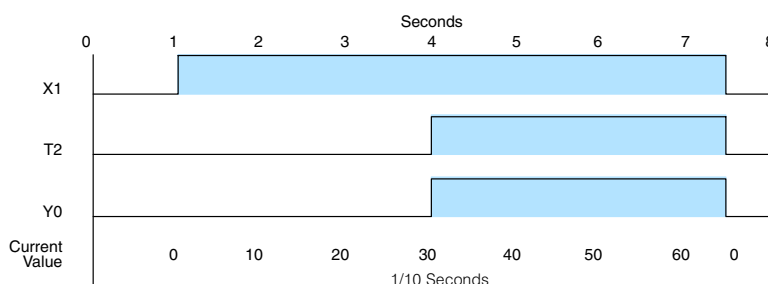
DirectSOFT32



Handheld Programmer Keystrokes

STR	X(IN)	1	←
TMR	TMR	2	K(CON) 3 0 ←
STR	TMR	2	←
OUT	Y(OUT)	0	←

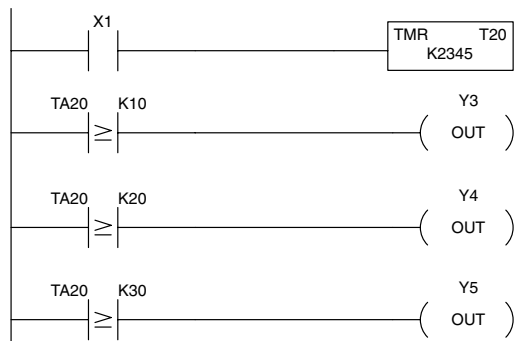
Timing Diagram



Timer Example Using Comparative Contacts

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

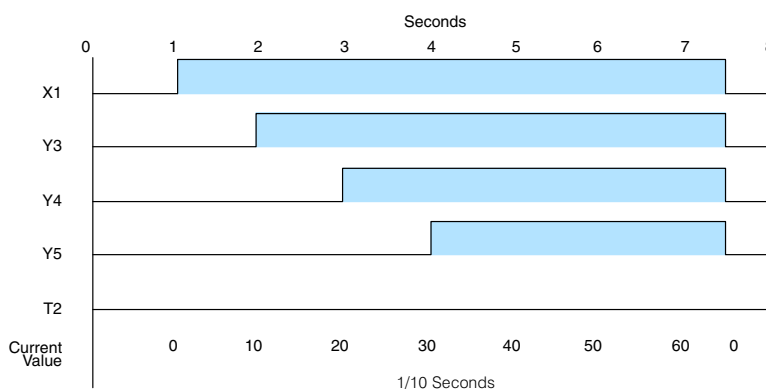
DirectSOFT32 (see Note)



Handheld Programmer Keystrokes

STR	X(IN)	1	←
TMR	TMR	2	0 K(CON) 2 3 4 5 ←
STR	TMR	2	0 K(CON) 1 0 ←
OUT	Y(OUT)	3	←
STR	TMR	2	0 K(CON) 2 0 ←
OUT	Y(OUT)	4	←
STR	TMR	2	0 K(CON) 3 0 ←
OUT	Y(OUT)	5	←

Timing Diagram



NOTE: Since this representation is showing a *DirectSOFT32* example, you would use the alias TA20 (or V20) instead of T20, which would be necessary for the equivalent rung entered with the Handheld Programmer.

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

✓ ✓ ✓
430 440 450

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

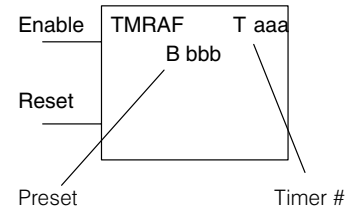
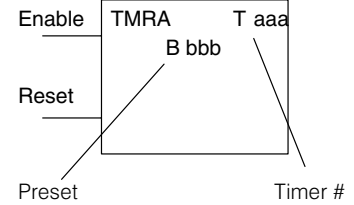
Instruction Specifications

Timer Reference (Taaa): Specifies the timer number.

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

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

Discrete Status Bit: The discrete status bit is accessed by referencing the associated T memory location. It will be on if the current value is equal to or greater than the preset value. For example the discrete status bit for timer 2 would be T2.



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

Caution: The TMRA uses two consecutive timer locations, since the preset can now be 8 digits, which requires two V memory locations. For example, if TMRA T0 is used in the program, the next available timer is T2. Or if T0 was a normal timer, and T1 was an accumulating timer, then the next available timer would be T3.

Operand Data Type	DL430 Range		DL440 Range		DL450 Range	
B	aaa	bbb	aaa	bbb	aaa	bbb
Timers T	0-176	—	0-376	—	0-376	—
Vmemory for preset values V	—	1400-7377	—	1400-7377 10000-17777	—	1400-7377 10000-17777
Pointers (preset only) P	—	—	—	1400-7377 10000-17777	—	1400-7377 10000-17777
Constants (preset only) K	—	0-99999999	—	0-99999999	—	0-99999999
Timer discrete status bits T	0-177		0-377		0-377	
Timer current values V/T*	0-177		0-377		0-377	

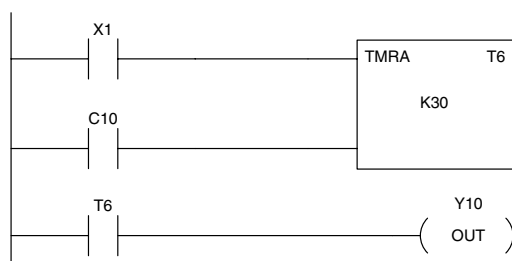
There are two methods of programming timers. You can perform functions when the timer reaches the specified preset using the the discrete status bit, or use the comparative contacts to perform functions at different time intervals based on one timer. The following examples show each method of using timers.

NOTE:* For the Handheld Programmer, both the Timer discrete status bits and current value can be accessed with the same data type (example T2). The way the data type is used determines if it is a status bit or a current value. Any comparative instruction using T2 will access the current value, all other instructions using T2 will access the status bit. Current values may also be accessed by the V-memory location. For **DirectSOFT32**, the use of T2 will refer to the timer's discrete status bit. You should use V2 (or the alias TA2) to refer to the current value.

Accumulating Timer Example using Discrete Status Bits

In the following example, a two input timer (accumulating timer) is used with a preset of 3 seconds. The timer discrete status bit (T6) will turn on when the timer has timed for 3 seconds. Notice in this example the timer times for 1 second, stops for one second, then resumes timing. The timer will reset when C10 turn on after 5.5 seconds turning the discrete status bit off and resetting the timer current value to 0.

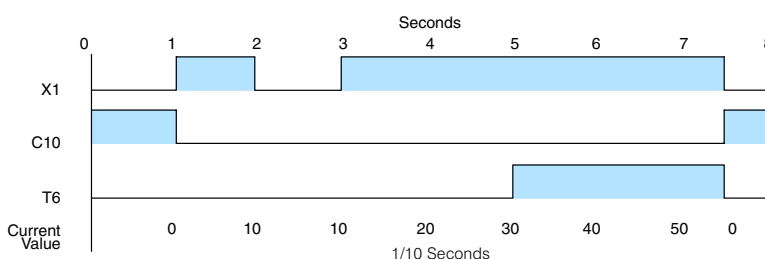
DirectSOFT32



Handheld Programmer Keystrokes

STR	X(IN)	1	←
STR	C(CR)	1	0 ←
TMR	SHFT	A	SHFT TMR 6 ←

Timing Diagram



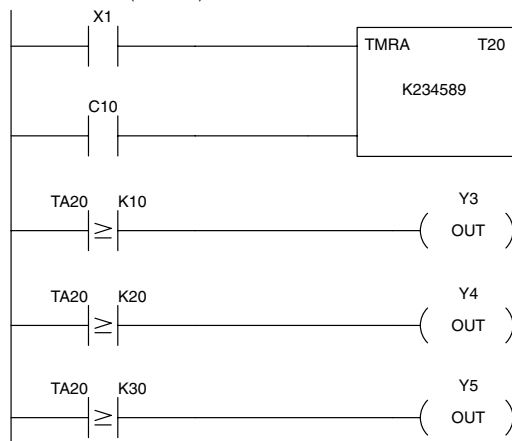
Handheld Programmer Keystrokes (cont)

K(CON)	3	0	←
STR	TMR	6	←
OUT	Y(OUT)	1	0 ←

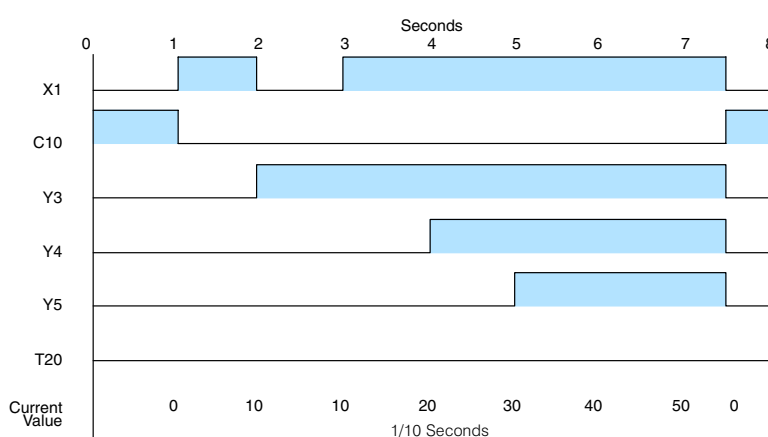
Accumulator Timer Example Using Comparative Contacts

In the following example, a single input timer is used with a preset of 23458.9 seconds. Comparative contacts are used to energized Y3, Y4, and Y5 at one second intervals respectively. The comparative contacts will turn off when the timer is reset.

DirectSOFT32 (see Note)



Timing Diagram



Handheld Programmer Keystrokes

STR	X(IN)	1	←
STR	C(CR)	1	0 ←
TMR	SHFT	A	SHFT TMR 2 0 ←
K(CON)	2	3	4 5 8 9 ←
STR	TMR	2	0 K(CON) 1 0 ←
OUT	Y(OUT)	3	←

Handheld Programmer Keystrokes (cont)

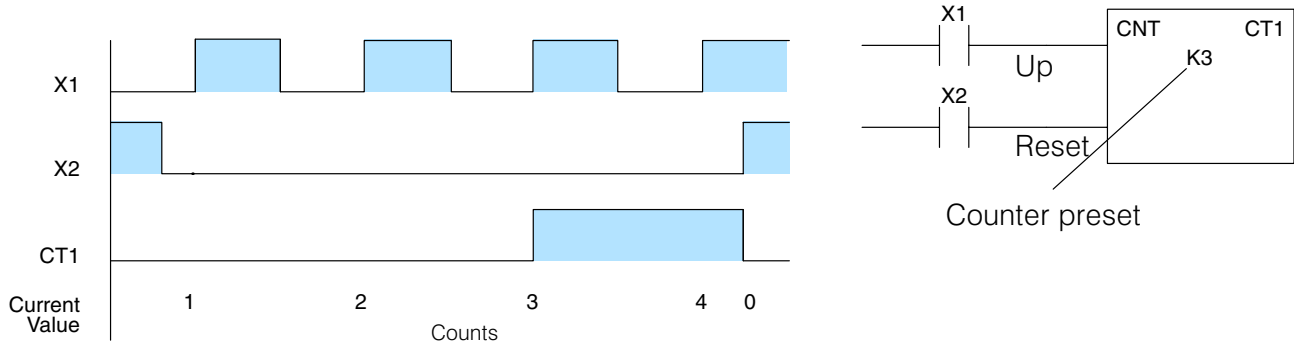
STR	TMR	2	0	K(CON)	2	0	←
OUT	Y(OUT)	4	←				
STR	TMR	2	0	K(CON)	3	0	←
OUT	Y(OUT)	5	←				

NOTE: Since this representation is showing a *DirectSOFT32* example, you would use the alias TA20 (or V20) instead of T20, which would be necessary for the equivalent rung entered with the Handheld Programmer.

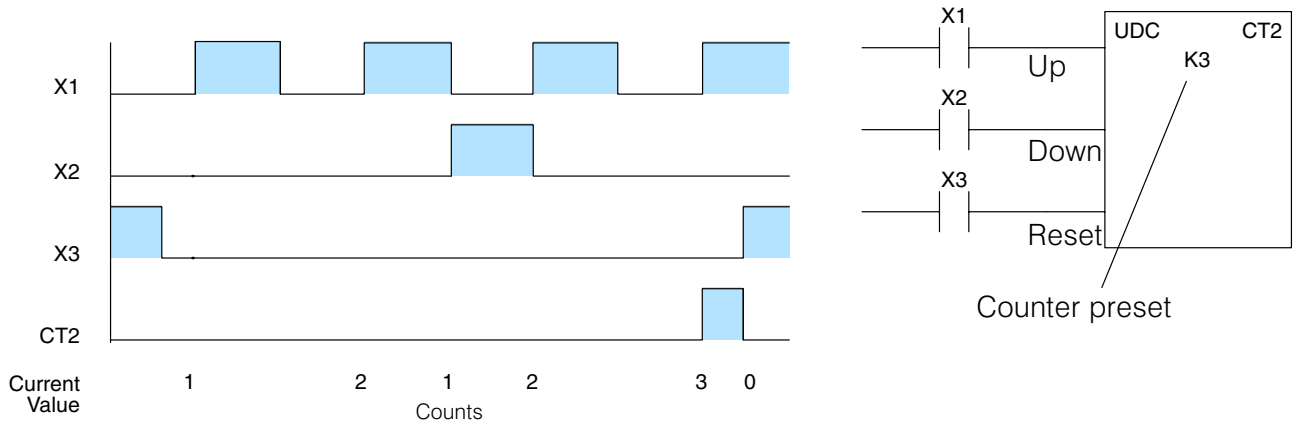
Using Counters

Counters are used to count events. The counters available are up counters, up/down counters, and stage counters (used with RLL^{PLUS} programming).

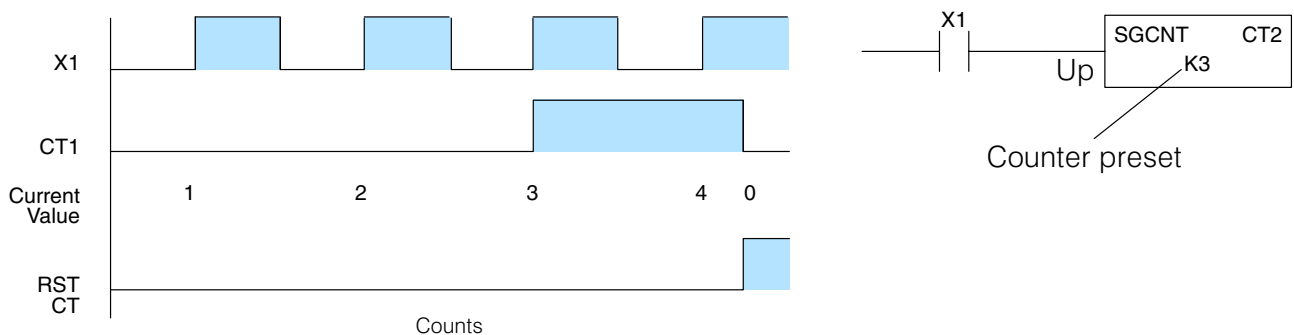
The up counter has two inputs, a count input and a reset input. The maximum count value is 9999. The timing diagram below shows the relationship between the counter input, counter reset, associated discrete bit, current value, and counter preset.



The up down counter has three inputs, a count up input, count down input and reset input. The maximum count value is 99999999. The timing diagram below shows the relationship between the counter input, counter reset, associated discrete bit, current value, and counter preset.



The stage counter has a count input and is reset by the RST instruction. This instruction is useful when programming RLL^{PLUS}, by allowing you to reference the same counter from multiple stages. The maximum count value is 9999. The timing diagram below shows the relationship between the counter input, associated discrete bit, current value, counter preset and reset instruction.



**Counter
(CNT)**

430 440 450

The Counter is a two input counter that increments when the count input logic transitions from off to on. When the counter reset input is on the counter resets to 0. When the current value equals the preset value, the counter status bit comes on and the counter continues to count up to a maximum count of 9999. The maximum value will be held until the counter is reset.

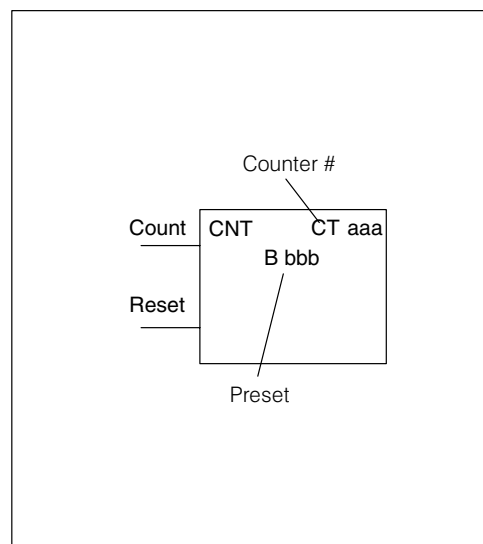
Instruction Specifications

Counter Reference (CTaaa): Specifies the counter number.

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

Current Values: Counter current values are accessed by referencing the associated V or CT memory locations*. The V-memory location is the counter location + 1000. For example, the counter current value for CT3 resides in V memory location V1003. (V locations are 16-bit words.)

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.



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

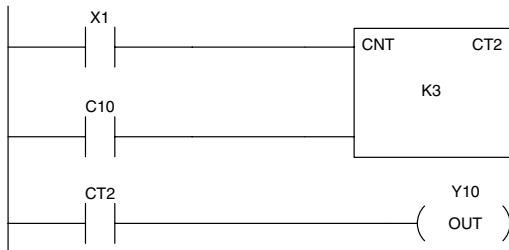
Operand Data Type		DL430 Range		DL440 Range		DL450 Range	
B		aaa	bbb	aaa	bbb	aaa	bbb
Counters	CT	0-177	—	0-177	—	0-377	—
Vmemory (preset only)	V	—	1400-7377	—	1400-7377 10000-17777	—	1400-7377 10000-37777
Pointers (preset only)	P	—	—	—	1400-7377 10000-17777	—	1400-7377 10000-37777
Constants (preset only)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

NOTE:* For the Handheld Programmer, both the Counter discrete status bits and current value can be accessed with the same data type (example CT2). The way the data type is used determines if it is a status bit or a current value. Any comparative instruction using CT2 will access the current value, all other instructions using CT2 will access the status bit. Current values may also be accessed by the V-memory location. For **Direct**SOFT32, the use of CT2 will refer to the timer's discrete status bit. You should use V1002 (or the alias CTA2) to refer to the current value.

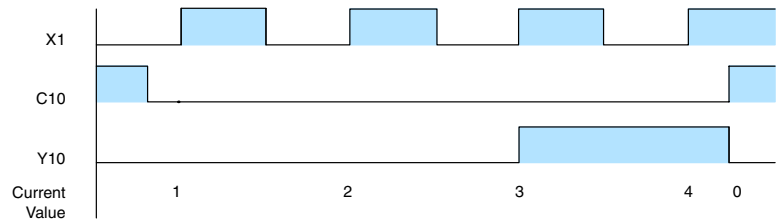
Counter Example Using Discrete Status Bits

In the following example, when X1 makes an off to on transition, counter CT2 will increment by one. When the current value reaches the preset value of 3, the counter status bit CT2 will turn on and energize Y10. When the reset C10 turns on, the counter status bit will turn off and the current value will be 0. The current value for counter CT2 will be held in Vmemory location V1002.

DirectSOFT32



Counting diagram



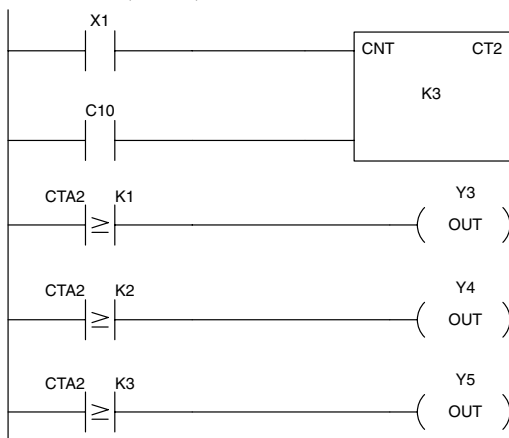
Handheld Programmer Keystrokes

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

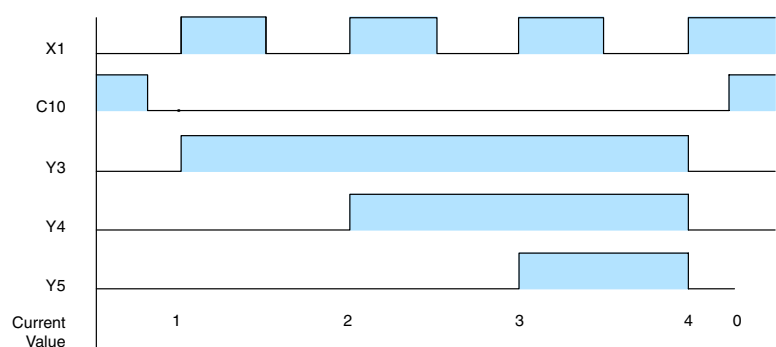
Counter Example Using Comparative Contacts

In the following example, when X1 is makes an off to on transition, counter CT2 will increment by one. Comparative contacts are used to energize Y3, Y4, and Y5 at different counts. The comparative contacts will turn off when the counter is reset. When the reset C10 turns on, the counter status bit will turn off and the counter current value will be 0.

DirectSOFT32 (see Note)



Counting diagram



Handheld Programmer Keystrokes

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

Handheld Programmer Keystrokes (cont)

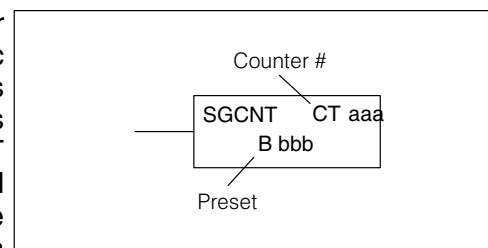
STR	CNT	2	K(CON) 2 ←
OUT	Y(OUT)	4	←
STR	CNT	2	K(CON) 3 ←
OUT	Y(OUT)	5	←

NOTE: Since this representation is showing a **DirectSOFT32** example, you would use the alias CTA2 (or V1002) instead of CT2, which would be necessary for the equivalent rung entered with the Handheld.

Stage Counter (SGCNT)

430 440 450

The Stage Counter is a single input counter that increments when the input logic transitions from off to on. This counter differs from other counters since it will hold its current value until reset using the RST instruction. The Stage Counter is designed for use in **RLL^{PLUS}** programs but can be used in relay ladder logic programs. When the current value equals the preset value, the counter status bit turns on and the counter continues to count up to a maximum count of 9999. The maximum value will be held until the counter is reset.



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

Instruction Specifications

Counter Reference (CTaaa): Specifies the counter number.

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

Current Values: Counter current values are accessed by referencing the associated V or CT memory locations*. The V-memory location is the counter location + 1000. For example, the counter current value for CT3 resides in V memory location V1003. (V locations are 16-bit words.)

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

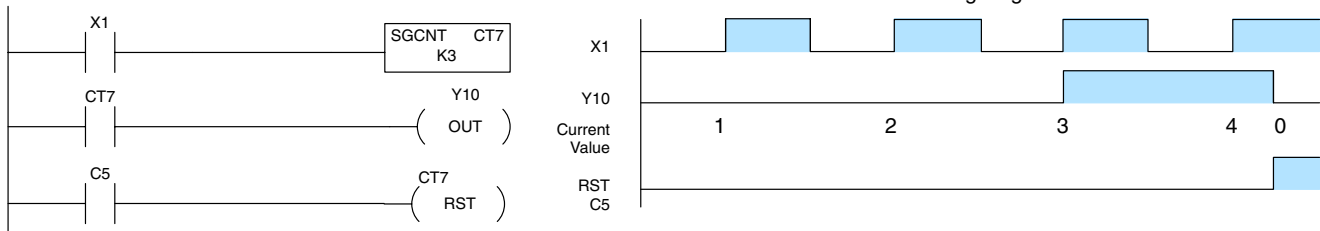
Operand Data Type		DL430 Range		DL440 Range		DL450 Range	
B		aaa	bbb	aaa	bbb	aaa	bbb
Counters	CT	0-177	—	0-177	—	0-377	—
Vmemory	V	—	1400-7377	—	1400-7377 10000-17777	—	1400-7377 10000-17777
Pointers (preset only)	P	—	—	—	1400-7377 10000-17777	—	1400-7377 10000-37777
Constants	K	—	0-9999	—	0-9999	—	0-9999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

NOTE:* For the Handheld Programmer, both the Stage Counter discrete status bits and current value can be accessed with the same data type (example CT2). The way the data type is used determines if it is a status bit or a current value. Any comparative instruction using CT2 will access the current value, all other instructions using CT2 will access the status bit. Current values may also be accessed by the V-memory location. For **DirectSOFT32**, the use of CT2 will refer to the timer's discrete status bit. You should use V1002 (or the alias CTA2) to refer to the current value.

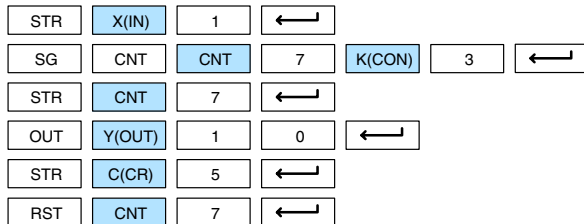
Stage Counter Example Using Discrete Status Bits

In the following example, when X1 makes an off to on transition, stage counter CT7 will increment by one. When the current value reaches 3, the counter status bit CT7 will turn on and energize Y10. The counter status bit CT7 will remain on until the counter is reset using the RST instruction. When the counter is reset, the counter status bit will turn off and the counter current value will be 0. The current value for counter CT7 will be held in Vmemory location V1007.

DirectSOFT32



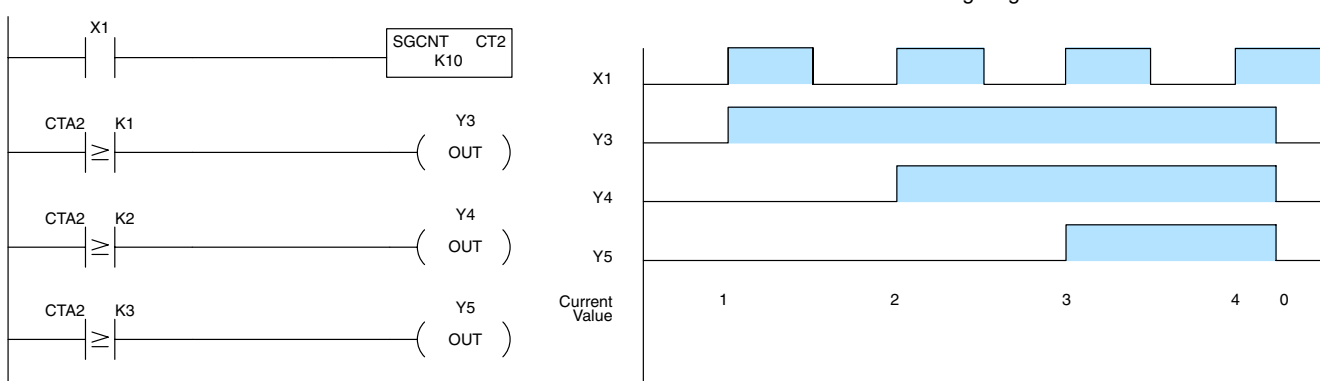
Handheld Programmer Keystrokes



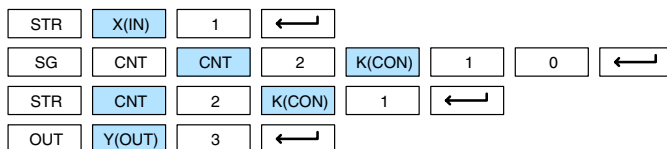
Stage Counter Example Using Comparative Contacts

In the following example, when X1 makes an off to on transition, counter CT2 will increment by one. Comparative contacts are used to energize Y3, Y4, and Y5 at different counts. Although this is not shown in the example, when the counter is reset using the Reset instruction, the counter status bit will turn off and the current value will be 0. The current value for counter CT2 will be held in Vmemory location V1002.

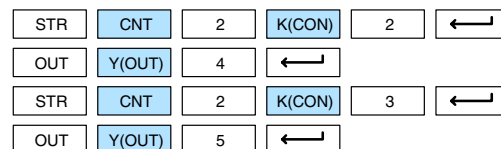
tSOFT32 (see Note)



Handheld Programmer Keystrokes



Handheld Programmer Keystrokes (cont)



NOTE: Since this representation is showing a **DirectSOFT32** example, you would use the alias CTA2 (or V1002) instead of CT2, which would be necessary for the equivalent rung entered with the Handheld.

Up/Down Counter (UDC)

430 440 450

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

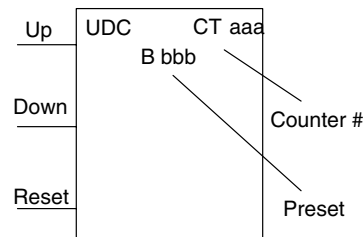
Instruction Specification

Counter Reference (CTaaa): Specifies the counter number.

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

Current Values: Current count is a double word value accessed by referencing the associated V or CT memory locations*. The V-memory location is the counter location + 1000. For example, the counter current value for CT5 resides in V memory location V1005 and V1006. (V locations are 16-bit words.)

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



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

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

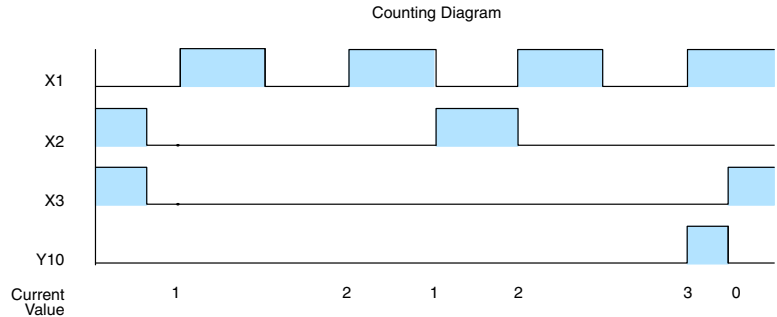
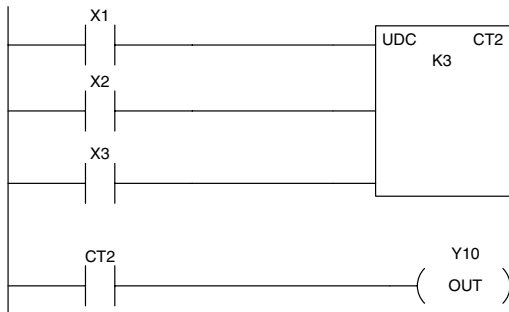
Operand Data Type		DL430 Range		DL440 Range		DL450 Range	
B		aaa	bbb	aaa	bbb	aaa	bbb
Counters	CT	0–176	—	0–176	—	0–376	—
Vmemory	V	—	1400–7377	—	1400–7377 10000–17777	—	1400–7377 10000–37777
Pointers (preset only)	P	—	—	—	1400–7377 10000–17777	—	1400–7377 10000–37777
Constants	K	—	0–99999999	—	0–99999999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

NOTE:* For the Handheld Programmer, both the Stage Counter discrete status bits and current value can be accessed with the same data type (example CT2). The way the data type is used determines if it is a status bit or a current value. Any comparative instruction using CT2 will access the current value, all other instructions using CT2 will access the status bit. Current values may also be accessed by the V-memory location. For **Direct**SOFT32, the use of CT2 will refer to the timer's discrete status bit. You should use V1002 (or the alias CTA2) to refer to the current value.

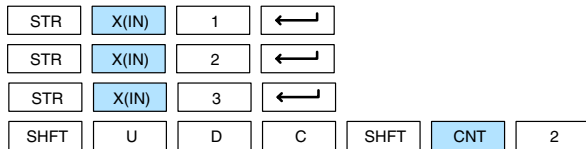
Up/Down Counter Example Using Discrete Status Bits

In the following example if X2 and X3 are off ,when X1 toggles from off to on the counter will increment by one. If X1 and X3 are off the counter will decrement by one when X2 toggles from off to on. When the count value reaches the preset value of 3, the counter status bit will turn on. When the reset X3 turns on, the counter status bit will turn off and the current value will be 0.

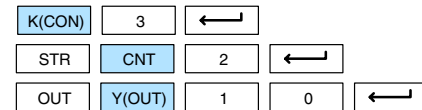
DirectSOFT32



Handheld Programmer Keystrokes



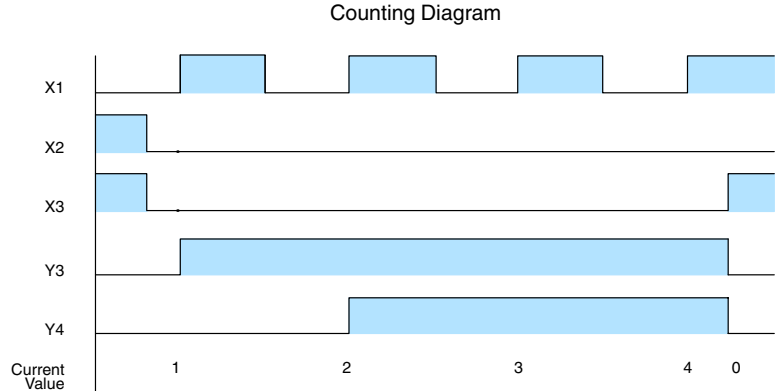
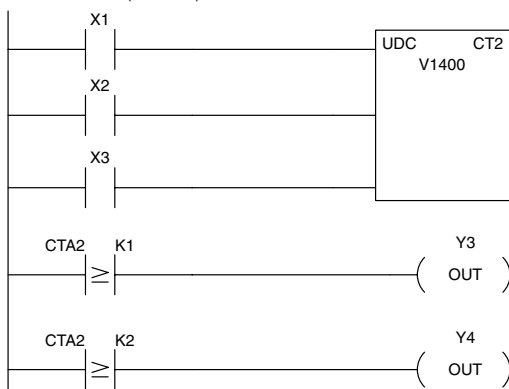
Handheld Programmer Keystrokes (cont)



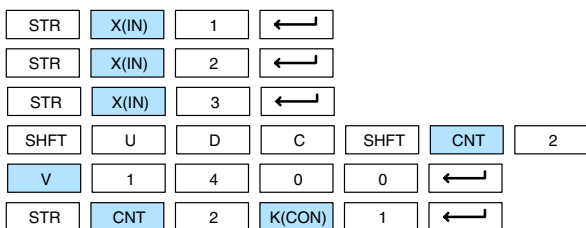
Up/Down Counter Example Using Comparative Contacts

In the following example, when X1 makes an off to on transition, counter CT2 will increment by one. Comparative contacts are used to energize Y3 and Y4 at different counts. The comparative contacts will turn off when the counter is reset. When the reset X3 turns on, the counter status bit will turn off and the current value will be 0.

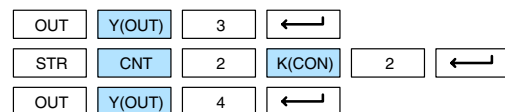
DirectSOFT32 (see Note)



Handheld Programmer Keystrokes



Handheld Programmer Keystrokes (cont)



NOTE: Since this representation is showing a *DirectSOFT32* example, you would use the alias CTA2 (or V1002) instead of CT2, which would be necessary for the equivalent rung entered with the Handheld.

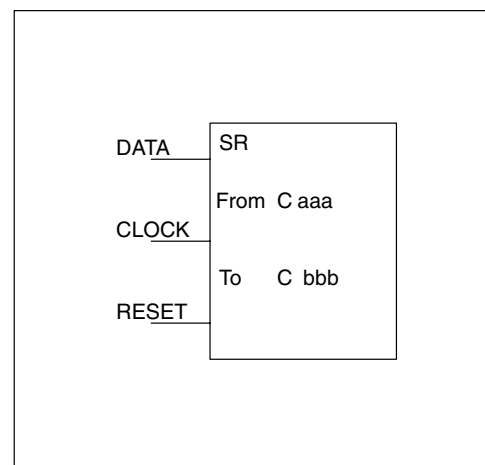
Shift Register (SR)

430 440 450

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.

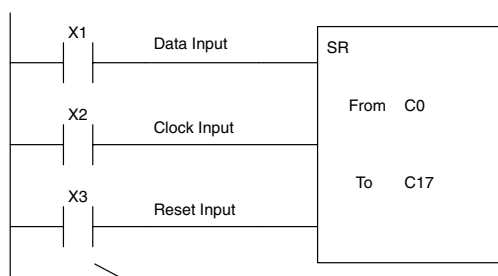
The Shift Register has three contacts.

- 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 lower address to higher address. From C17 to C0 would define a block of sixteen bits, to be shifted from higher address to lower address. The maximum size of the shift register block depends on the number of available control relays. The minimum block size is 8 control relays.

Operand Data Type	DL430 Range		DL440 Range		DL440 Range	
B	aaa	bbb	aaa	bbb	aaa	bbb
Control Relay C	0-737	0-737	0-1777	0-1777	0-3777	0-3777

DirectSOFT32**Handheld Programmer Keystrokes**

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

Inputs on Successive Scans**Shift Register Bits**

Data	Clock	Reset		C0	C17
1	⏏	0	—	1	
0	⏏	0	—	0	
0	⏏	0	—		0
1	⏏	0	—	1	0
0	⏏	0	—	0	0
—	—	1	—		

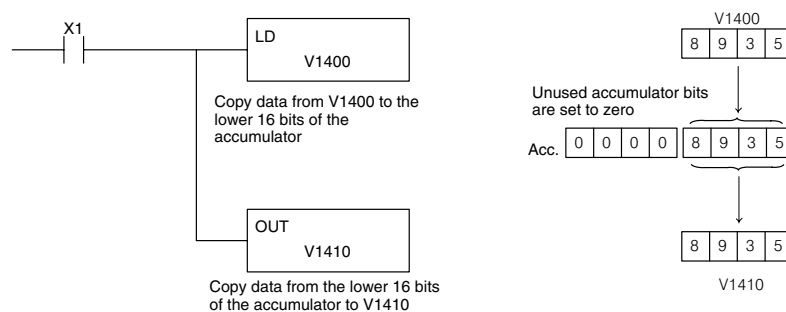
Accumulator / Data Stack Load and Output Instructions

Using the Accumulator

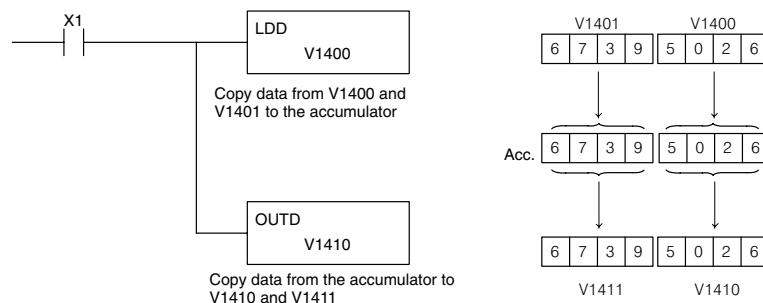
The accumulator in the DL405 series CPUs is a 32 bit register which is used as a temporary storage location for data that is being copied or manipulated in some manor. For example, you have to use the accumulator to perform math operations such as add, subtract, multiply, etc. Since there are 32 bits, 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 Vmemory location V1410.

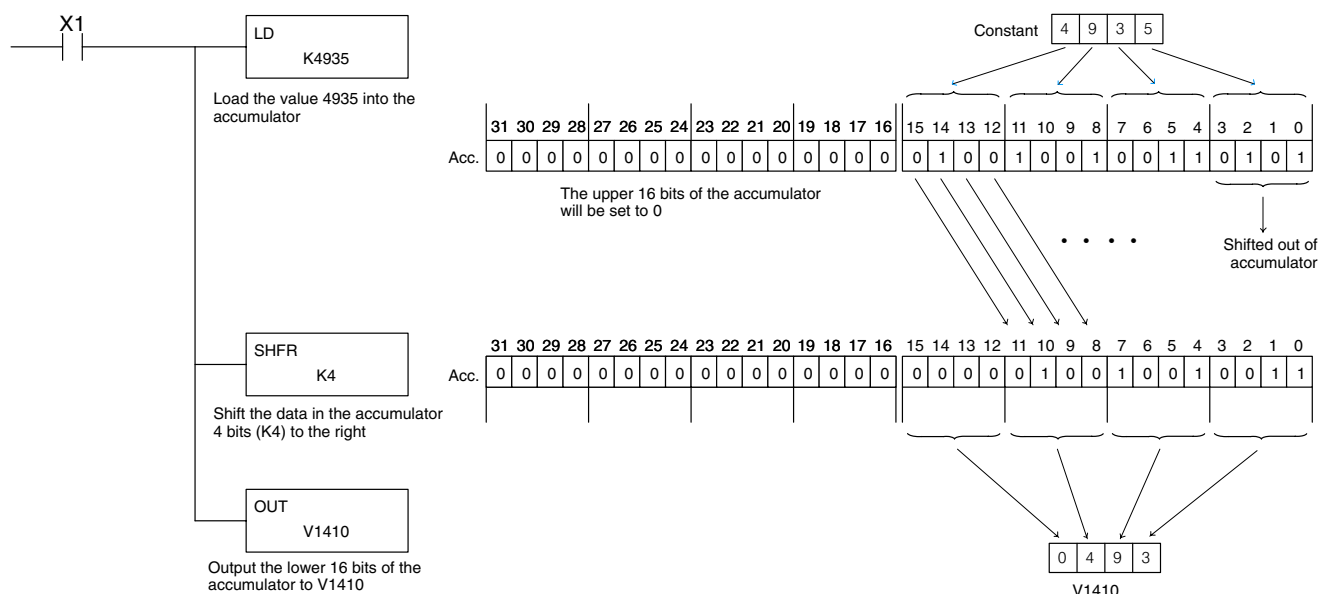


Since the accumulator is 32 bits and V memory locations are 16 bits the Load Double and Out Double (or variations thereof) use two consecutive V memory locations or 8 digit BCD constants to copy data either to the accumulator from a Vmemory address or from a Vmemory address to the accumulator. For example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

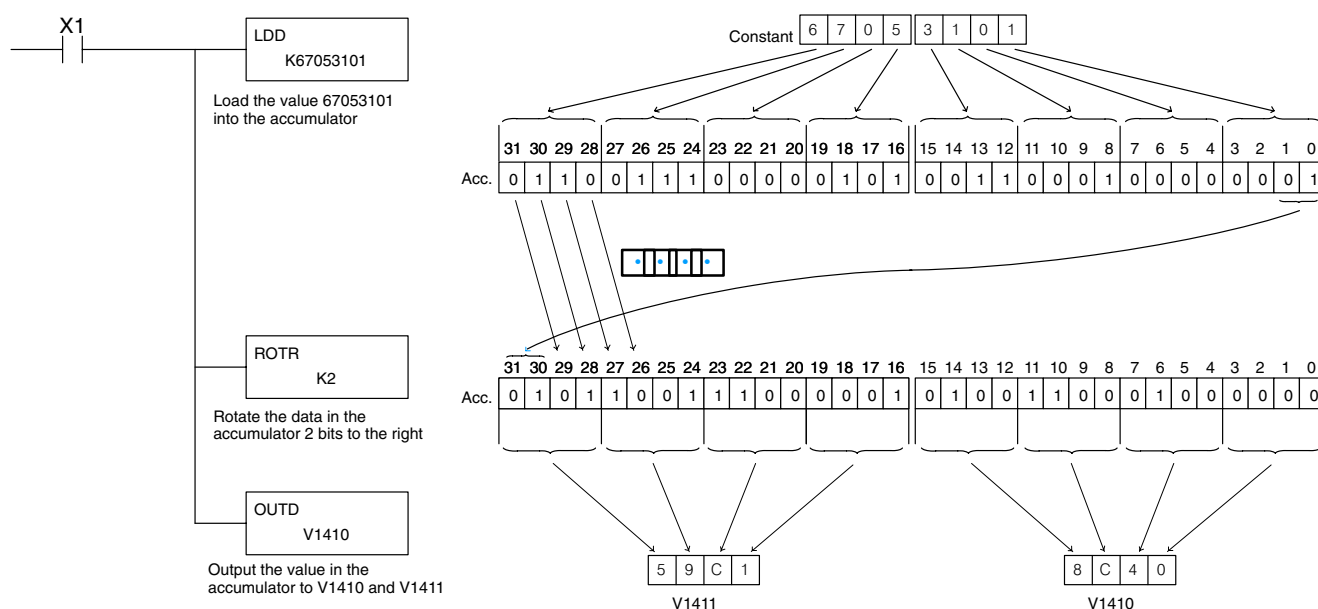


Changing the Accumulator Data

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.

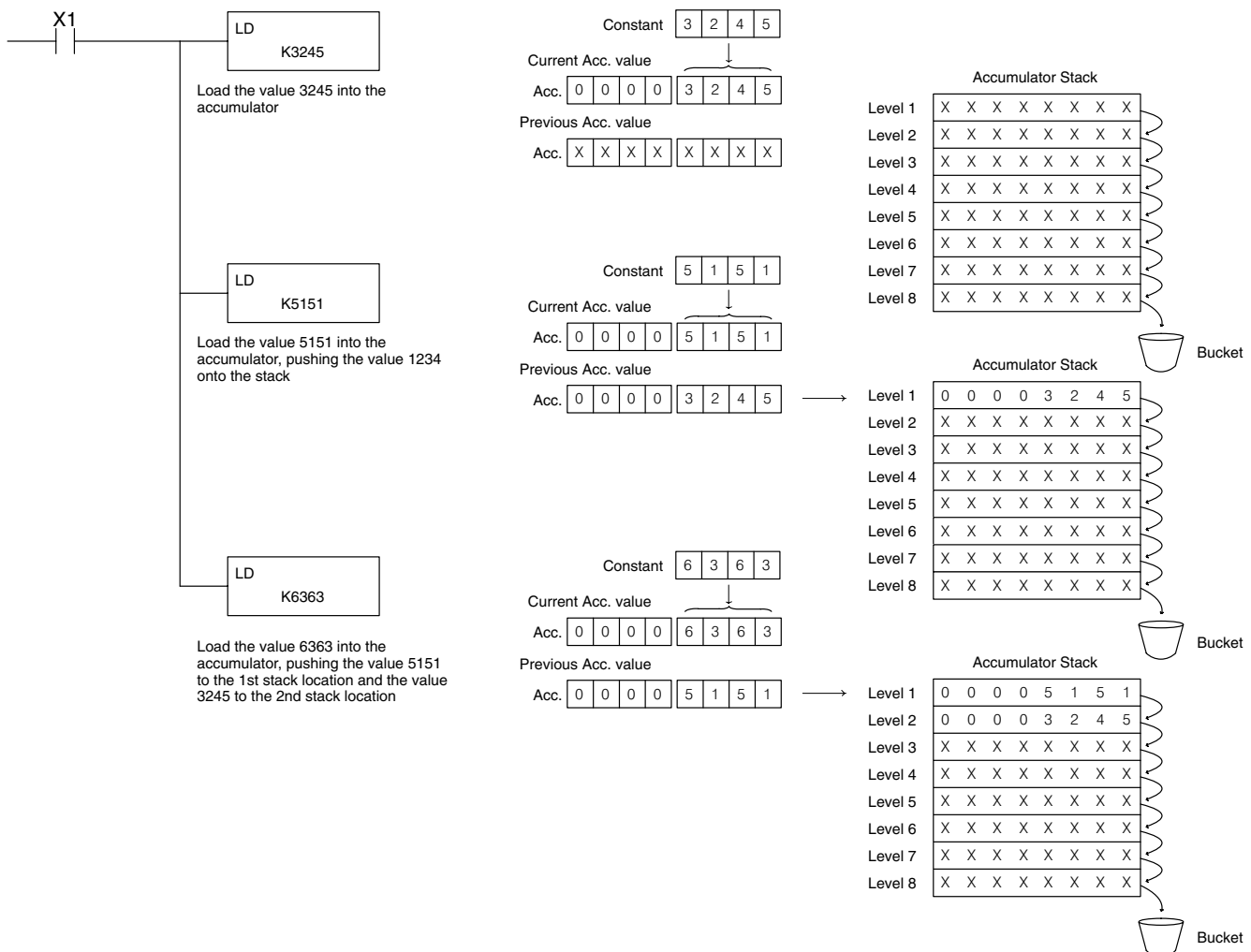


Some of the data manipulation instructions use 32 bits. They use two consecutive V memory locations or 8 digit BCD constants to manipulate data in the accumulator. The following example rotates the value 67053101 two bits to the right and outputs the value to V1410 and V1411.

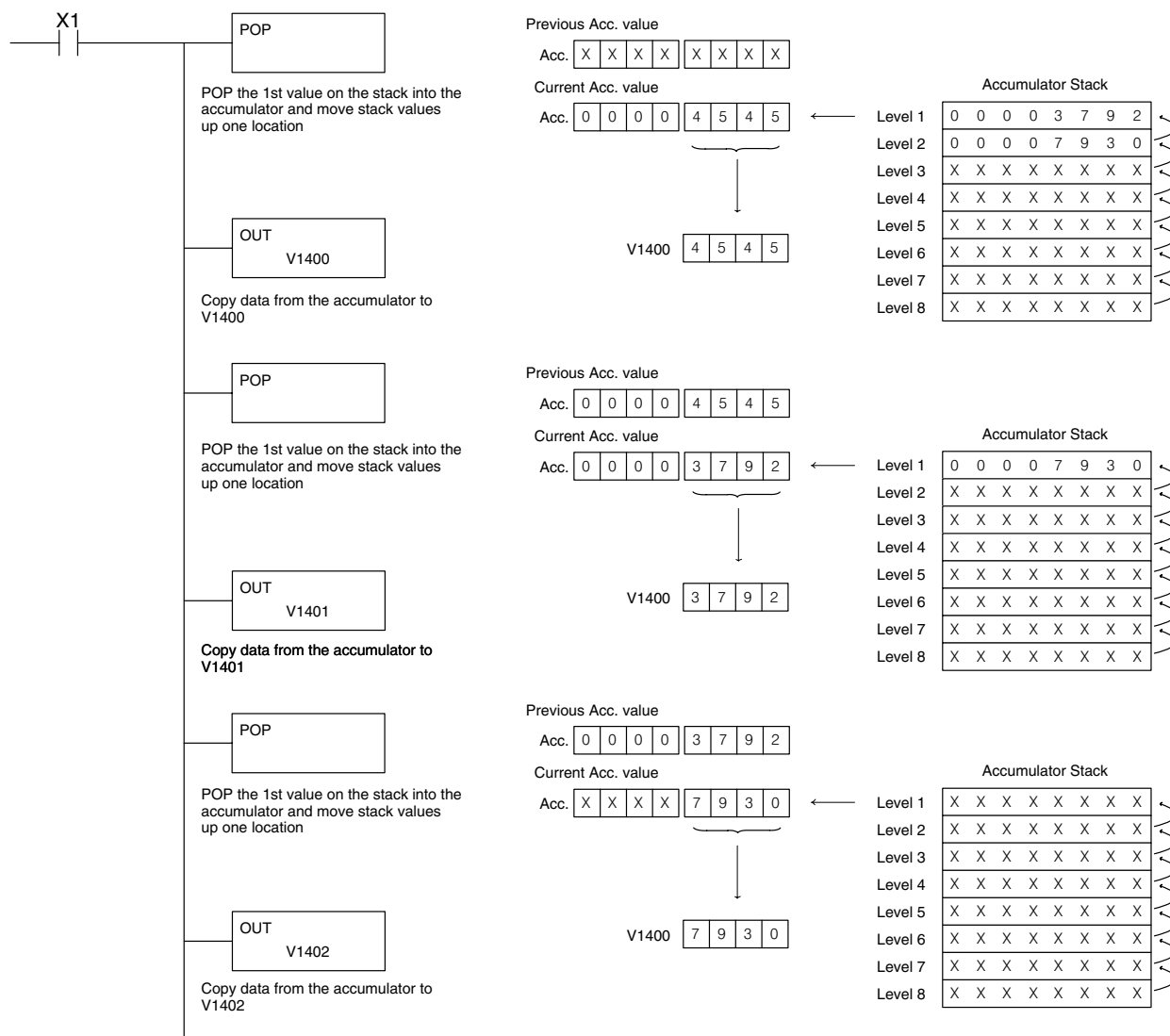


Using the Accumulator Stack

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.



Accumulator and Accumulator Stack Memory Locations

☒ 430
 ☒ 440
 ☒ 450

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

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

Accumulator								
0	0	0	0	3	7	9	2	
V701				V700				

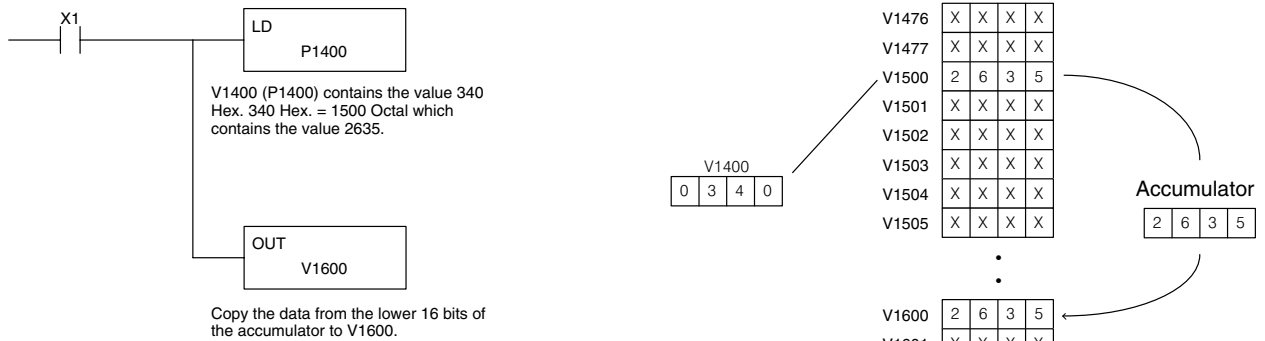
Accumulator Stack									
Level 1	0	0	0	0	3	7	9	2	V703 – V702
Level 2	0	0	0	0	7	9	3	0	V705 – V704
Level 3	X	X	X	X	X	X	X	X	V707 – V706
Level 4	X	X	X	X	X	X	X	X	V711 – V710
Level 5	X	X	X	X	X	X	X	X	V713 – V712
Level 6	X	X	X	X	X	X	X	X	V715 – V714
Level 7	X	X	X	X	X	X	X	X	V717 – V716
Level 8	X	X	X	X	X	X	X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

Using Pointers

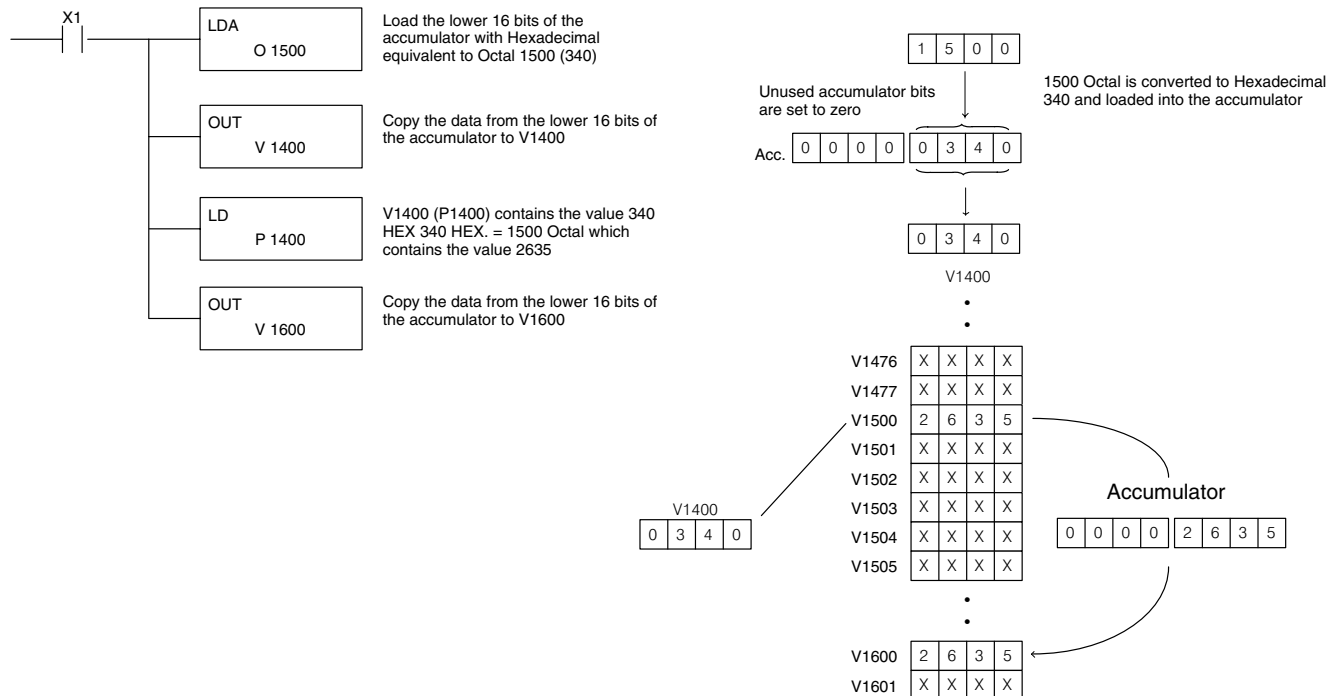
Many of the DL405 series instructions will allow Vmemory pointers as a operand. Pointers can be useful in ladder logic programming, but can be difficult to understand or implement in your application if you do not have prior experience with pointers (commonly known as indirect addressing). Pointers allow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

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

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

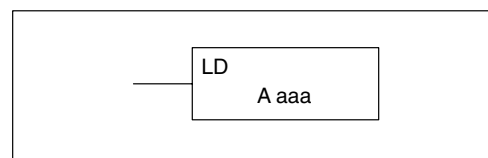


The following example is similar to the one above, except for the LDA (load address) instruction which automatically converts the Octal address to the Hex equivalent.

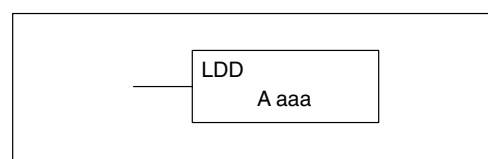


**Load
(LD)**

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

**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.

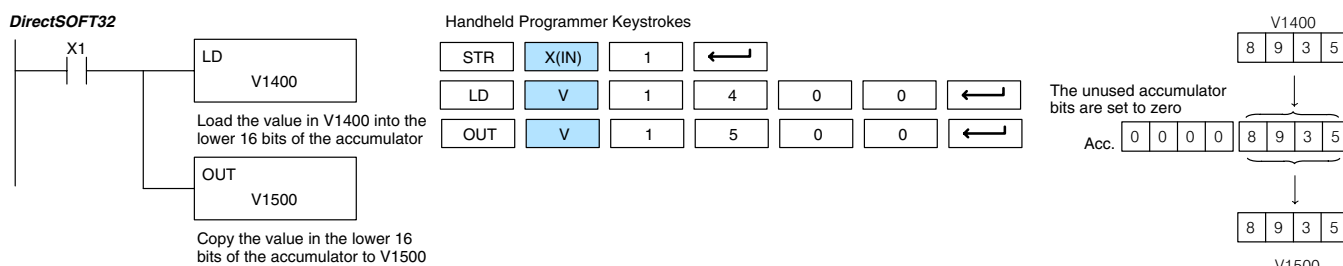


Operand Data Type		DL430 Range	DL440 Range	DL440 Range
A		aaa	aaa	aaa
Vmemory	V	All (See p. 3-40)	All (See p. 3-41)	All (See p. 3-42)
Pointer	P	All V mem (See p. 3-40)	All V mem (See p. 3-41)	All V mem (See p. 3-42)
Constant	K	0-FFFF	0-FFFF	0-FFFF

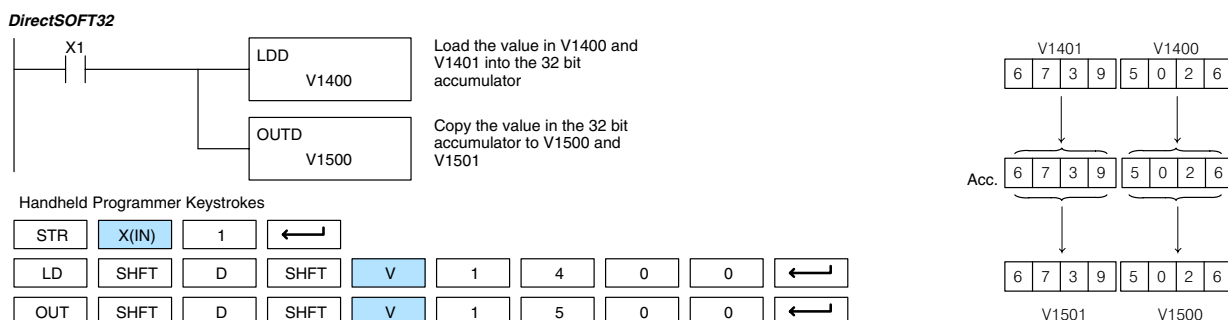
Discrete Bit Flags	Description
SP76	on when the value loaded into the accumulator by any instruction is zero.

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

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



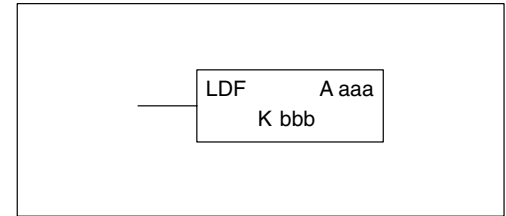
In the following example, when X1 is on, the 32 bit value in V1400 and V1401 will be loaded into the accumulator and output to V1500 and V1501.



Load Formatted (LDF)

430 440 450

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.



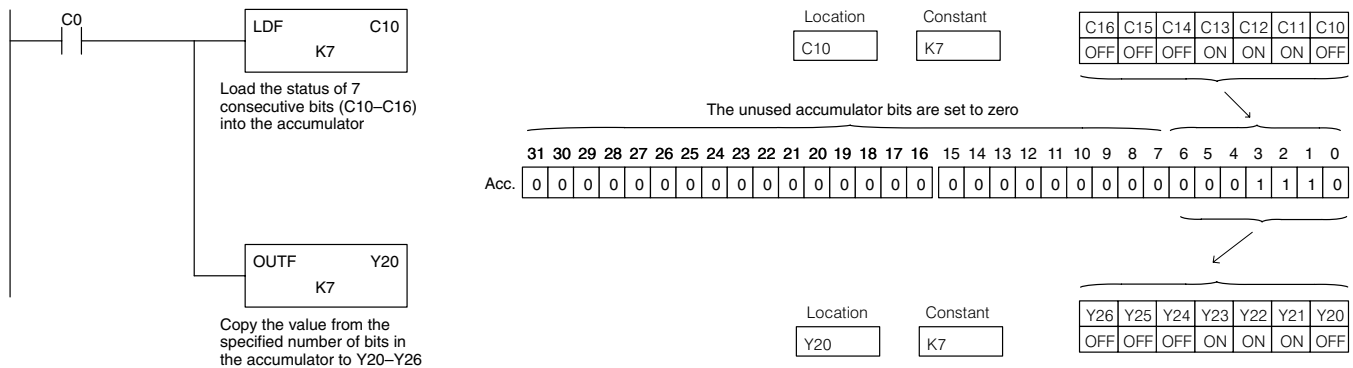
Operand Data Type	A	DL440 Range		DL450 Range	
		aaa	bbb	aaa	bbb
Inputs	X	0–477	—	0–1777	—
Outputs	Y	0–477	—	0–1777	—
Control Relays	C	0–1777	—	0–3777	—
Stage Bits	S	0–1777	—	0–1777	—
Timer Bits	T	0–377	—	0–377	—
Counter Bits	CT	0–177	—	0–377	—
Special Relays	SP	0–137 320–717	—	0–137 320–717	—
Global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

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

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

In the following example, when C0 is on, the binary pattern of C10–C16 (7 bits) will be loaded into the accumulator using the Load Formatted instruction. The lower 6 bits of the accumulator are output to Y20–Y26 using the Out Formatted instruction.

DirectSOFT32



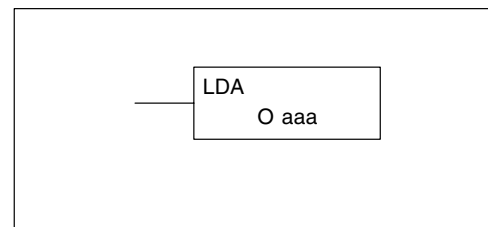
Handheld Programmer Keystrokes

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

Load Address (LDA)

✓ ✓ ✓
430 440 450

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

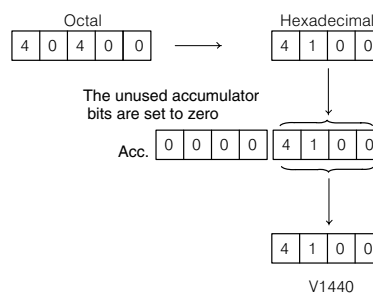
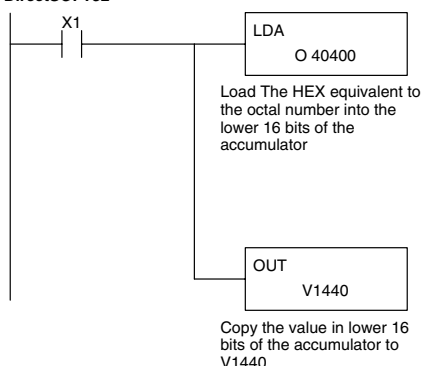
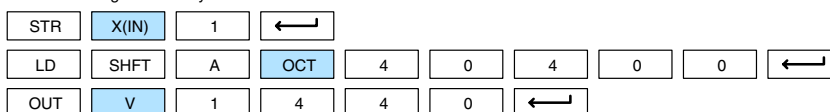


Operand Data Type	DL430 Range	DL440 Range	DL450 Range
	aaa	aaa	aaa
Octal Address O	0 – 77777	0 – 77777	0 – 177777

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

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

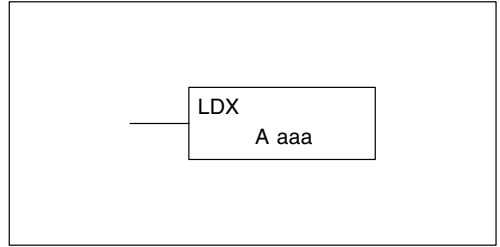
In the following example when X1 is on, the octal number 40400 will be converted to a HEX 4100 and loaded into the accumulator using the Load Address instruction. The value in the lower 16 bits of the accumulator is copied to V1440 using the Out instruction.

DirectSOFT32**Handheld Programmer Keystrokes**

Load Accumulator Indexed (LDX)

✓ ✓ ✓
430 440 450

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.



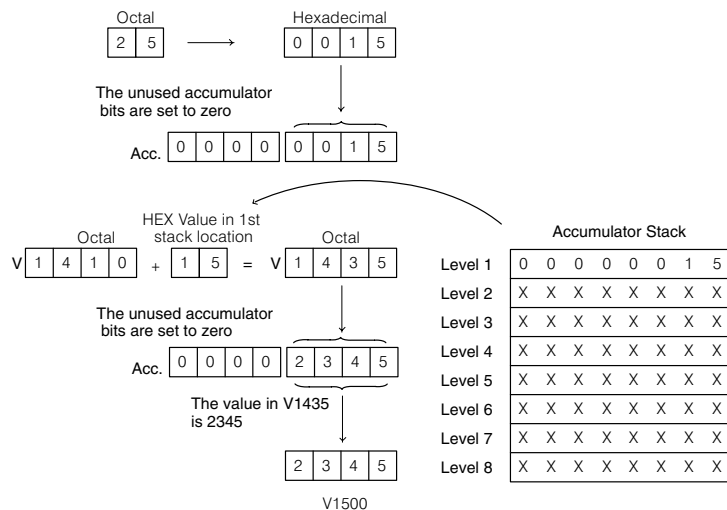
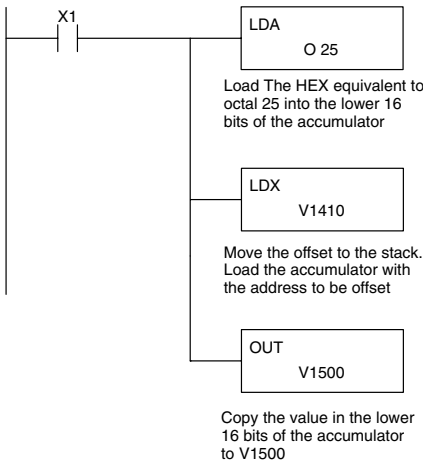
Helpful Hint: — The Load Address instruction can be used to convert an octal address to a HEX address and load the value into the accumulator.

Operand Data Type		DL430 Range	DL440 Range	DL440 Range
A		aaa	aaa	aaa
Vmemory	V	All (See p. 3-40)	All (See p. 3-41)	All (See p. 3-42)
Pointer	P	—	All (See p. 3-41)	All (See p. 3-42)

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

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

In the following example when X1 is on, the HEX equivalent for octal 25 will be loaded into the accumulator (this value will be placed on the stack when the Load Accumulator Indexed instruction is executed). V memory location V1410 will be added to the value in the 1st. level of the stack and the value in this location (V1435 = 2345) is loaded into the lower 16 bits of the accumulator using the Load Accumulator Indexed instruction. The value in the lower 16 bits of the accumulator is output to V1500 using the Out instruction.



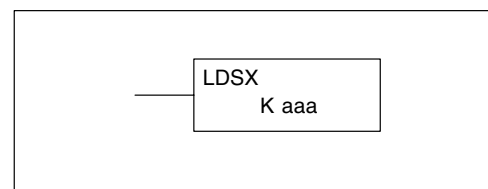
Handheld Programmer Keystrokes

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

Load Accumulator Indexed from Data Constants (LDSX)

☐ 430
 ☒ 440
 ☒ 450

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



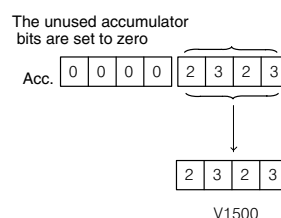
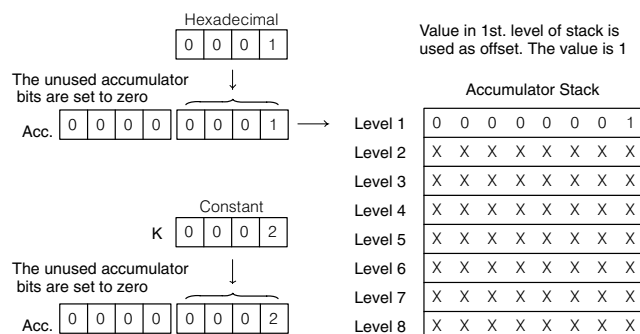
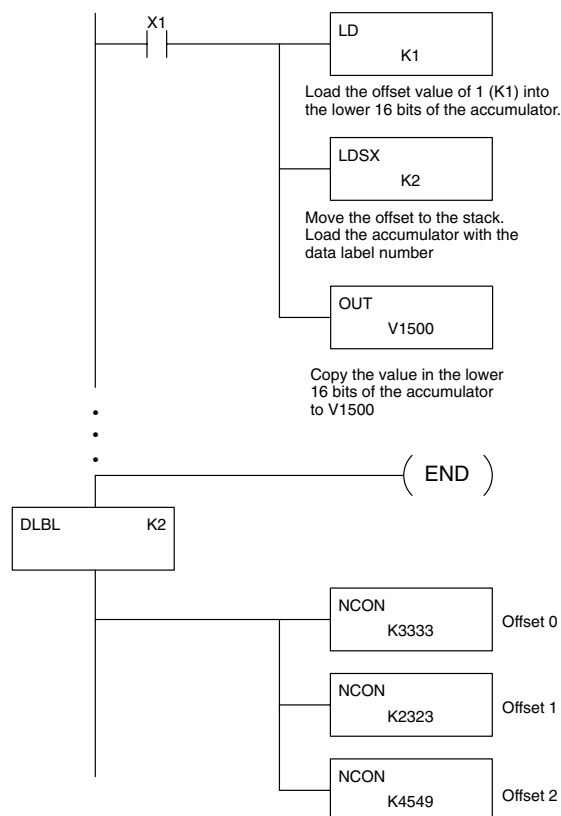
The LDSX instruction uses the value in the first level of the accumulator stack as an offset to determine which numerical or ASCII constant within the Data Label Area will be loaded into the accumulator. The LDSX instruction interprets the value in the first level of the accumulator stack as a HEX value.

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

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

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

In the following example when X1 is on, the Load instruction loads the offset of 1 (K1) into the accumulator. When the LDSX instruction executes, this value is placed into the first level of the accumulator stack. The LDSX instruction specifies the Data Label (DLBL K2) where the numerical constant(s) are located in the program. It loads the constant value according to the offset value into the accumulator's lower 16 bits.



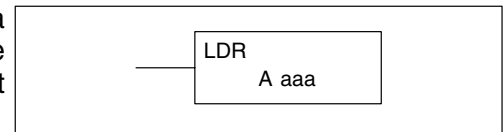
Handheld Programmer Keystrokes

STR	X(IN)	1	←								
LD	K(CON)	1	←								
LD	SHFT	S	X	K(CON)	2	←					
OUT	V	1	5	0	0	←					
END	←										
D	L	B	L	K(CON)	2	←					
SHFT	N	C	O	N	SHFT	K(CON)	2	2	2	2	←
SHFT	N	C	O	N	SHFT	K(CON)	2	3	2	3	←
SHFT	N	C	O	N	SHFT	K(CON)	4	4	4	4	←

Load Real Number (LDR)

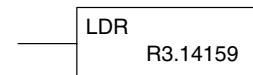
✗ ✗ ✓
430 440 450

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

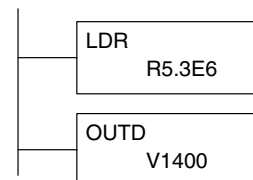


Operand Data Type	DL450 Range
A	aaa
Vmemory V	All V mem (See p. 3-42)
Pointer P	All V mem (See p. 3-42)
Real Constant R	Full IEEE 32-bit range

DirectSOFT32 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”.

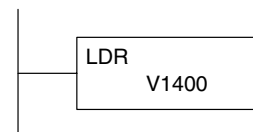


For very large numbers or very small numbers, you can use exponential notation. The number to the right is 5.3 million. The OUTD instruction stores it in V1400 and V1401.



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

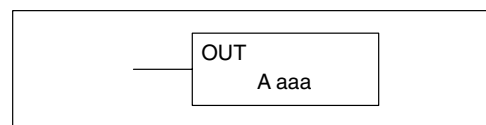
The previous example above stored a real number in V1400 and V1401. Suppose that now we want to retrieve that number. Just use the Load Real with the V data type, as shown to the right. Next we could perform real math on it, or convert it to a binary number.



**Out
(OUT)**

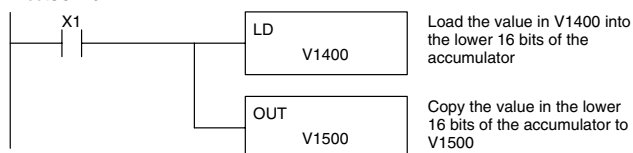
✓ ✓ ✓
430 440 450

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).

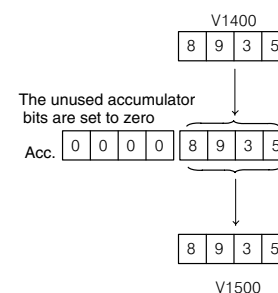


Operand Data Type		DL430 Range	DL440 Range	DL450 Range
A		aaa	aaa	aaa
Vmemory	V	All (See p. 3-40)	All (See p. 3-41)	All (See p. 3-42)
Pointer	P	All (See p. 3-40)	All (See p. 3-41)	All (See p. 3-42)

In the following example, when X1 is on, the value in V1400 will be loaded into the lower 16 bits of the accumulator using the Load instruction. The value in the lower 16 bits of the accumulator are copied to V1500 using the Out instruction.

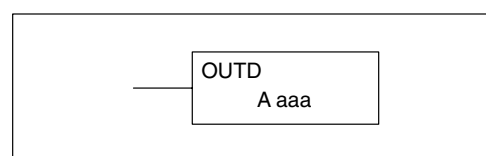
DirectSOFT32**Handheld Programmer Keystrokes**

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

**Out DOUBLE
(OUTD)**

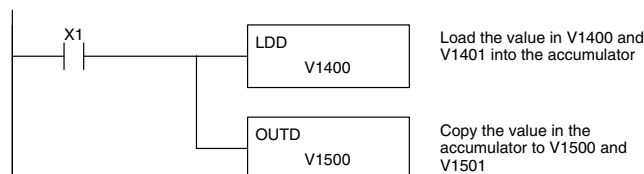
✓ ✓ ✓
430 440 450

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).

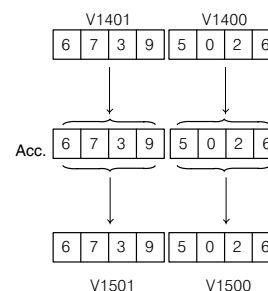


Operand Data Type		DL430 Range	DL440 Range	DL450 Range
A		aaa	aaa	aaa
Vmemory	V	All (See p. 3-40)	All (See p. 3-41)	All (See p. 3-42)
Pointer	P	All V mem (See p. 3-40)	All V mem (See p. 3-41)	All V mem (See p. 3-42)

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

DirectSOFT32**Handheld Programmer Keystrokes**

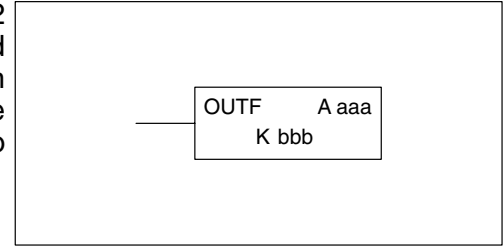
STR	X(IN)	1	←
LD	SHFT	D	SHFT V 1 4 0 0 ←
OUT	SHFT	D	SHFT V 1 5 0 0 ←



Out Formatted (OUTF)

430 440 450

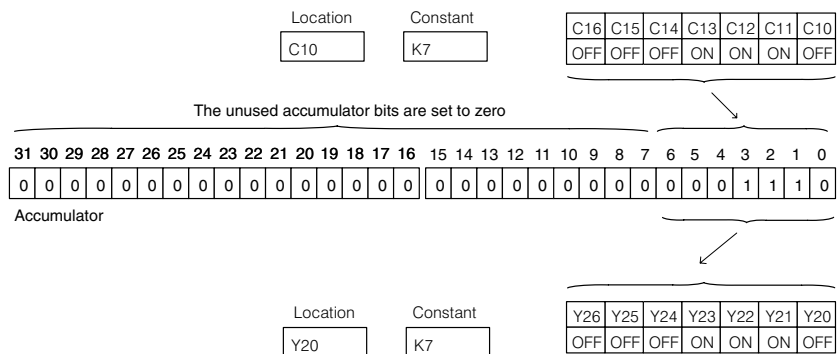
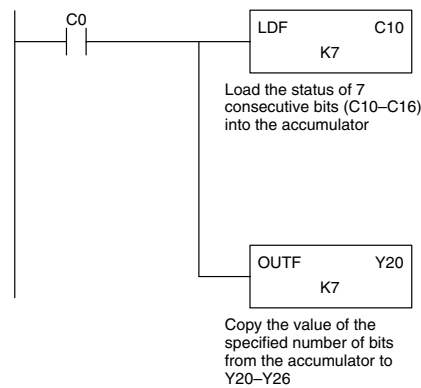
The Out Formatted instruction outputs 1–32 bits from the accumulator to the specified discrete memory locations. The instruction requires a starting location (Aaaa) for the destination and the number of bits (Kbbb) to be output.



Operand Data Type		DL440 Range		DL450 Range	
	A	aaa	bbb	aaa	bbb
Inputs	X	0–477	—	0–1777	—
Outputs	Y	0–477	—	0–1777	—
Control Relays	C	0–1777	—	0–3777	—
Global I/O	GX	0–1777	—	0–2777	—
Constant	K	—	1–32	—	1–32

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

DirectSOFT32



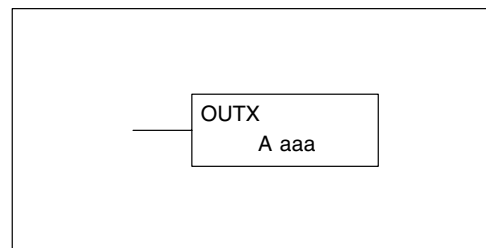
Handheld Programmer Keystrokes

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

Out Indexed (OUTX)

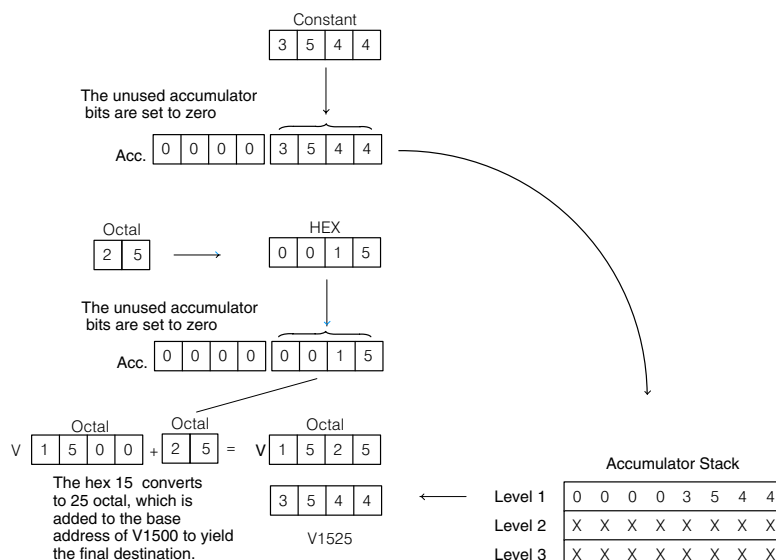
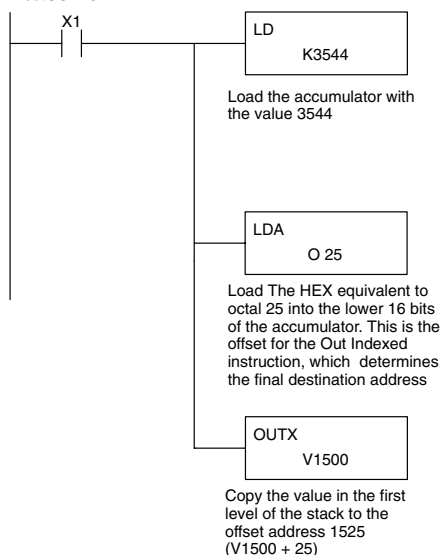
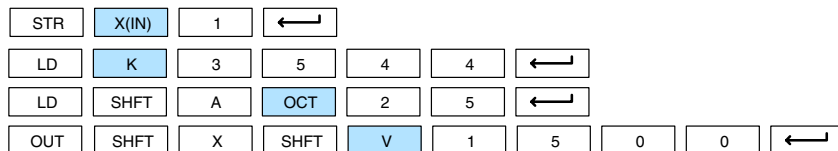
✓ ✓ ✓
430 440 450

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		DL430 Range	DL440 Range	DL450 Range
A		aaa	aaa	aaa
Vmemory	V	All (See p. 3-40)	All (See p. 3-41)	All (See p. 3-42)
Pointer	P	All (See p. 3-40)	All (See p. 3-41)	All (See p. 3-42)

In the following example, when X1 is on, the constant value 3544 is loaded into the accumulator. This is the value that will be output to the specified offset V memory location (V1525). The value 3544 will be placed onto the stack when the Load Address instruction is executed. Remember, two consecutive Load instructions places the value of the first load instruction onto the stack. The Load Address instruction converts octal 25 to HEX 15 and places the value in the accumulator. The Out Indexed instruction outputs the value 3544 which resides in the first level of the accumulator stack to V1525.

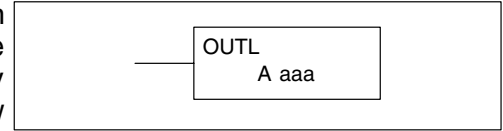
DirectSOFT32**Handheld Programmer Keystrokes**

	Accumulator Stack
Level 1	0 0 0 0 3 5 4 4
Level 2	X X X X X X X X
Level 3	X X X X X X X X
Level 4	X X X X X X X X
Level 5	X X X X X X X X
Level 6	X X X X X X X X
Level 7	X X X X X X X X
Level 8	X X X X X X X X

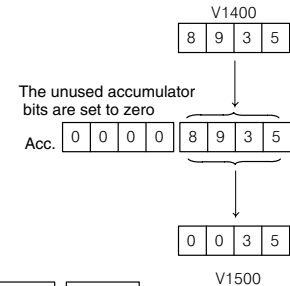
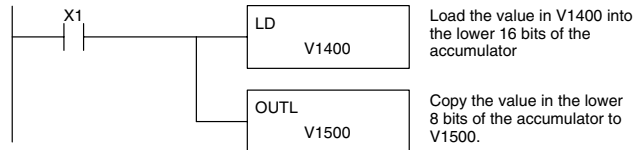
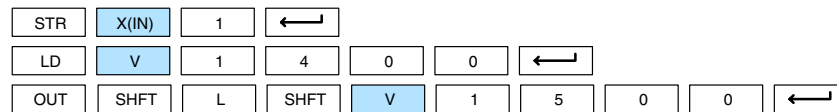
**Out Least
(OUTL)**

☒ 430
 ☒ 440
 ☒ 450

The Out Least instruction copies the value in the lower eight bits of the accumulator to the lower eight bits of the specified V-memory location (i.e., it copies the low byte of the low word of the accumulator).

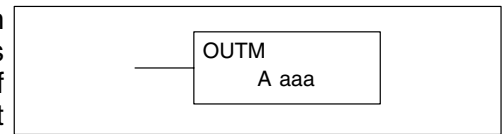


In the following example, when X1 is on, the value in V1400 will be loaded into the lower 16 bits of the accumulator using the Load instruction. The value in the lower 8 bits of the accumulator are copied to V1500 using the Out Least instruction.

DirectSOFT32**Handheld Programmer Keystrokes****Out Most
(OUTM)**

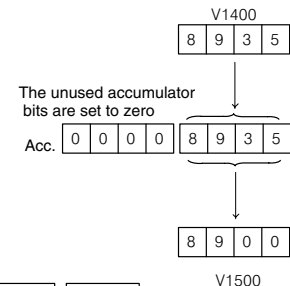
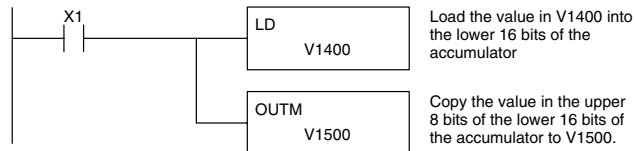
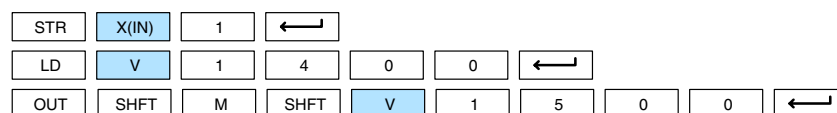
☒ 430
 ☒ 440
 ☒ 450

The Out Most instruction copies the value in the upper eight bits of the lower sixteen bits of the accumulator to the upper eight bits of the specified V-memory location (i.e., it copies the high byte of the low word of the accumulator).



Operand Data Type		DL450 Range
A		aaa
Vmemory	V	All (See p. 3-42)
Pointer	P	All (See p. 3-42)

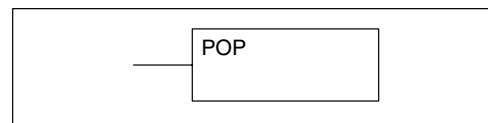
In the following example, when X1 is on, the value in V1400 will be loaded into the lower 16 bits of the accumulator using the Load instruction. The value in the upper 8 bits of the lower 16 bits of the accumulator are copied to V1500 using the Out Most instruction.

DirectSOFT32**Handheld Programmer Keystrokes**

**Pop
(POP)**

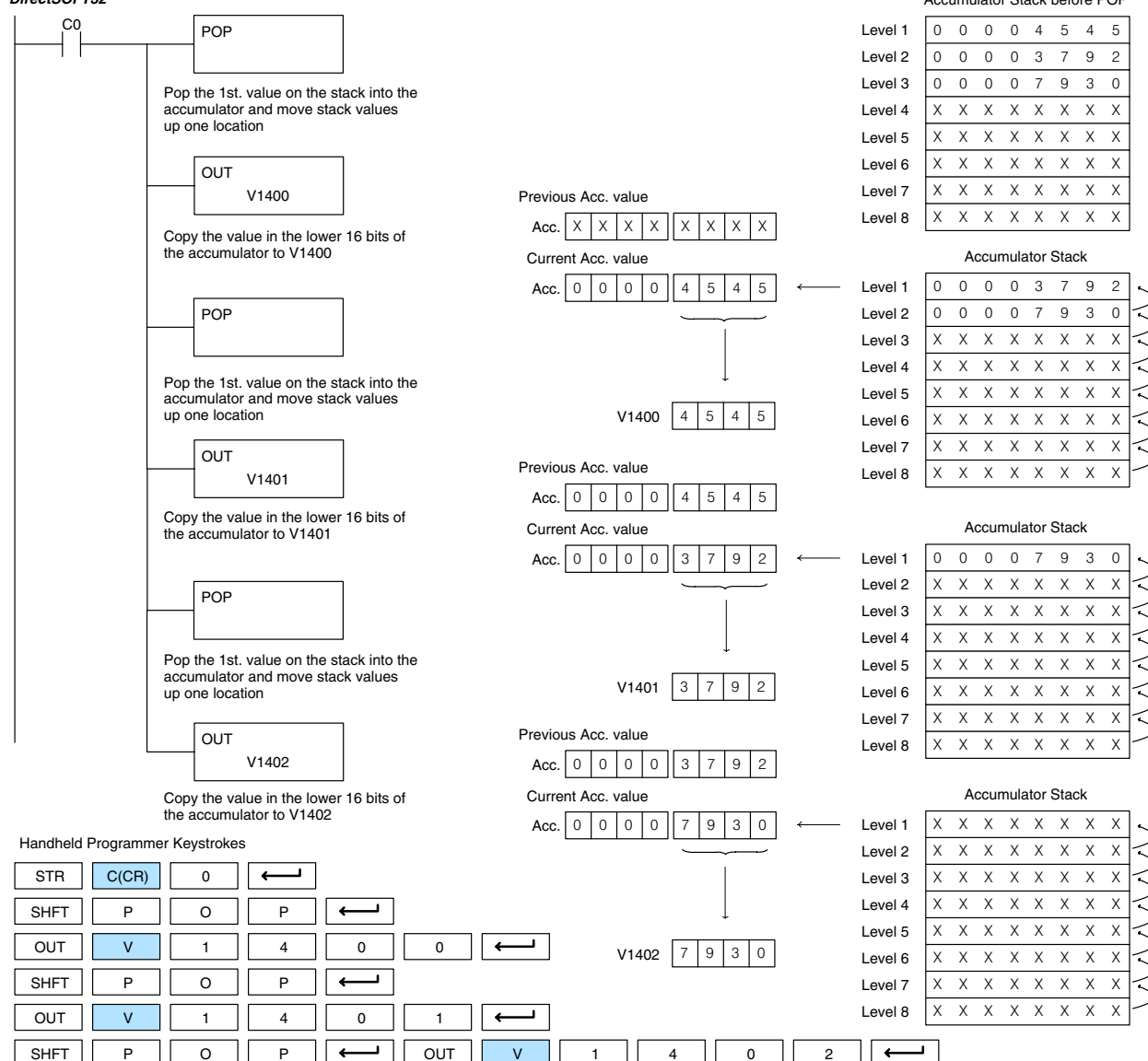
✓ ✓ ✓
430 440 450

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

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

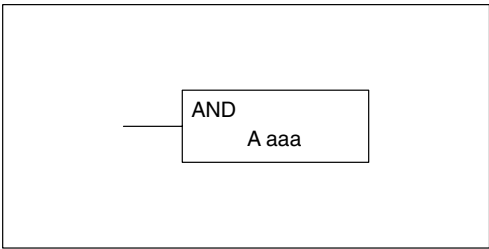
DirectSOFT32

Accumulator Logic Instructions

And (AND)

✓ ✓ ✓
430 440 450

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

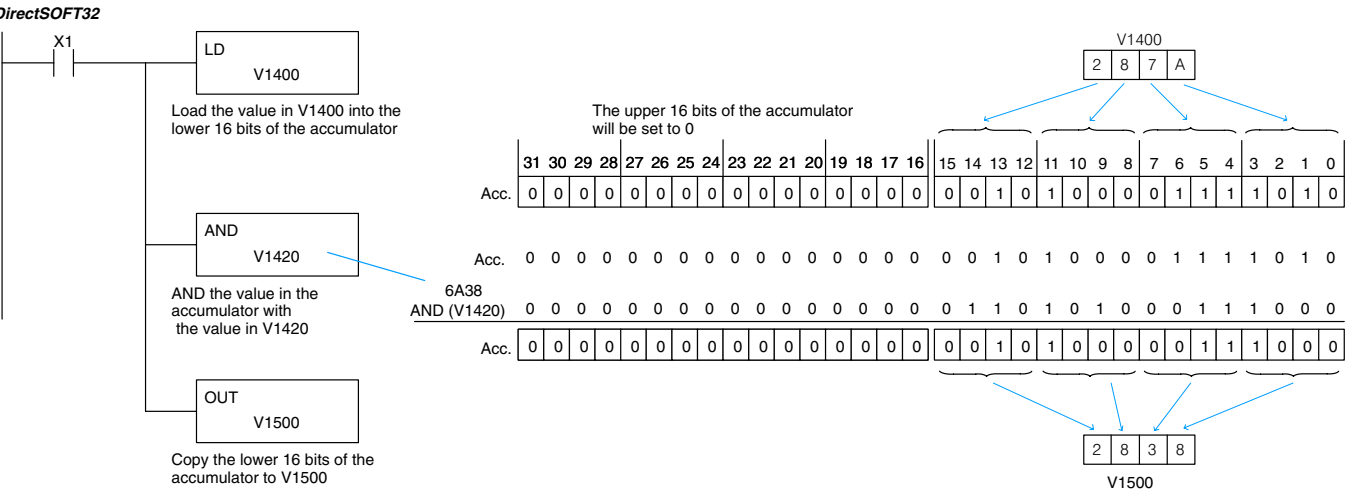


Operand Data Type		DL430 Range	DL440 Range	DL450 Range
A		aaa	aaa	aaa
Vmemory	V	All (See p. 3-40)	All (See p. 3-41)	All (See p. 3-42)
Pointer	P	—	All (See p. 3-41)	All (See p. 3-42)

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

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

In the following example, when X1 is on, the value in V1400 will be loaded into the accumulator using the Load instruction. The value in the accumulator is anded with the value in V1420 using the And instruction. The value in the lower 16 bits of the accumulator are output to V1500 using the Out instruction.



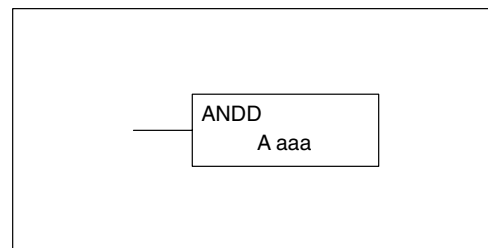
Handheld Programmer Keystrokes

STR	X(IN)	1	←
LD	V	1	4 0 0 ←
AND	V	1	4 2 0 ←
OUT	V	1	5 0 0 ←

**And Double
(ANDD)**


430 440 450

The And Double is a 32 bit instruction that logically ands the value in the accumulator with two consecutive V memory locations or an 8 digit (max.) constant value (Aaaa). The result resides in the accumulator. Discrete status flags indicate if the result of the And Double is zero or a negative number (the most significant bit is on).



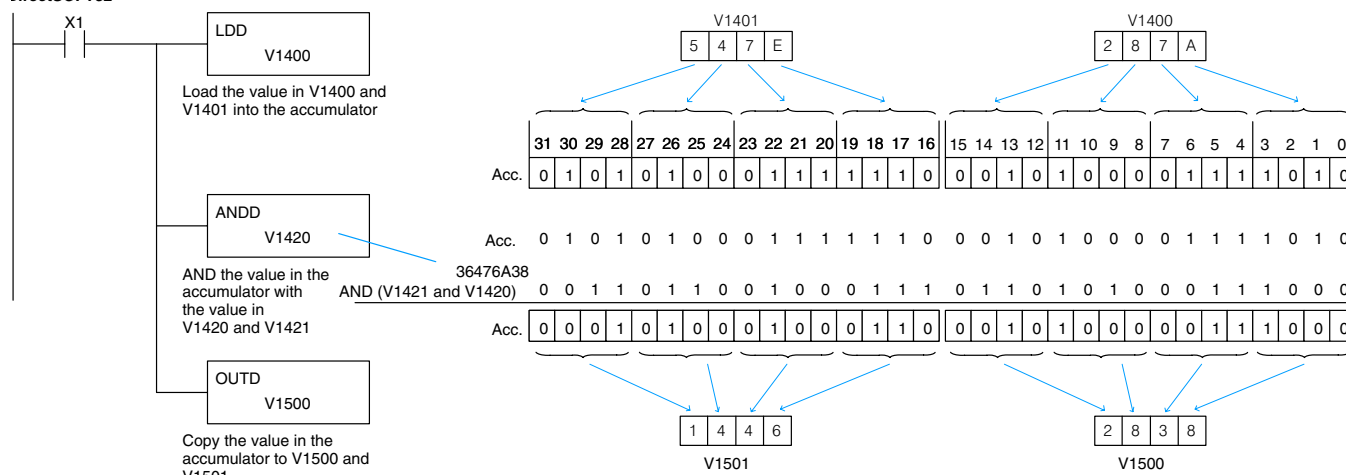
Operand Data Type		DL430 Range	DL440 Range	DL440 Range
A		aaa	aaa	aaa
Vmemory	V	—	All (See p. 3-41)	All (See p. 3-42)
Pointer	P	—	All (See p. 3-41)	All (See p. 3-42)
Constant	K	0-FFFF	0-FFFF	0-FFFF

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

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

In the following example, when X1 is on, the value in V1400 and V1401 will be loaded into the accumulator using the Load Double instruction. The value in the accumulator is anded with V1420 and V1421 using the And double instruction. The value in the accumulator is output to V1500 and V1501 using the Out Double instruction.

DirectSOFT32



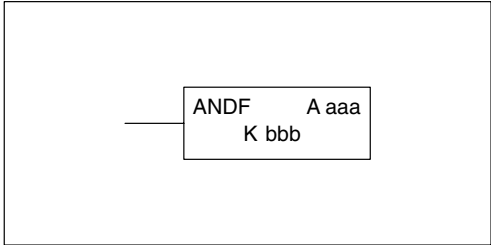
Handheld Programmer Keystrokes

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

And
Formatted
(ANDF)

✕ ✓ ✓
430 440 450

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).

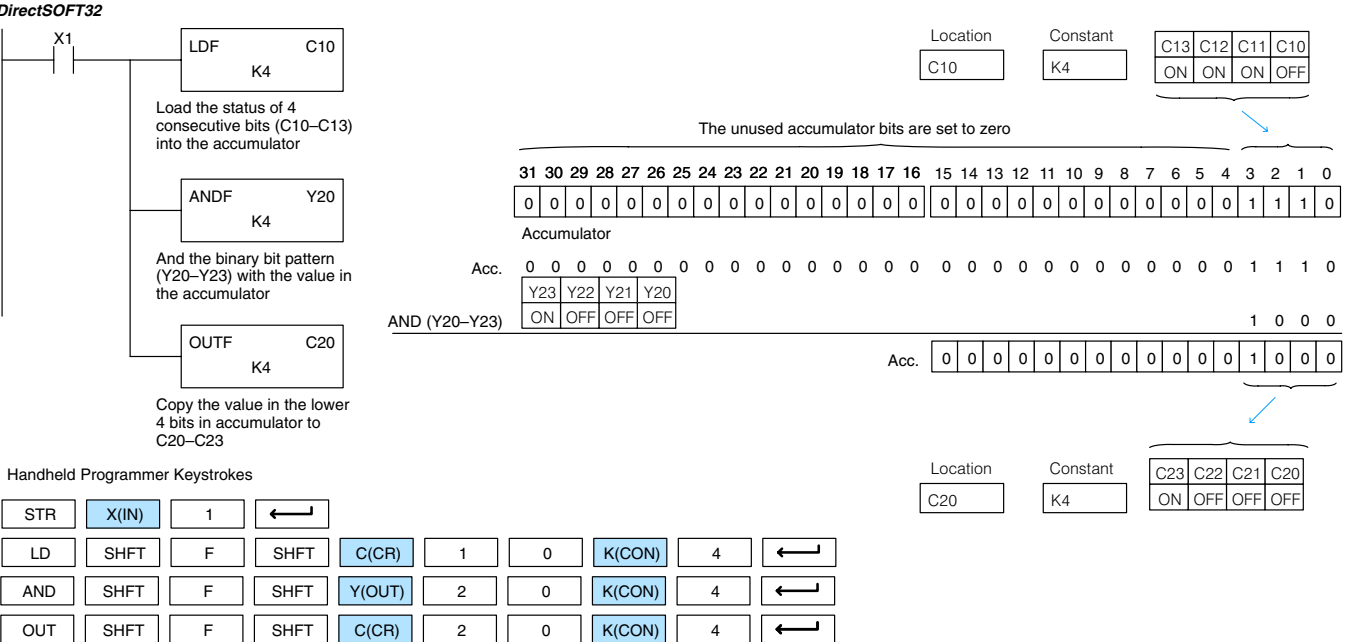


Operand Data Type		DL440 Range		DL450 Range	
	A/B	aaa	bbb	aaa	bbb
Inputs	X	0–477	—	0–1777	—
Outputs	Y	0–477	—	0–1777	—
Control Relays	C	0–1777	—	0–3777	—
Stage Bits	S	0–1777	—	0–1777	—
Timer Bits	T	0–377	—	0–377	—
Counter Bits	CT	0–177	—	0–377	—
Special Relays	SP	0–137 320–717	—	0–137 320–717	—
Global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

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

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

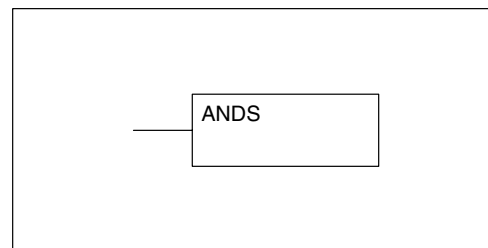
In the following example, when X1 is on the Load Formatted instruction loads C10–C13 (4 binary bits) into the accumulator. The accumulator contents is logically ANDed with the bit pattern from Y20–Y23 using the And Formatted instruction. The Out Formatted instruction outputs the accumulator's lower four bits to C20–C23.



**And with Stack
(ANDS)**

×	✓	✓
430	440	450

The And with Stack instruction is a 32 bit instruction that logically ands the value in the accumulator with the first level of the accumulator stack. The result resides in the accumulator. The value in the first level of the accumulator stack is removed from the stack and all values are moved up one level. Discrete status flags indicate if the result of the And with Stack is zero or a negative number (the most significant bit is on).

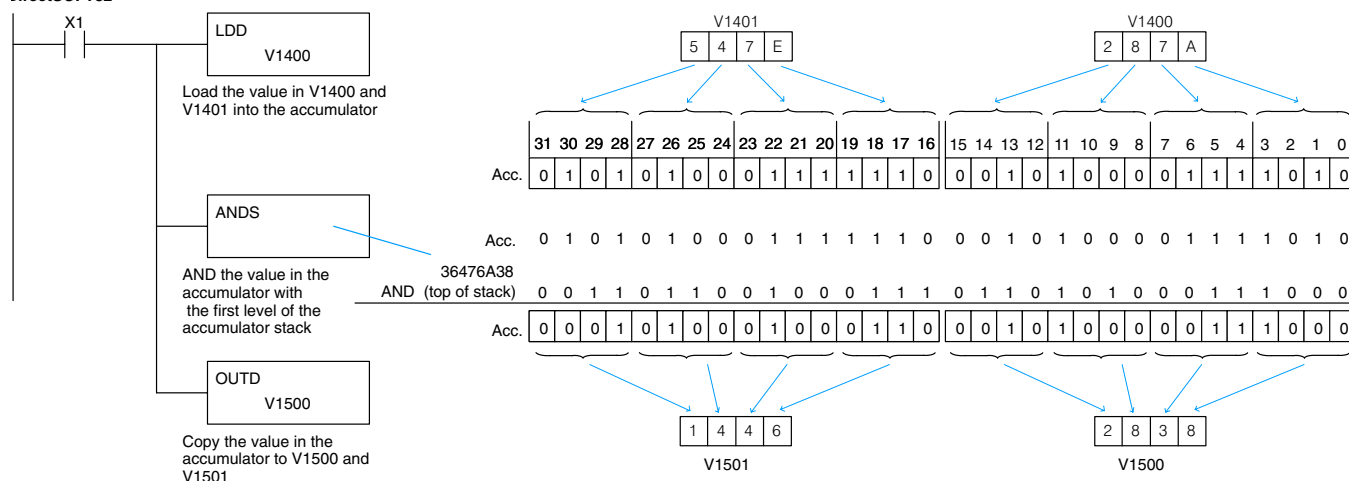


Discrete Bit Flags	Description
SP63	Will be on if the result in the accumulator is zero
SP70	Will be on is the result in the accumulator is negative

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

In the following example when X1 is on, the binary value in the accumulator will be anded with the binary value in the first level of the accumulator stack. The result resides in the accumulator. The 32 bit value is then output to V1500 and V1501.

DirectSOFT32



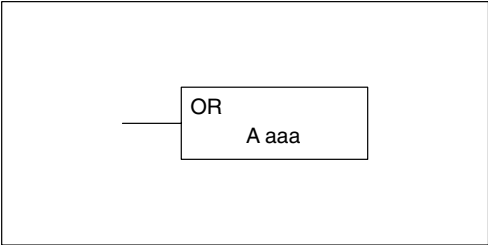
Handheld Programmer Keystrokes

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

Or
(OR)

✓ ✓ ✓
430 440 450

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



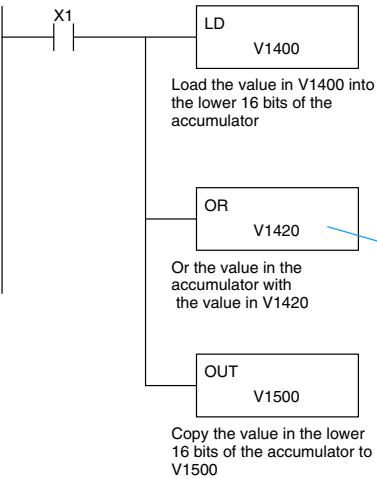
Operand Data Type		DL430 Range	DL440 Range	DL440 Range
A		aaa	aaa	aaa
Vmemory	V	All (See p. 3-40)	All (See p. 3-41)	All (See p. 3-42)
Pointer	P	—	All V mem (See p. 3-41)	All V mem (See p. 3-42)

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

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

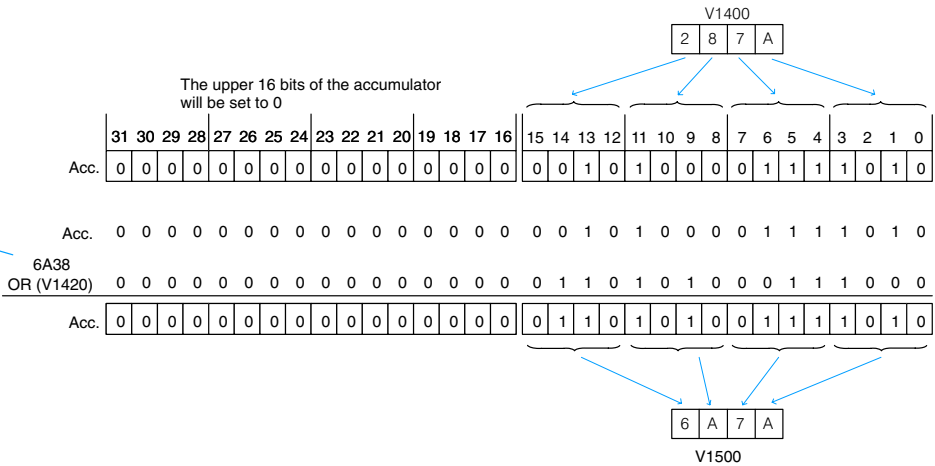
In the following example, when X1 is on, the value in V1400 will be loaded into the accumulator using the Load instruction. The value in the accumulator is ored with V1420 using the Or instruction. The value in the lower 16 bits of the accumulator are output to V1500 using the Out instruction.

DirectSOFT32



Handheld Programmer Keystrokes

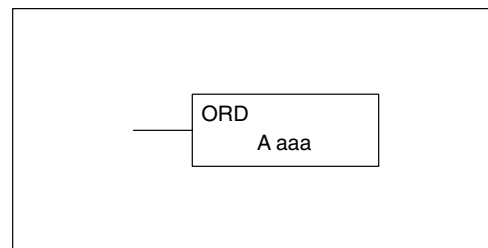
STR	X(IN)	1	←
LD	V	1	4 0 0 ←
OR	V	1	4 2 0 ←
OUT	V	1	5 0 0 ←



**Or Double
(ORD)**

✓ ✓ ✓
430 440 450

The OR Double is a 32 bit instruction that ORs the value in the accumulator with the value (Aaaa), which is either two consecutive V-memory locations or an 8 digit (max.) constant value. The result resides in the accumulator. Discrete status flags indicate if the result of the Or Double is zero or a negative number (the most significant bit is on).



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

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

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

In the following example, when X1 is on, the value in V1400 and V1401 will be loaded into the accumulator using the Load Double instruction. The value in the accumulator is ored with V1420 and V1421 using the Or Double instruction. The value in the accumulator is output to V1500 and V1501 using the Out Double instruction.

DirectSOFT32



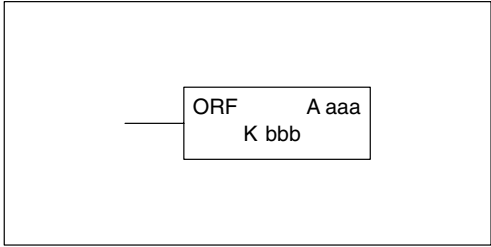
Handheld Programmer Keystrokes

STR	X(IN)	1	←
LD	SHFT	D	SHFT V 1 4 0 0 ←
OR	SHFT	D	SHFT V 1 4 2 0 ←
OUT	SHFT	D	SHFT V 1 5 0 0 ←

Or
Formatted
(ORF)

430 440 450

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).

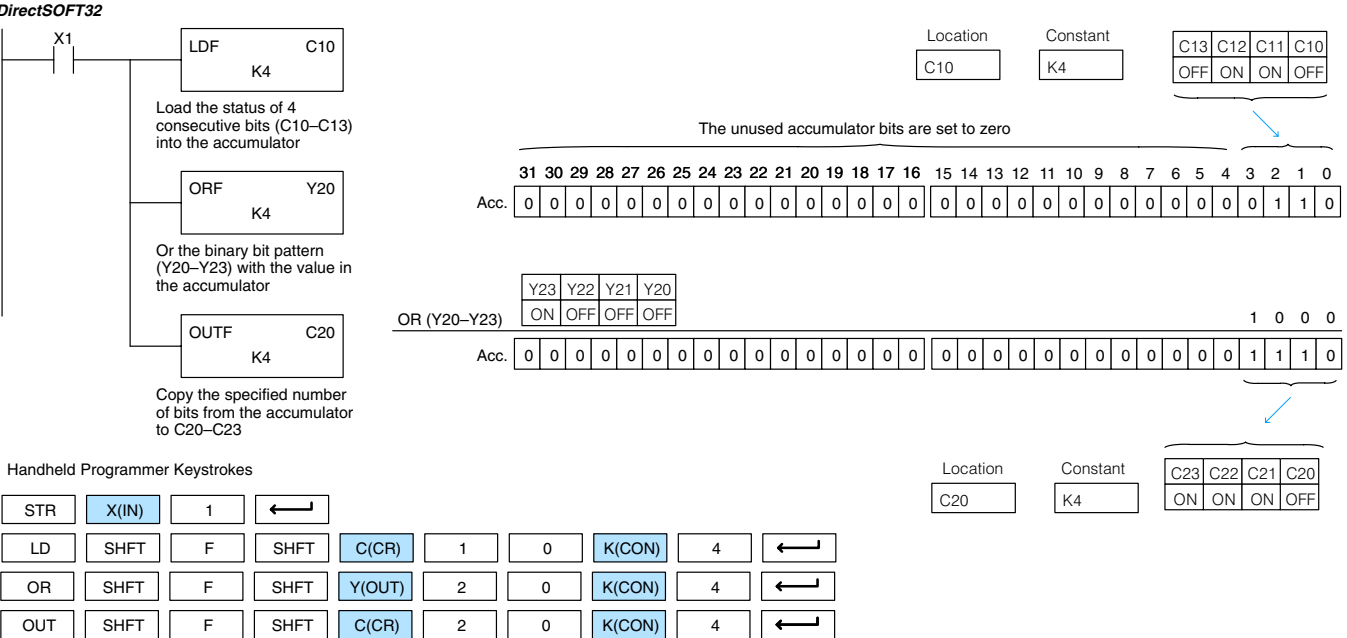


Operand Data Type	A/B	DL440 Range		DL450 Range	
		aaa	bbb	aaa	bbb
Inputs	X	0–477	—	0–1777	—
Outputs	Y	0–477	—	0–1777	—
Control Relays	C	0–1777	—	0–3777	—
Stage Bits	S	0–1777	—	0–1777	—
Timer Bits	T	0–377	—	0–377	—
Counter Bits	CT	0–177	—	0–377	—
Special Relays	SP	0–137 320–717	—	0–137 320–717	—
Global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

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

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

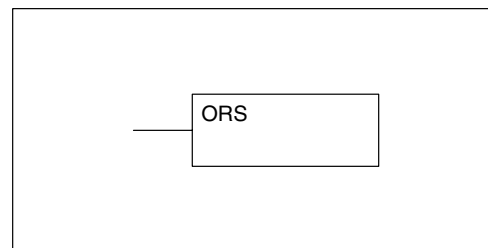
In the following example, when X1 is on the Load Formatted instruction loads C10–C13 (4 binary bits) into the accumulator. The Or Formatted instruction logically ORs the accumulator contents with Y20–Y23 bit pattern. The Out Formatted instruction outputs the accumulator's lower four bits to C20–C23.



**Or with Stack
(ORS)**

✗	✓	✓
430	440	450

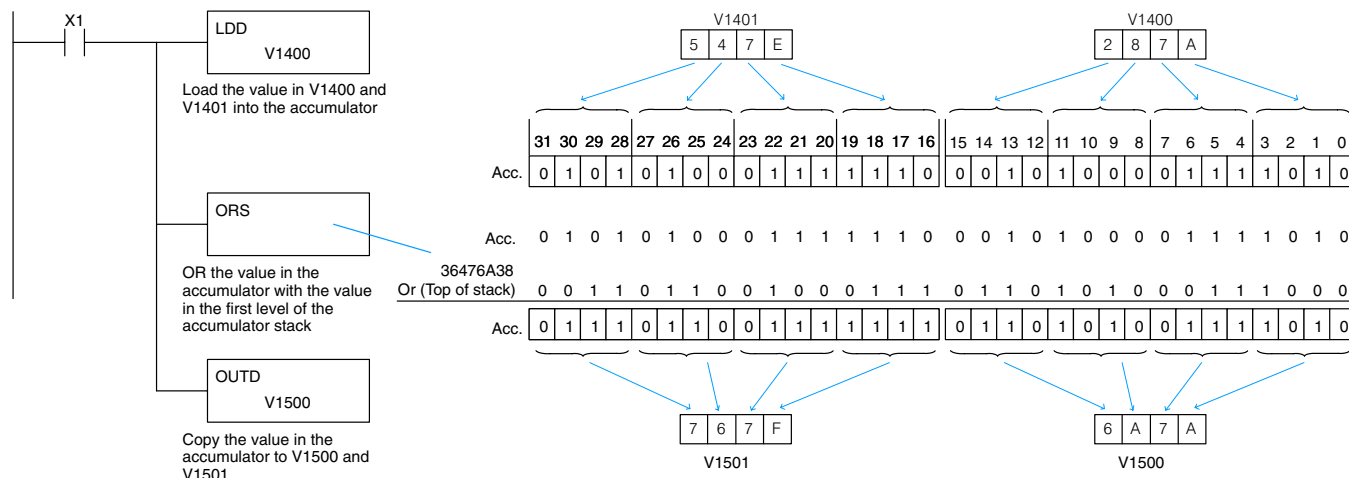
The Or with Stack instruction is a 32 bit instruction that logically ors the value in the accumulator with the first level of the accumulator stack. The result resides in the accumulator. The value in the first level of the accumulator stack is removed from the stack and all values are moved up one level. Discrete status flags indicate if the result of the Or with Stack is zero or a negative number (the most significant bit is on).



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

In the following example when X1 is on, the binary value in the accumulator will be ored with the binary value in the first level of the stack. The result resides in the accumulator.

DirectSOFT32



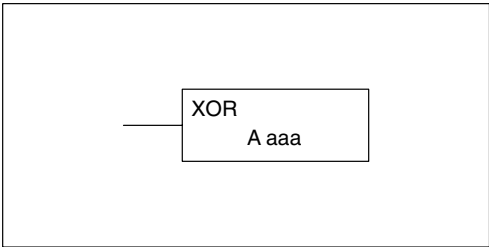
Handheld Programmer Keystrokes

STR	X(IN)	1	←							
LD	SHFT	D	SHFT	V	1	4	0	0	←	
OR	SHFT	S	←							
OUT	SHFT	D	SHFT	V	1	5	0	0	←	

Exclusive Or (XOR)

✓✓✓
430 440 450

The Exclusive Or instruction is a 16 bit instruction that performs an exclusive or of the value in the lower 16 bits of the accumulator and a specified V memory location (Aaaa). The discrete status flag indicates if the result of the Xor is zero.



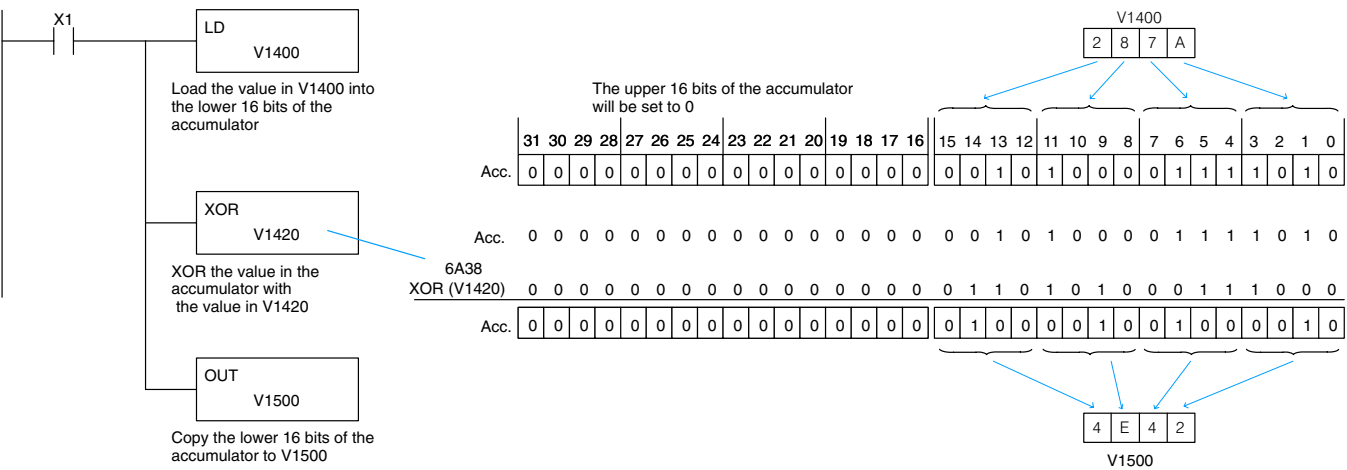
Operand Data Type		DL430 Range	DL440 Range	DL440 Range
A		aaa	aaa	aaa
Vmemory	V	All (See p. 3-40)	All (See p. 3-41)	All (See p. 3-42)
Pointer	P	—	All V mem (See p. 3-41)	All V mem (See p. 3-42)

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

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

In the following example, when X1 is on, the value in V1400 will be loaded into the accumulator using the Load instruction. The value in the accumulator is exclusive orred with V1420 using the Exclusive Or instruction. The value in the lower 16 bits of the accumulator are output to V1500 using the Out instruction.

DirectSOFT32



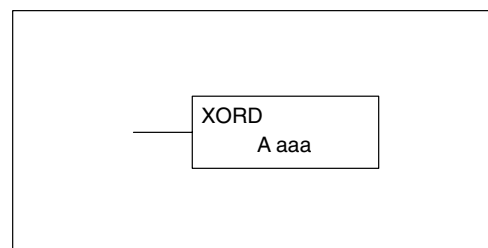
Handheld Programmer Keystrokes

STR	X(IN)	1	←
LD	V	1	4 0 0 ←
SHFT	X	SHFT	OR V 1 4 2 0 ←
OUT	V	1	5 0 0 ←

**Exclusive Or
Double
(XORD)**

✓ ✓ ✓
430 440 450

The Exclusive OR Double is a 32 bit instruction that performs an exclusive or of the value in the accumulator and the value (Aaaa), which is either two consecutive V memory locations or an 8 digit (max.) constant. The result resides in the accumulator. Discrete status flags indicate if the result of the Exclusive Or Double is zero or a negative number (the most significant bit is on).

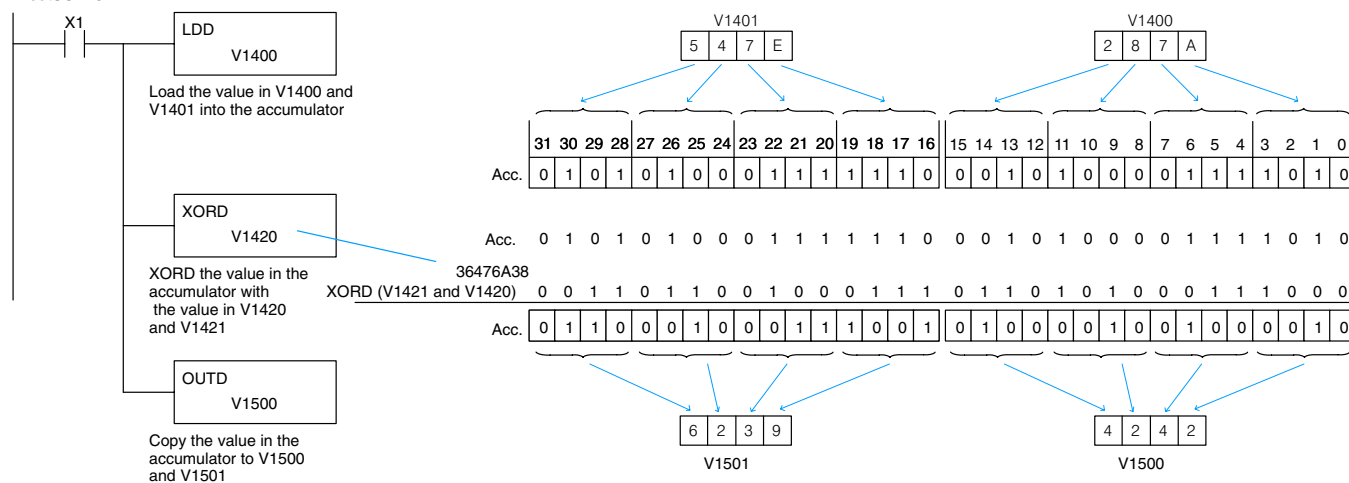


Operand Data Type		DL430 Range	DL440 Range	DL450 Range
A		aaa	aaa	aaa
Vmemory	V	—	All (See p. 3-41)	All (See p. 3-42)
Pointer	P	—	All (See p. 3-41)	All (See p. 3-42)
Constant	K	0-FFFF	0-FFFF	0-FFFF

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

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

In the following example, when X1 is on, the value in V1400 and V1401 will be loaded into the accumulator using the Load Double instruction. The value in the accumulator is exclusively ored with V1420 and V1421 using the Exclusive Or Double instruction. The value in the accumulator is output to V1500 and V1501 using the Out Double instruction.

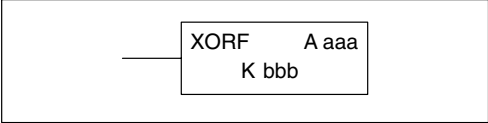
DirectSOFT32**Handheld Programmer Keystrokes**

STR	X(IN)	1	←
LD	SHFT	D	SHFT V 1 4 0 0 ←
SHFT	X	SHFT OR	SHFT D SHFT V 1 4 2 0 ←
OUT	SHFT D	SHFT V	1 5 0 0 ←

Exclusive Or
Formatted
(XORF)

430 440 450

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).



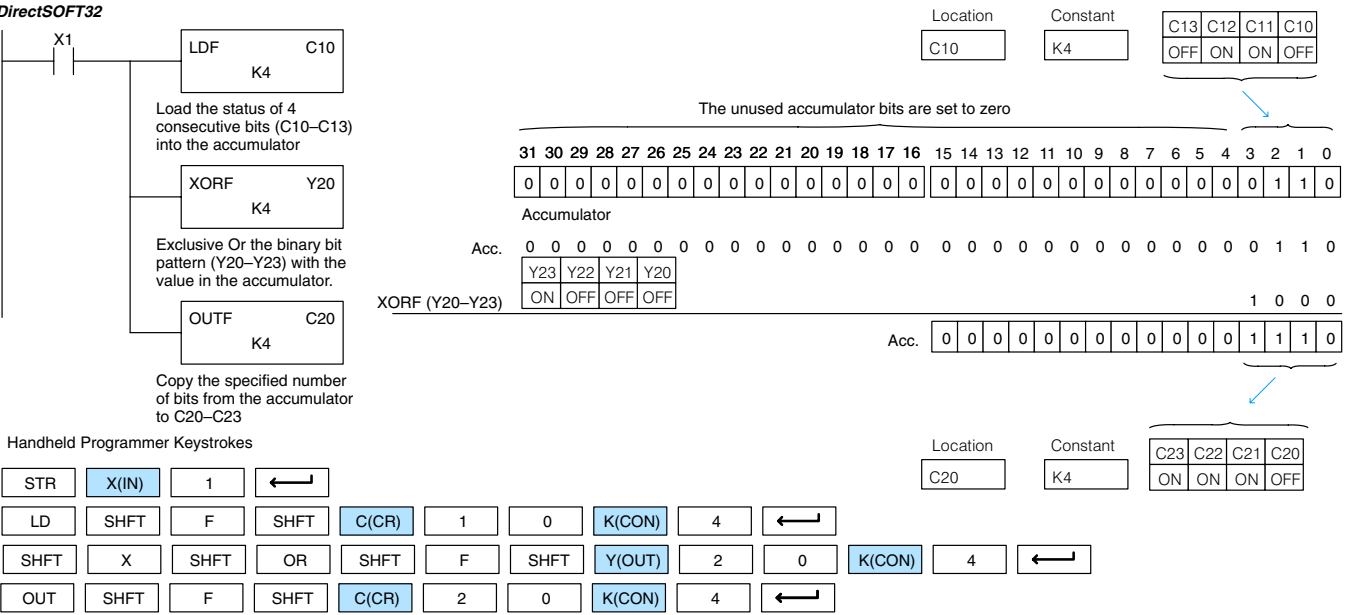
The instruction requires a starting location (Aaaa) and the number of bits (Bbbb) to be exclusive ORed. Discrete status flags indicate if the result of the Exclusive Or Formatted is zero or negative (the most significant bit =1).

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

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

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

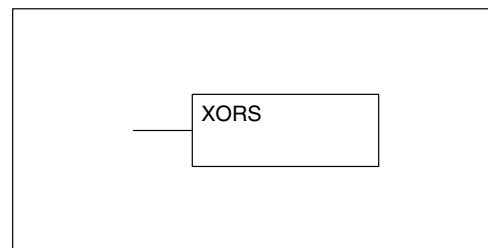
In the following example, when X1 is on, the binary pattern of C10–C13 (4 bits) will be loaded into the accumulator using the Load Formatted instruction. The value in the accumulator will be logically Exclusive Ored with the bit pattern from Y20–Y23 using the Exclusive Or Formatted instruction. The value in the lower 4 bits of the accumulator are output to C20–C23 using the Out Formatted instruction.



Exclusive Or with Stack (XORS)

X	✓	✓
430	440	450

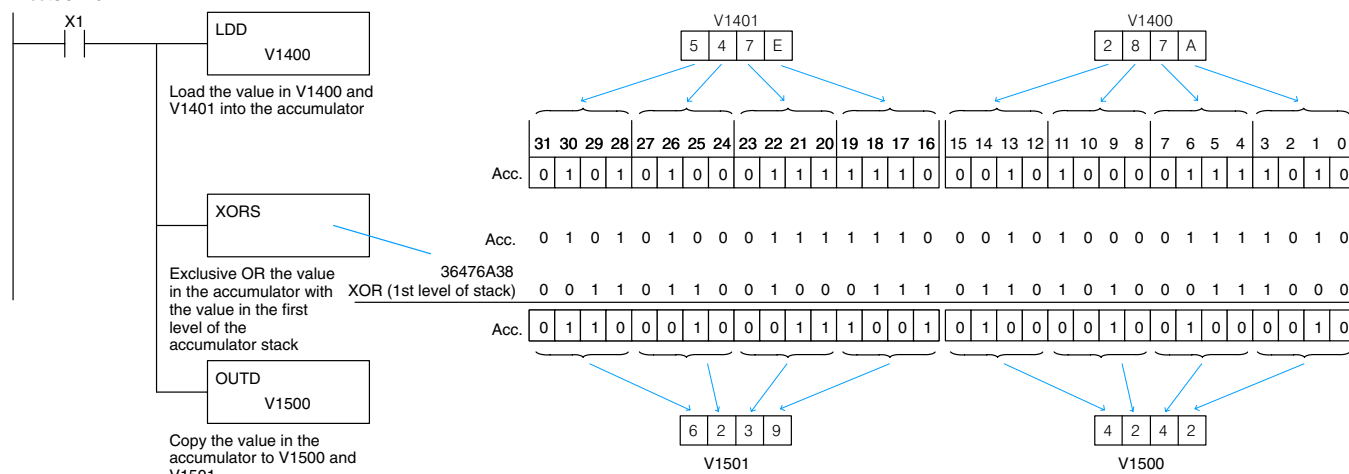
The Exclusive Or with Stack instruction is a 32 bit instruction that performs an exclusive or of the value in the accumulator with the first level of the accumulator stack. The result resides in the accumulator. The value in the first level of the accumulator stack is removed from the stack and all values are moved up one level. Discrete status flags indicate if the result of the Exclusive Or with Stack is zero or a negative number (the most significant bit is on).



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

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

In the following example when X1 is on, the binary value in the accumulator will be exclusive ored with the binary value in the first level of the accumulator stack. The result will reside in the accumulator.

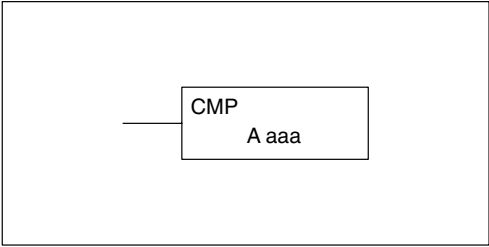
DirectSOFT32**Handheld Programmer Keystrokes**

STR	X(IN)	1	←
LD	SHFT	D	SHFT V 1 4 0 0 ←
SHFT	X	SHFT	OR SHFT S ←
OUT	SHFT	D	SHFT V 1 5 0 0 ←

Compare (CMP)

✓ ✓ ✓
430 440 450

The compare instruction is a 16 bit instruction that compares the value in the lower 16 bits of the accumulator with the value in a specified V memory location (Aaaa). The corresponding status flag will be turned on indicating the result of the comparison. You can compare either binary or BCD numbers, as long as both numbers are of the same data type.

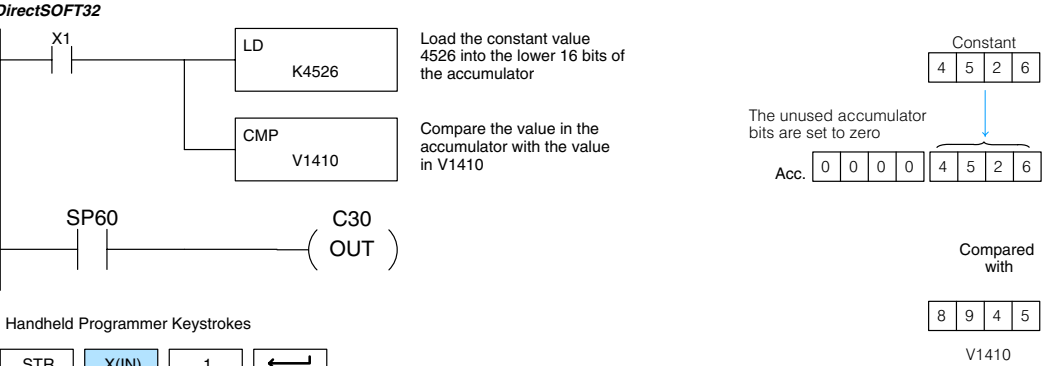


Operand Data Type		DL430 Range	DL440 Range	DL450 Range
A		aaa	aaa	aaa
Vmemory	V	All (See p. 3-40)	All (See p. 3-41)	All (See p. 3-42)
Pointer	P	—	All (See p. 3-41)	All (See p. 3-42)

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

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

In the following example when X1 is on, the constant 4526 will be loaded into the lower 16 bits of the accumulator using the Load instruction. The value in the accumulator is compared with the value in V1410 using the Compare instruction. The corresponding discrete status flag will be turned on indicating the result of the comparison. In this example, if the value in the accumulator is less than the value specified in the Compare instruction, SP60 will turn on energizing C30.



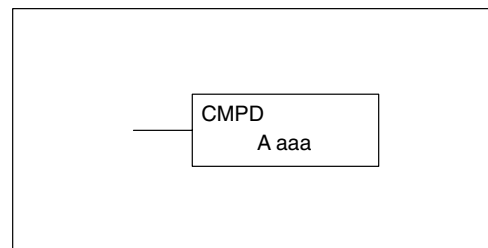
Handheld Programmer Keystrokes

STR	X(IN)	1	←				
LD	K(CON)	4	5	2	6	←	
CMP	V	1	4	1	0	←	
STR	SPCL	6	0	←			
OUT	C(CR)	3	0	←			

**Compare Double
(CMPD)**

✓ ✓ ✓
430 440 450

The Compare Double instruction is a 32-bit instruction that compares the value in the accumulator with the value (Aaaa), which is either two consecutive V memory locations or an 8-digit (max.) constant. The corresponding status flag will be turned on indicating the result of the comparison. You can compare either binary or BCD numbers, as long as both numbers are of the same data type.

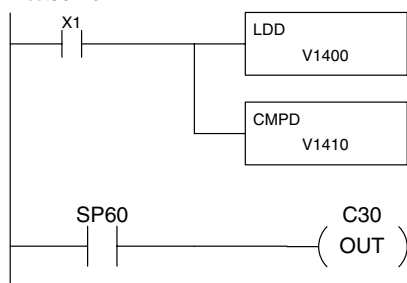


Operand Data Type		DL430 Range	DL440 Range	DL450 Range
A		aaa	aaa	aaa
Vmemory	V	—	All (See p. 3-41)	All (See p. 3-42)
Pointer	P	—	All (See p. 3-41)	All (See p. 3-42)
Constant	K	0-FFFFFFFF	0-FFFFFFFF	0-FFFFFFFF

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

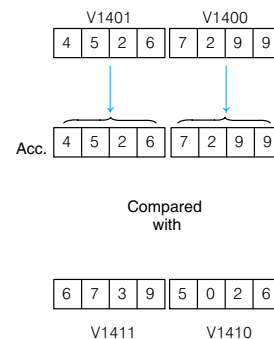
NOTE: Status flags are valid only until another instruction uses the same flag.

In the following example when X1 is on, the value in V1400 and V1401 will be loaded into the accumulator using the Load Double instruction. The value in the accumulator is compared with the value in V1410 and V1411 using the CMPD instruction. The corresponding discrete status flag will be turned on indicating the result of the comparison. In this example, if the value in the accumulator is less than the value specified in the Compare instruction, SP60 will turn on energizing C30.

DirectSOFT32

Load the value in V1400 and V1401 into the accumulator

Compare the value in the accumulator with the value in V1410 and V1411

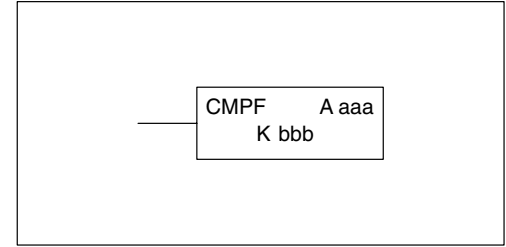
**Handheld Programmer Keystrokes**

STR	X(IN)	1	←							
LD	SHFT	D	SHFT	V	1	4	0	0	←	
CMP	SHFT	D	SHFT	V	1	4	1	0	←	
STR	SPCL	6	0	←						
OUT	C(CR)	3	0	←						

Compare Formatted (CMPF)

430 440 450

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.



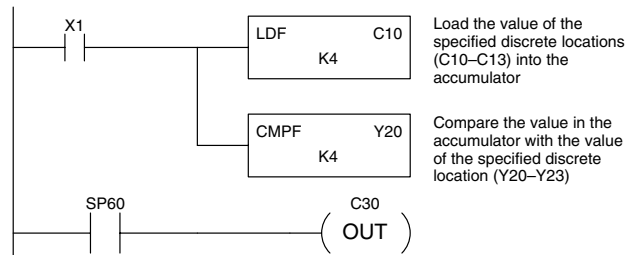
Operand Data Type	A/B	DL440 Range		DL450 Range	
		aaa	bbb	aaa	bbb
Inputs	X	0–477	—	0–1777	—
Outputs	Y	0–477	—	0–1777	—
Control Relays	C	0–1777	—	0–3777	—
Stage Bits	S	0–1777	—	0–1777	—
Timer Bits	T	0–377	—	0–377	—
Counter Bits	CT	0–177	—	0–377	—
Special Relays	SP	0–137 320–717	—	0–137 320–717	—
Global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

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

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

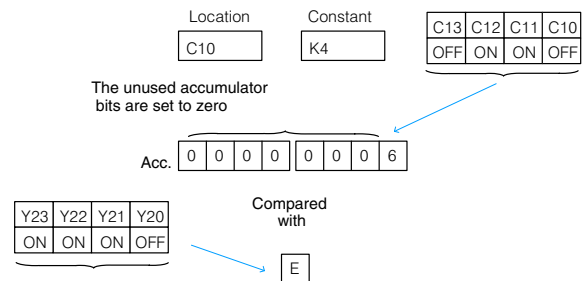
In the following example, when X1 is on the Load Formatted instruction loads the binary value (6) from C10–C13 into the accumulator. The CMPF instruction compares the value in the accumulator to the value in Y20–Y23 (E hex). The corresponding discrete status flag will be turned on indicating the result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

DirectSOFT32



Load the value of the specified discrete locations (C10–C13) into the accumulator

Compare the value in the accumulator with the value of the specified discrete location (Y20–Y23)



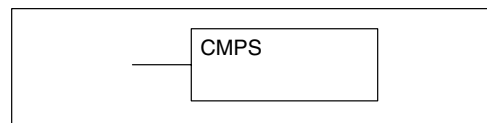
Handheld Programmer Keystrokes

STR	X(IN)	1	←
LD	SHFT	F	SHFT
		C	1
		0	K(CON)
		4	←
CMP	SHFT	F	SHFT
		Y(OUT)	2
		0	K(CON)
		4	←
STR	SPCL	6	←
OUT	C(CR)	3	←

Compare with Stack (CMPS)

✓ ✓ ✓
430 440 450

The Compare with Stack instruction is a 32-bit instruction that compares the value in the accumulator with the value in the first level of the accumulator stack.



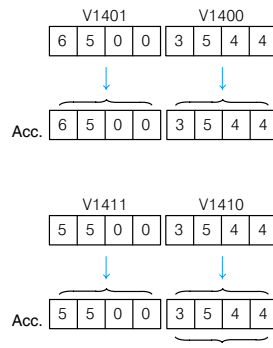
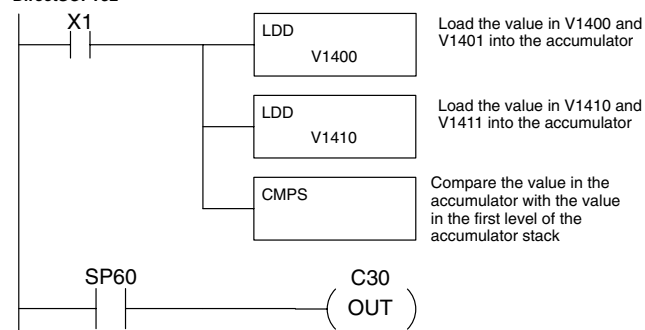
The corresponding status flag will be turned on indicating the result of the comparison. This does not affect the value in the accumulator.

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

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

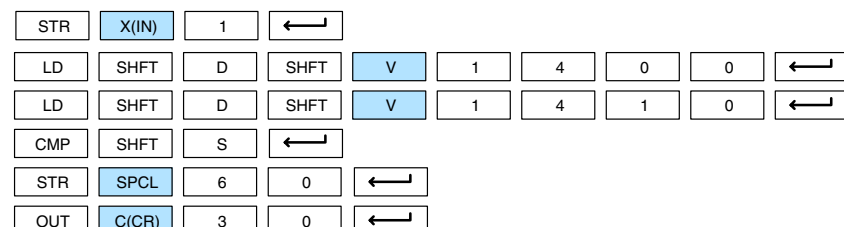
In the following example when X1 is on, the value in V1400 and V1401 is loaded into the accumulator using the Load Double instruction. The value in V1410 and V1411 is loaded into the accumulator using the Load Double instruction. The value that was loaded into the accumulator from V1400 and V1401 is placed on top of the stack when the second Load instruction is executed. The value in the accumulator is compared with the value in the first level of the accumulator stack using the CMPS instruction. The corresponding discrete status flag will be turned on indicating the result of the comparison. In this example, if the value in the accumulator is less than the value in the stack, SP60 will turn on, energizing C30.

DirectSOFT32



Compared with
Top of
Stack

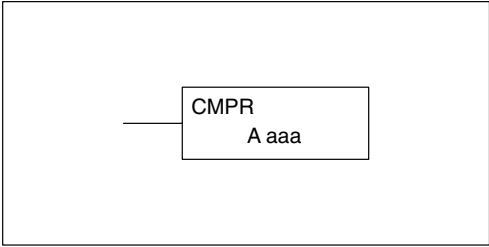
Handheld Programmer Keystrokes



Compare Real Number (CMPR)

☐ 430
 ☐ 440
 ☒ 450

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.

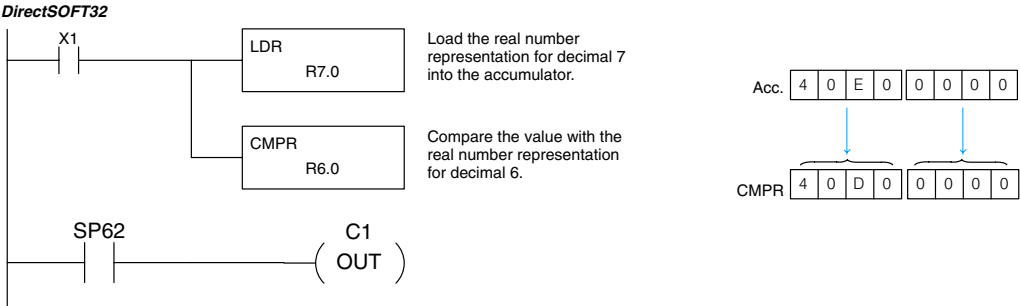


Operand Data Type	DL450 Range
A	aaa
Vmemory V	All (See p. 3-42)
Pointer P	All (See p. 3-42)
Constant R	-3.402823E+038 to +3.402823E+038

Discrete Bit Flags	Description
SP60	On when the value in the accumulator is less than the instruction value.
SP61	On when the value in the accumulator is equal to the instruction value.
SP62	On when the value in the accumulator is greater than the instruction value.
SP71	On anytime the V-memory specified by a pointer (P) is not valid.
SP75	On when a real number instruction is executed and a non-real number was encountered.

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

In the following example when X1 is on, the LDR instruction loads the real number representation for 7 decimal into the accumulator. The CMPR instruction compares the accumulator contents with the real representation for decimal 6. Since 7 > 6, the corresponding discrete status flag is turned on (special relay SP62).



Handheld Programmer Keystrokes

STR	X(IN)	1	LD	SHFT	D	SHFT	←
K(CON)	4	0	E	0	0	0	←
CMP	SHFT	R	SHFT				
K(CON)	4	0	D	0	0	0	←
STR	SPCL	6	0	←			
OUT	C(CR)	3	0	←			

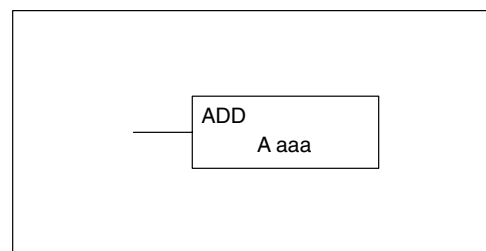
Math Instructions

Add (ADD)



430 440 450

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.

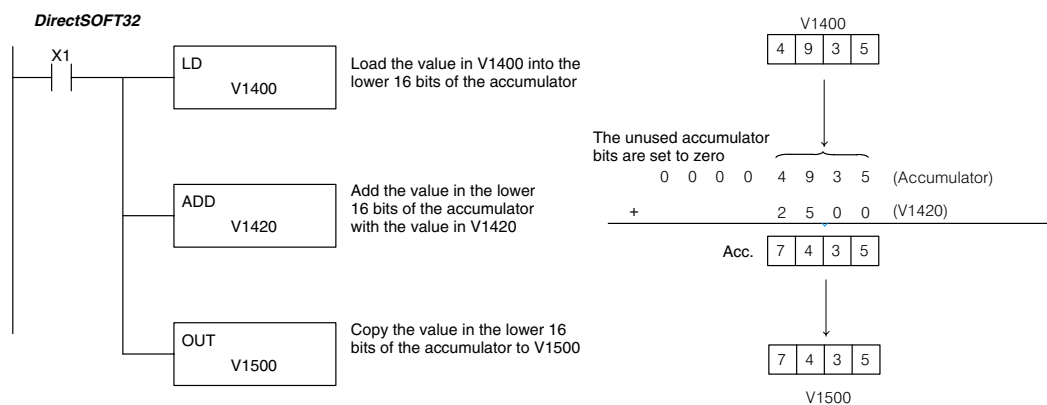


Operand Data Type		DL430 Range	DL440 Range	DL450 Range
A		aaa	aaa	aaa
Vmemory	V	All (See p. 3-40)	All (See p. 3-41)	All (See p. 3-42)
Pointer	P	All (See p. 3-40)	All (See p. 3-41)	All (See p. 3-42)

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

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

In the following example, when X1 is on, the value in V1400 will be loaded into the accumulator using the Load instruction. The value in the lower 16 bits of the accumulator are added to the value in V1420 using the Add instruction. The value in the accumulator is copied to V1500 using the Out instruction.



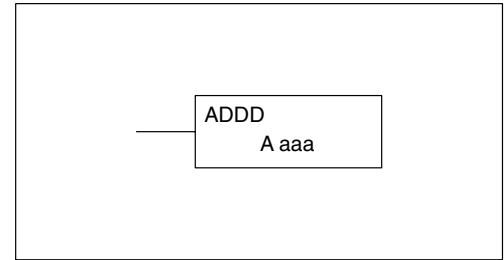
Handheld Programmer Keystrokes

STR	X(IN)	1	←
LD	V	1	4 0 0 ←
ADD	V	1	4 2 0 ←
OUT	V	1	5 0 0 ←

Add Double (ADDD)

✓ ✓ ✓
430 440 450

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.

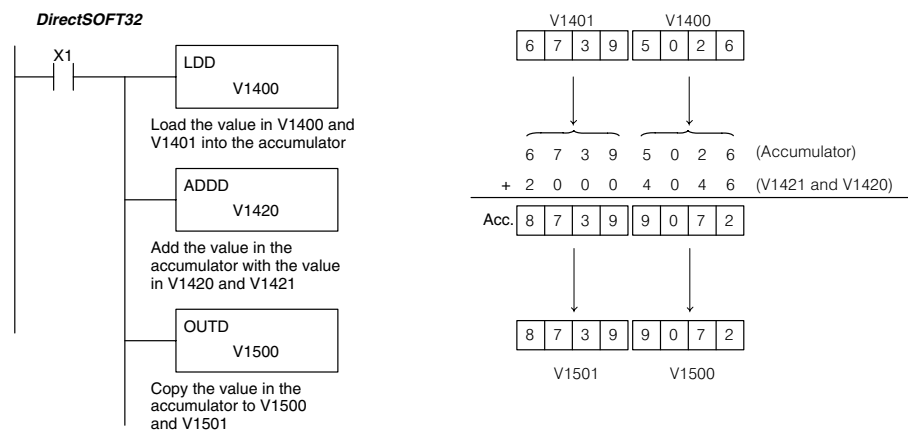


Operand Data Type		DL430 Range	DL440 Range	DL450 Range
A		aaa	aaa	aaa
Vmemory	V	All (See p. 3-40)	All (See p. 3-41)	All (See p. 3-42)
Pointer	P	All (See p. 3-40)	All (See p. 3-41)	All (See p. 3-42)
Constant	K	0-99999999	0-99999999	0-99999999

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

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

In the following example, when X1 is on, the value in V1400 and V1401 will be loaded into the accumulator using the Load Double instruction. The value in the accumulator is added with the value in V1420 and V1421 using the Add Double instruction. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.



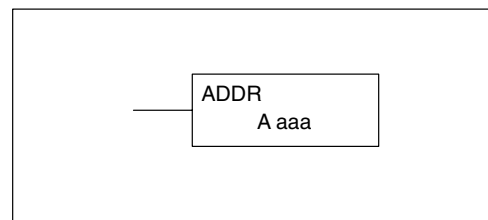
Handheld Programmer Keystrokes

STR	X(IN)	1	←						
LD	SHFT	D	SHFT	V	1	4	0	0	↵
ADD	SHFT	D	SHFT	V	1	4	2	0	↵
OUT	SHFT	D	SHFT	V	1	5	0	0	↵

**Add Real
(ADDR)**

X	X	✓
430	440	450

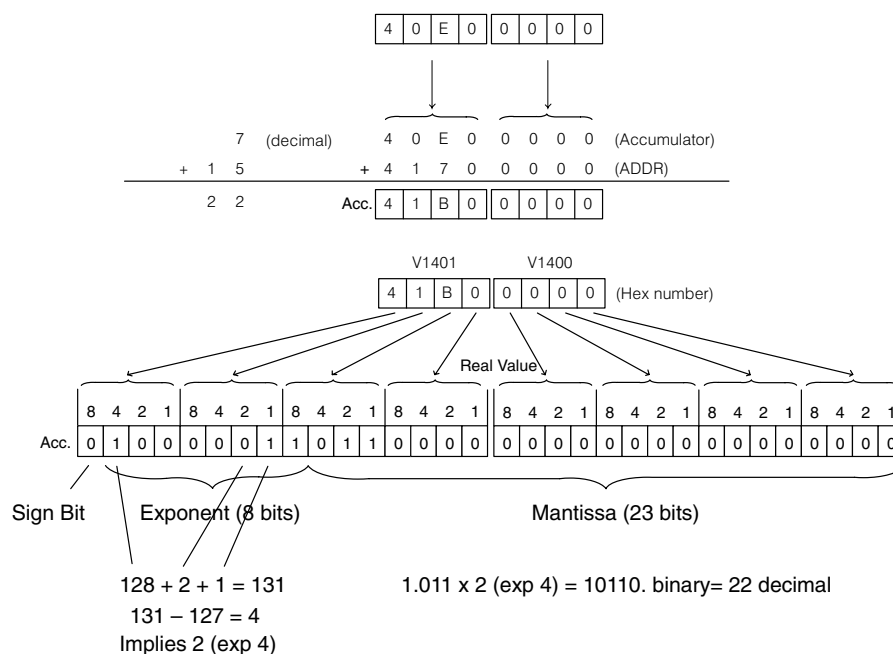
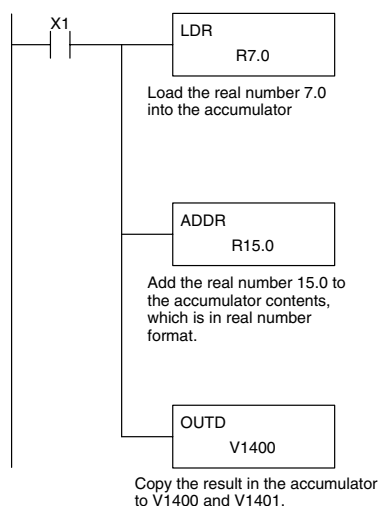
The Add Real instruction adds a real number in the accumulator with either a real constant or a real number occupying two consecutive V-memory locations. The result resides in the accumulator. Both numbers must conform to the IEEE floating point format.



Operand Data Type	DL450 Range
A	aaa
Vmemory V	All (See p. 3-42)
Pointer P	All V mem (See p. 3-42)
Constant R	-3.402823E+038 to +3.402823E+038

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.
SP71	On anytime the V-memory specified by a pointer (P) is not valid.
SP72	On anytime the value in the accumulator is a valid floating point number.
SP73	on when a signed addition or subtraction results in a incorrect sign bit.
SP74	On anytime a floating point math operation results in an underflow error.
SP75	On when a real number instruction is executed and a non-real number was encountered.

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

DirectSOFT32

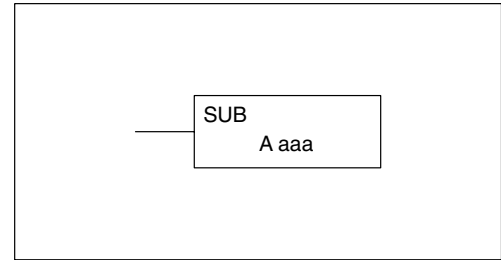
NOTE: If the value being added to a real number is 16,777,216 times smaller than the real number, the calculation will not work.

NOTE: The current HPP does not support real number entry with automatic conversion to the 32-bit IEEE format. You must use **DirectSOFT32** for this feature.

Subtract (SUB)

✓ ✓ ✓
430 440 450

Subtract is a 16 bit instruction that subtracts the BCD value (Aaaa) in a V memory location from the BCD value in the lower 16 bits of the accumulator. The result resides in the accumulator.

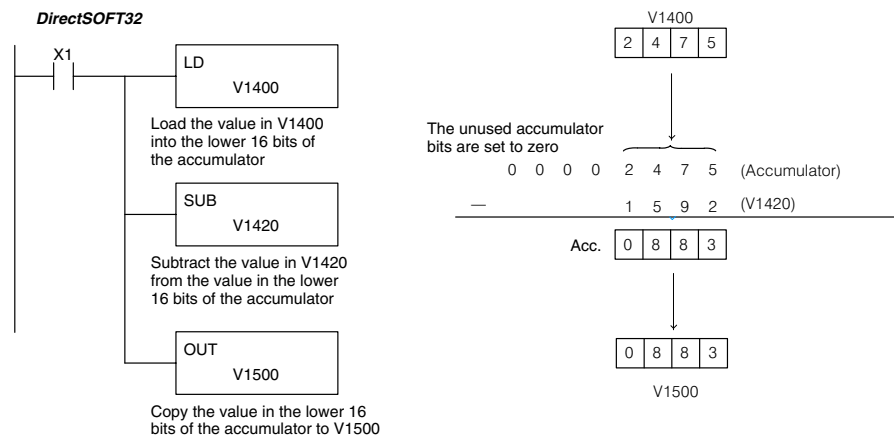


Operand Data Type		DL430 Range	DL440 Range	DL450 Range
A		aaa	aaa	aaa
A		aaa	aaa	aaa
Vmemory	V	All (See p. 3-40)	All (See p. 3-41)	All (See p. 3-42)
Pointer	P	All (See p. 3-40)	All (See p. 3-41)	All (See p. 3-42)

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

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

In the following example, when X1 is on, the value in V1400 will be loaded into the accumulator using the Load instruction. The value in V1420 is subtracted from the value in the accumulator using the Subtract instruction. The value in the accumulator is copied to V1500 using the Out instruction.



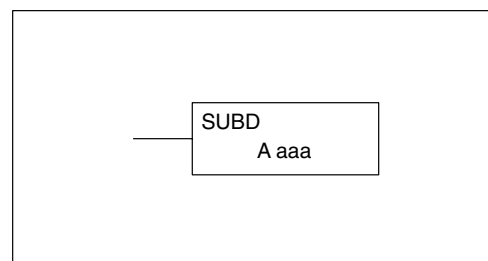
Handheld Programmer Keystrokes

STR	X(IN)	1				←
LD	V	1	4	0	0	←
SUB	V	1	4	2	0	←
OUT	V	1	5	0	0	←

Subtract Double (SUBD)



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.

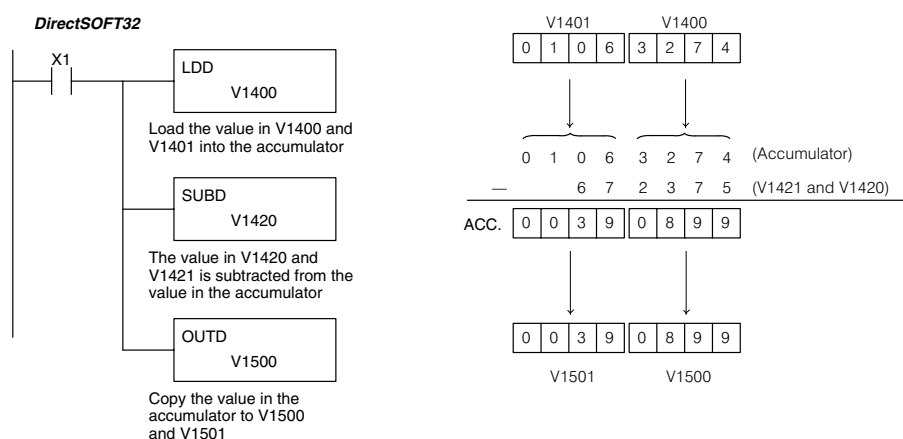


Operand Data Type		DL430 Range	DL440 Range	DL450 Range
	A	aaa	aaa	aaa
Vmemory	V	All (See p. 3–40)	All (See p. 3–41)	All (See p. 3–42)
Pointer	P	All (See p. 3–40)	All (See p. 3–41)	All (See p. 3–42)
Constant	K	0–99999999	0–99999999	0–99999999





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

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

In the following example, when X1 is on, the value in V1400 and V1401 will be loaded into the accumulator using the Load Double instruction. The value in V1420 and V1421 is subtracted from the value in the accumulator. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.



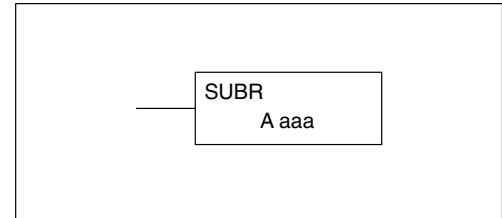
Handheld Programmer Keystrokes

STR	X(IN)	1								
LD	SHFT	D	SHFT	V	1	4	0	0		
SUB	SHFT	D	SHFT	V	1	4	2	0		
OUT	SHFT	D	SHFT	V	1	5	0	0		

Subtract Real (SUBR)

430 440 450

The Subtract Real instruction subtracts a real number in the accumulator from 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.

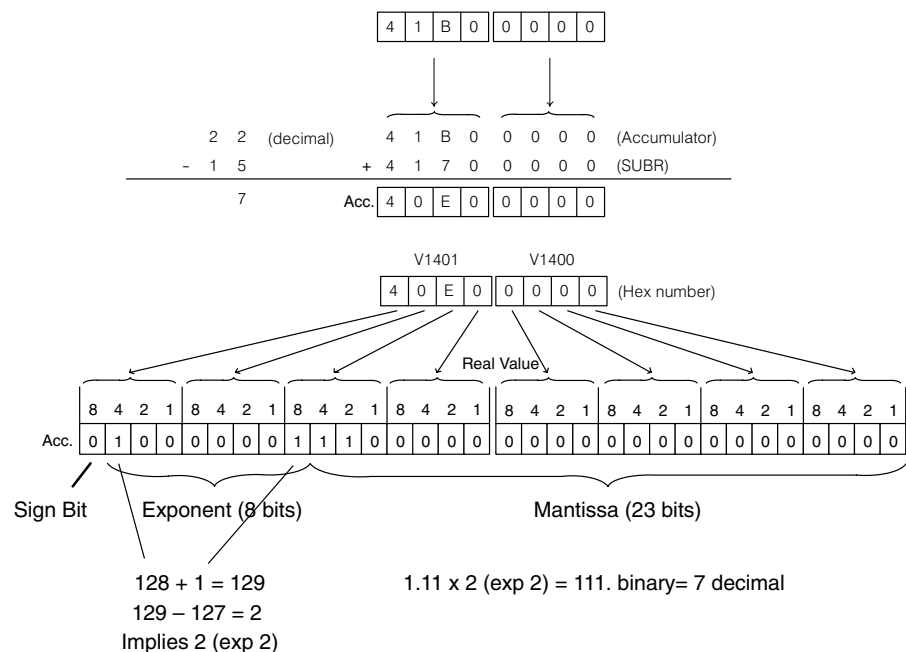
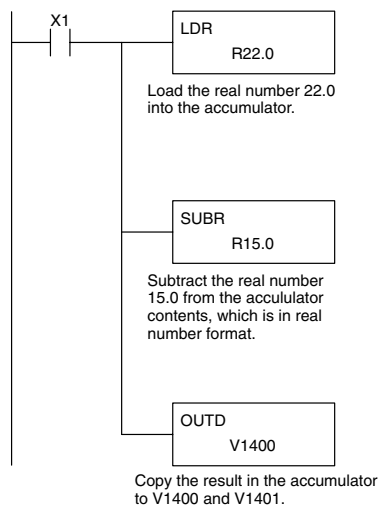


Operand Data Type	DL450 Range
A	aaa
Vmemory	V All (See p. 3-42)
Pointer	P All V mem (See p. 3-42)
Constant	R -3.402823E+038 to +3.402823E+038

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.
SP71	On anytime the V-memory specified by a pointer (P) is not valid.
SP72	On anytime the value in the accumulator is a valid floating point number.
SP73	On when a signed addition or subtraction results in a incorrect sign bit.
SP74	On anytime a floating point math operation results in an underflow error.
SP75	On when a real number instruction is executed and a non-real number was encountered.

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

DirectSOFT32

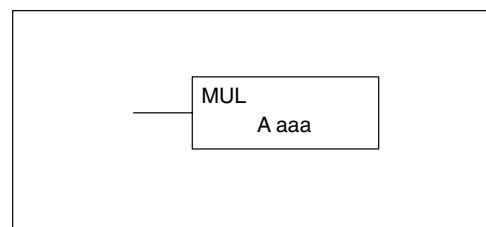


NOTE: The current HPP does not support real number entry with automatic conversion to the 32-bit IEEE format. You must use **DirectSOFT32** for this feature.

**Multiply
(MUL)**

✓	✓	✓
430	440	450

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.

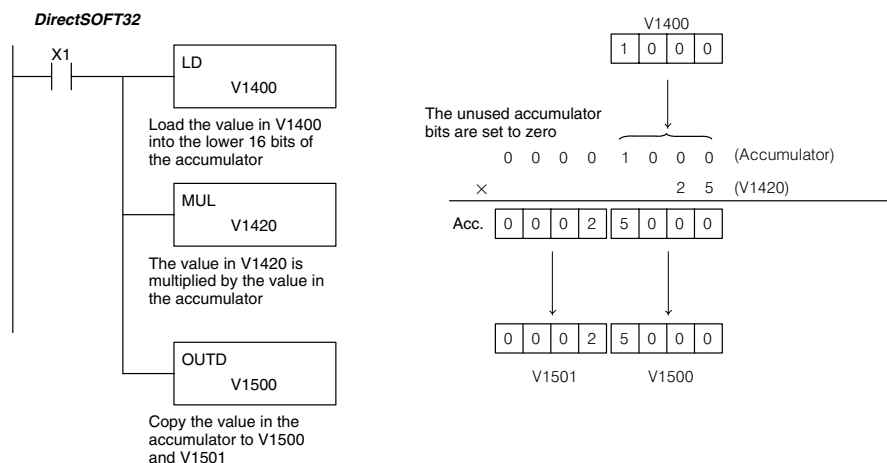


Operand Data Type		DL430 Range	DL440 Range	DL450 Range
A		aaa	aaa	aaa
Vmemory	V	All (See p. 3-40)	All (See p. 3-41)	All (See p. 3-42)
Pointer	P	All (See p. 3-40)	All (See p. 3-41)	All (See p. 3-42)
Constant	K	0-9999	0-9999	0-9999

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

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

In the following example, when X1 is on, the value in V1400 will be loaded into the accumulator using the Load instruction. The value in V1420 is multiplied by the value in the accumulator. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.

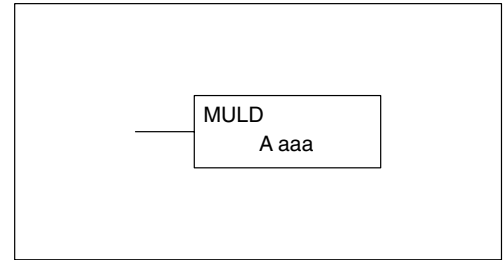
**Handheld Programmer Keystrokes**

STR	X(IN)	1	←
LD	V	1	4 0 0 ←
MUL	V	1	4 2 0 ←
OUT	SHFT	D	SHFT V 1 5 0 0 ←

Multiply Double (MULD)

☐ 430
 ☐ 440
 ☒ 450

Multiply Double is a 32 bit instruction that multiplies the 8-digit BCD value in the accumulator by the 8-digit BCD value in the two consecutive V-memory locations specified in the instruction. The lower 8 digits of the results reside in the accumulator. Upper digits of the result reside in the accumulator stack.

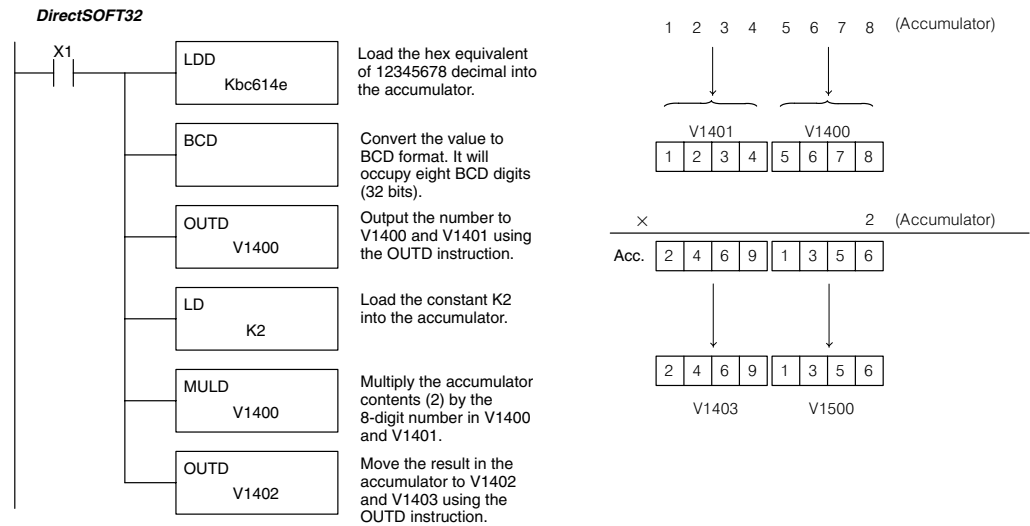


Operand Data Type	DL450 Range
A	aaa
Vmemory V	All (See p. 3-42)
Pointer P	—

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

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

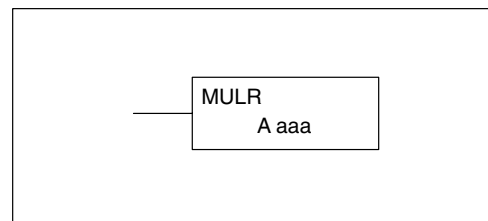
In the following example, when X1 is on, the constant Kbc614e hex will be loaded into the accumulator. When converted to BCD the number is "12345678". That number is stored in V1400 and V1401. After loading the constant K2 into the accumulator, we multiply it times 12345678, which is 24691356.



Multiply Real (MULR)

X	X	✓
430	440	450

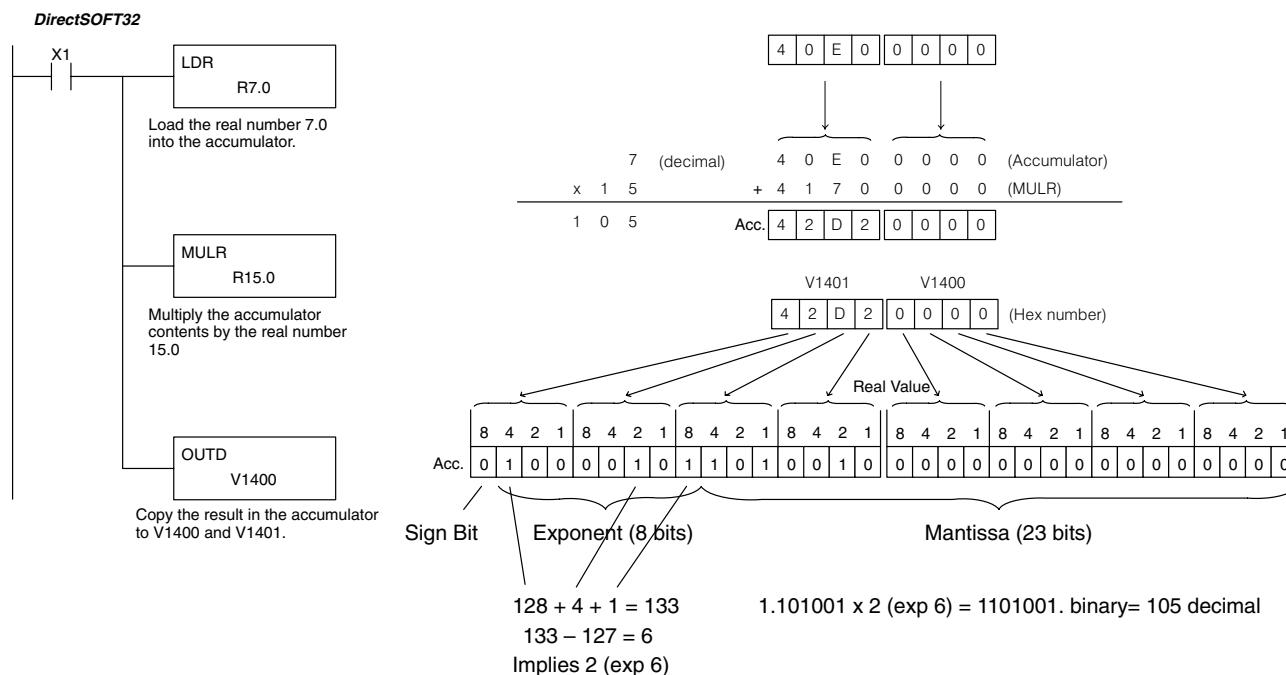
The Multiply Real instruction multiplies a real number in the accumulator with either a real constant or a real number occupying two consecutive V-memory locations. The result resides in the accumulator. Both numbers must conform to the IEEE floating point format.



Operand Data Type	DL450 Range
A	aaa
Vmemory V	All (See p. 3-42)
Pointer P	All (See p. 3-42)
Constant R	-3.402823E+038 to +3.402823E+038

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.
SP71	On anytime the V-memory specified by a pointer (P) is not valid.
SP72	On anytime the value in the accumulator is a valid floating point number.
SP73	on when a signed addition or subtraction results in a incorrect sign bit.
SP74	On anytime a floating point math operation results in an underflow error.
SP75	On when a real number instruction is executed and a non-real number was encountered.

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

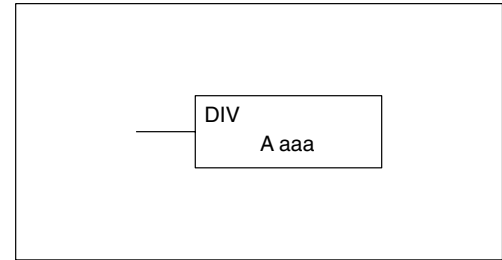


NOTE: The current HPP does not support real number entry with automatic conversion to the 32-bit IEEE format. You must use **DirectSOFT32** for this feature.

Divide (DIV)

✓ ✓ ✓
430 440 450

Divide is a 16 bit instruction that divides the BCD value in the 32-bit accumulator by a BCD value (Aaaa), which is either a V memory location or a 4-digit (max.) constant. The first part of the quotient resides in the accumulator and the remainder resides in the first stack location.

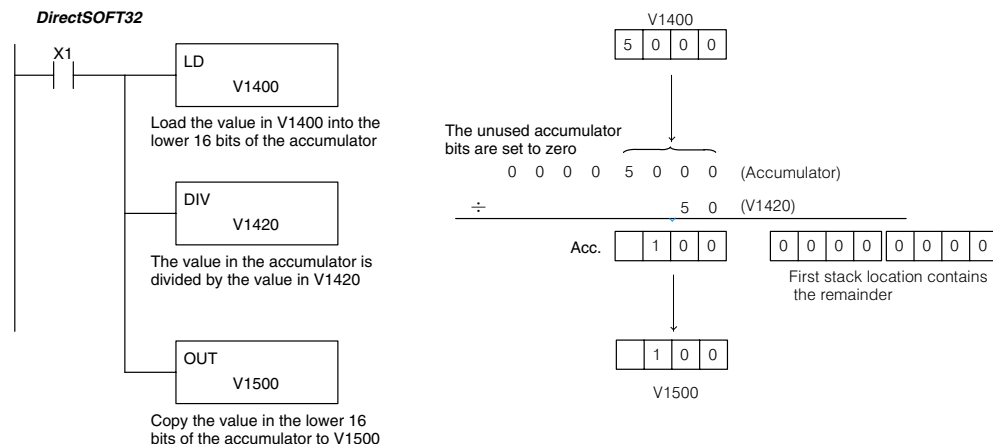


Operand Data Type		DL430 Range	DL440 Range	DL450 Range
A		aaa	aaa	aaa
Vmemory	V	All (See p. 3-40)	All (See p. 3-41)	All (See p. 3-42)
Pointer	P	All (See p. 3-40)	All (See p. 3-41)	All V mem (See p. 3-42)
Constant	K	0-9999	0-9999	0-9999

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

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

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



Handheld Programmer Keystrokes

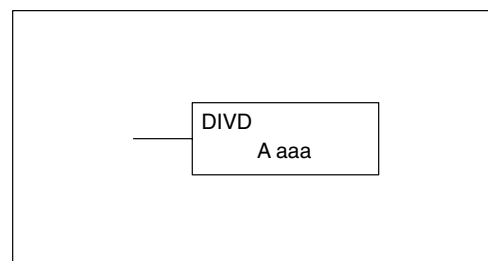
STR	X(IN)	1	←
LD	V	1	4 0 0 ←
DIV	V	1	4 2 0 ←
OUT	V	1	5 0 0 ←

Divide Double (DIVD)




 430 440 450

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.

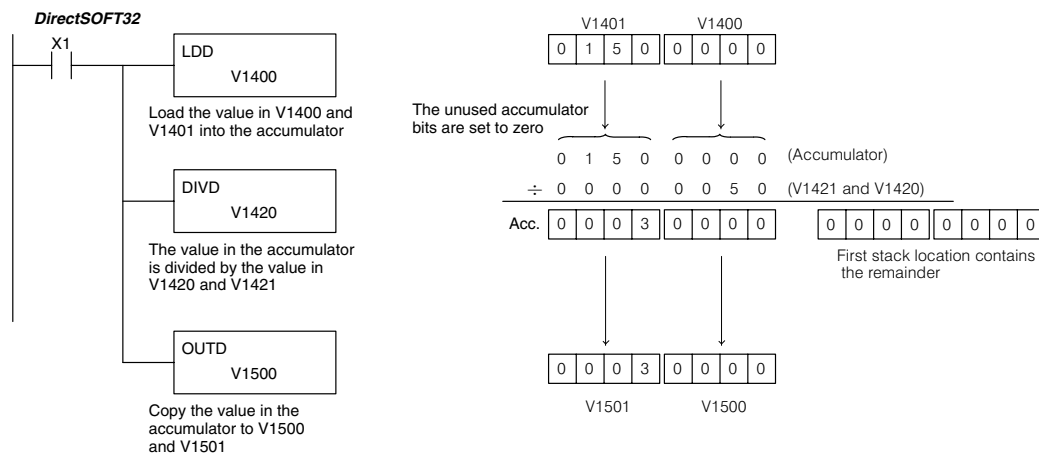


Operand Data Type		DL440 Range	DL450 Range
A		aaa	aaa
Vmemory	V	All (See p. 3-41)	All (See p. 3-42)
Pointer	P	All (See p. 3-41)	All (See p. 3-42)

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

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

In the following example, when X1 is on, the value in V1400 and V1401 will be loaded into the accumulator using the Load Double instruction. The value in the accumulator is divided by the value in V1420 and V1421 using the Divide Double instruction. The first part of the quotient resides in the accumulator and the remainder resides in the first stack location. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.

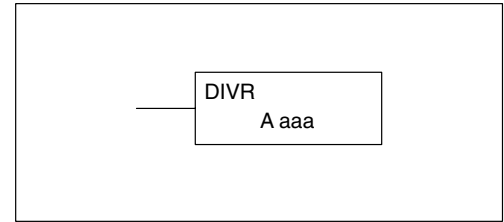
**Handheld Programmer Keystrokes**

STR	X(IN)	1	←						
LD	SHFT	D	SHFT	V	1	4	0	0	←
DIV	SHFT	D	SHFT	V	1	4	2	0	←
OUT	SHFT	D	SHFT	V	1	5	0	0	←

Divide Real (DIVR)

430 440 450

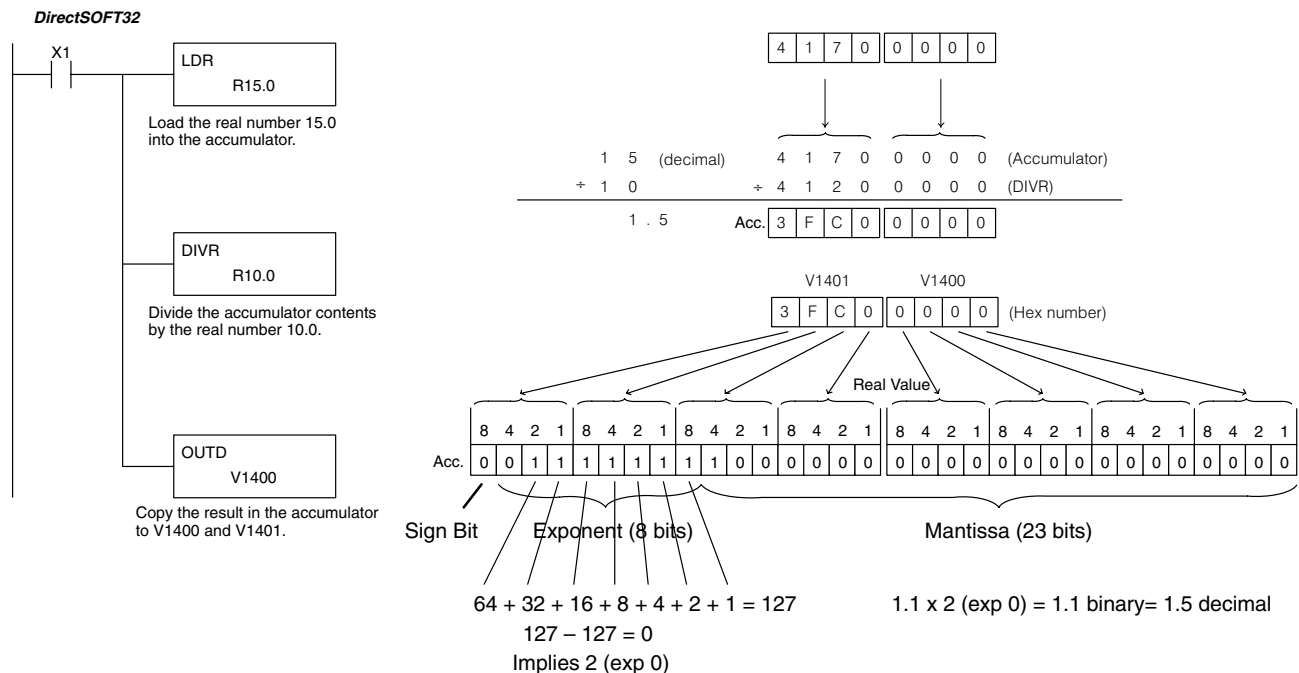
The Divide Real instruction divides a real number in the accumulator by either a real constant or a real number occupying two consecutive V-memory locations. The result resides in the accumulator. Both numbers must conform to the IEEE floating point format.



Operand Data Type	DL450 Range
A	aaa
Vmemory	V All (See p. 3-42)
Pointer	P All (See p. 3-42)
Constant	R -3.402823E+038 to +3.402823E+038

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.
SP71	On anytime the V-memory specified by a pointer (P) is not valid.
SP72	On anytime the value in the accumulator is a valid floating point number.
SP73	on when a signed addition or subtraction results in a incorrect sign bit.
SP74	On anytime a floating point math operation results in an underflow error.
SP75	On when a real number instruction is executed and a non-real number was encountered.

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

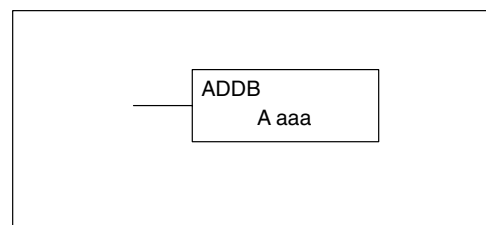


NOTE: The current HPP does not support real number entry with automatic conversion to the 32-bit IEEE format. You must use **DirectSOFT32** for this feature.

**Add Binary
(ADDB)**

✓ ✓ ✓
430 440 450

Add Binary is a 16 bit instruction that adds the unsigned 2's complement binary value in the lower 16 bits of the accumulator with an unsigned 2's complement binary value (Aaaa), which is either a V memory location or a 16-bit constant. The result can be up to 32 bits (unsigned 2's complement) and resides in the accumulator.

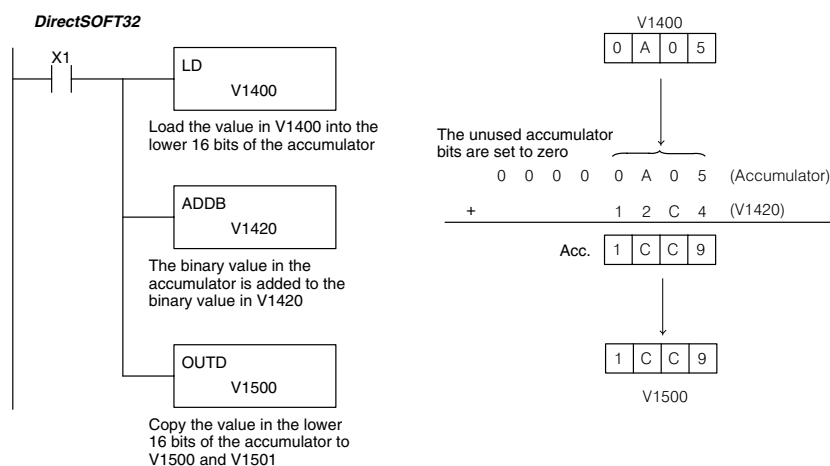


Operand Data Type		DL430 Range	DL440 Range	DL450 Range
A		aaa	aaa	aaa
Vmemory	V	All (See p. 3-40)	All (See p. 3-41)	All (See p. 3-42)
Pointer	P	All (See p. 3-40)	All V mem (See p. 3-41)	All V mem (See p. 3-42)
Constant	K	0-FFFF	0-FFFF	0-FFFF

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP66	On when the 16 bit addition instruction results in a carry.
SP67	On when the 32 bit addition instruction results in a carry.
SP70	On anytime the value in the accumulator is negative.
SP73	On when a signed addition or subtraction results in a incorrect sign bit.

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

In the following example, when X1 is on, the value in V1400 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.

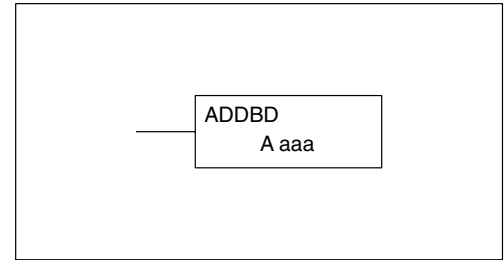
**Handheld Programmer Keystrokes**

STR	X(IN)	1	←
LD	V	1	4 0 0 ←
ADD	SHFT	B	SHFT V 1 4 2 0 ←
OUT	SHFT	D	SHFT V 1 5 0 0 ←

Add Binary Double (ADDBD)

430 440 450

Add Binary Double is a 32 bit instruction that adds the unsigned 2's complement binary value in the accumulator with the value (Aaaa), which is either two consecutive V memory locations or 32-bit unsigned 2's complement binary constant. The result resides in the accumulator.

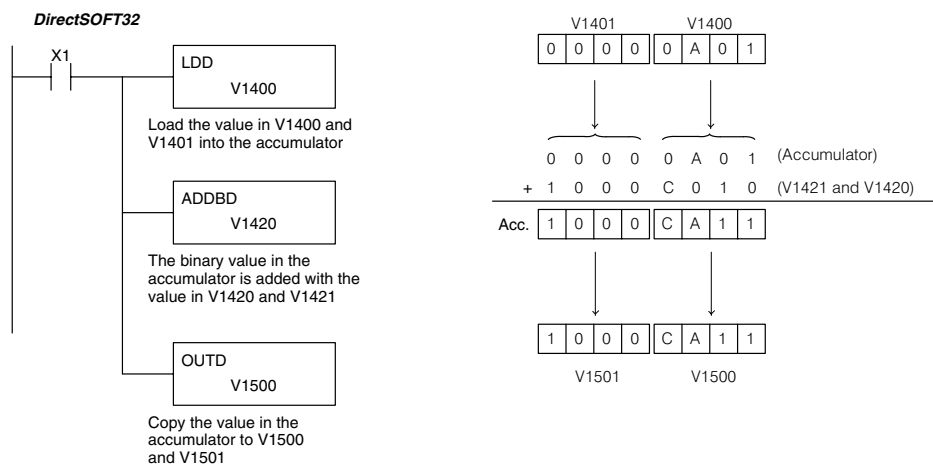


Operand Data Type	DL440 Range	DL450 Range
A	aaa	aaa
Vmemory V	All (See p. 3-41)	All (See p. 3-42)
Pointer P	All (See p. 3-41)	All V mem (See p. 3-42)
Constant K	0-FFFFFFFF	0-FFFFFFFF

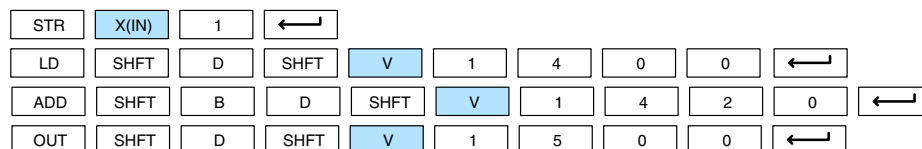
Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP66	On when the 16 bit addition instruction results in a carry.
SP67	On when the 32 bit addition instruction results in a carry.
SP70	On anytime the value in the accumulator is negative.
SP73	On when a signed addition or subtraction results in a incorrect sign bit.

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

In the following example, when X1 is on, the value in V1400 and V1401 will be loaded into the accumulator using the Load Double instruction. The binary value in the accumulator is added with the binary value in V1420 and V1421 using the Add Binary Double instruction. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.



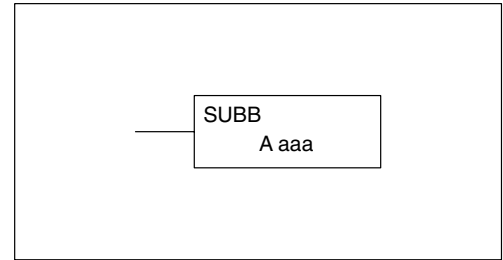
Handheld Programmer Keystrokes



Subtract Binary (SUBB)

✓ ✓ ✓
430 440 450

Subtract Binary is a 16 bit instruction that subtracts the unsigned 2's complement binary value (Aaaa), which is either a V memory location or a 16-bit 2's complement binary value, from the binary value in the accumulator. The result resides in the accumulator.

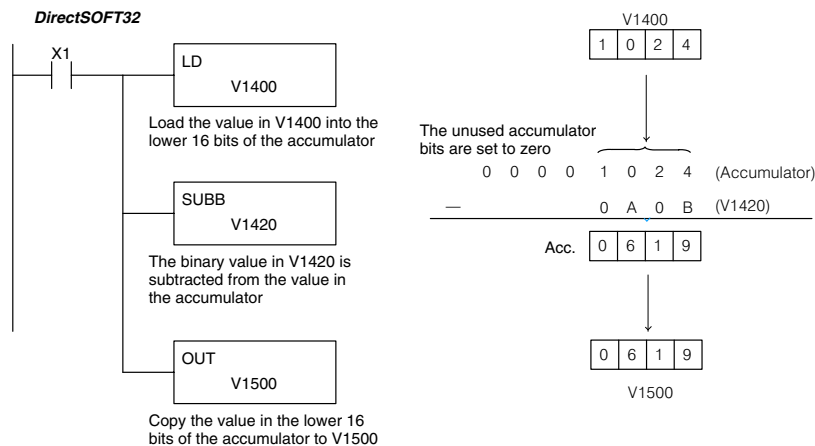


Operand Data Type		DL430 Range	DL440 Range	DL450 Range
A		aaa	aaa	aaa
Vmemory	V	All (See p. 3-40)	All (See p. 3-41)	All (See p. 3-42)
Pointer	P	All (See p. 3-40)	All (See p. 3-41)	All (See p. 3-42)
Constant	K	0-FFFF	0-FFFF	0-FFFF

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP64	On when the 16 bit subtraction instruction results in a borrow.
SP65	On when the 32 bit subtraction instruction results in a borrow.
SP70	On anytime the value in the accumulator is negative.

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

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



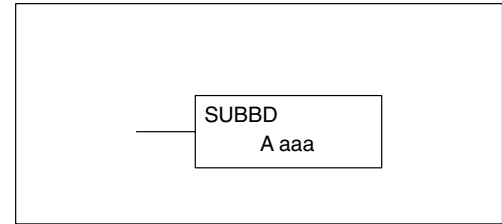
Handheld Programmer Keystrokes

STR	X(IN)	1	←
LD	V	1	4 0 0 ←
SUB	SHFT	B	SHFT V 1 4 2 0 ←
OUT	V	1	5 0 0 ←

Subtract Binary Double (SUBBD)

✕ ✓ ✓
430 440 450

Subtract Binary Double is a 32 bit instruction that subtracts the unsigned 2's complement binary value (Aaaa), which is either two consecutive V memory locations or a 32-bit unsigned 2's complement binary constant, from the binary value in the accumulator. The result resides in the accumulator.

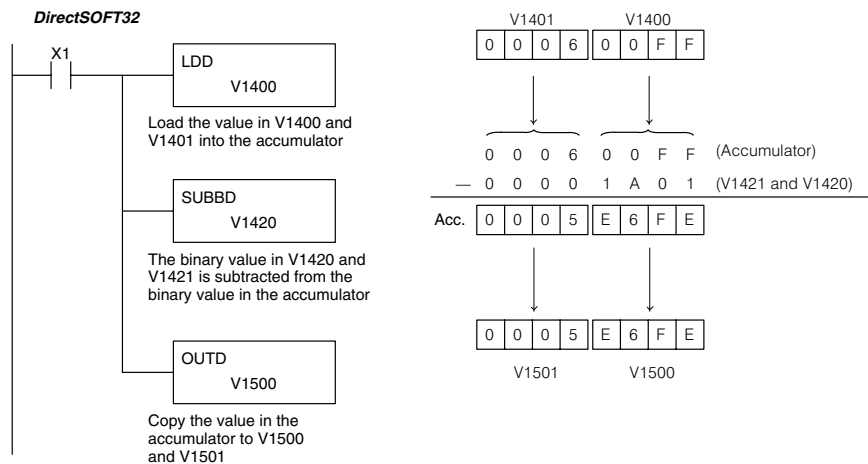


Operand Data Type		DL440 Range	DL450 Range
A		aaa	aaa
Vmemory	V	All (See p. 3-41)	All (See p. 3-42)
Pointer	P	All (See p. 3-42)	All (See p. 3-42)
Constant	K	0-FFFFFFFF	0-FFFFFFFF

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP64	On when the 16 bit subtraction instruction results in a borrow.
SP65	On when the 32 bit subtraction instruction results in a borrow.
SP70	On anytime the value in the accumulator is negative.

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

In the following example, when X1 is on, the value in V1400 and V1401 will be loaded into the accumulator using the Load Double instruction. The binary value in V1420 and V1421 is subtracted from the binary value in the accumulator using the Subtract Binary Double instruction. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.



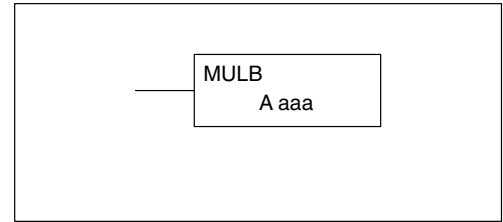
Handheld Programmer Keystrokes

STR	X(IN)	1	←
LD	SHFT	D	SHFT
	V	1	4
	0	0	←
SUB	SHFT	B	D
	SHFT	V	1
	4	2	0
OUT	SHFT	D	SHFT
	V	1	5
	0	0	←

Multiply Binary (MULB)

✓ ✓ ✓
430 440 450

Multiply Binary is a 16 bit instruction that multiplies the unsigned 2's complement binary value (Aaaa), which is either a V memory location or a 16-bit unsigned 2's complement binary constant, by the 16-bit binary value in the accumulator. The result can be up to 32 bits and resides in the accumulator.

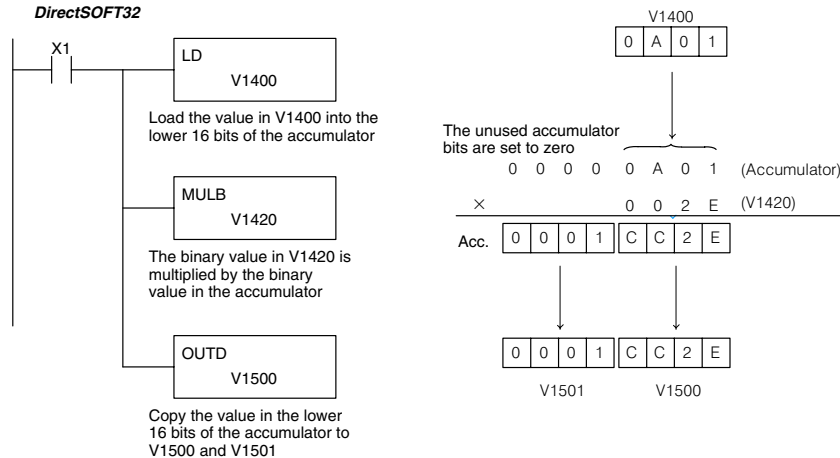


Operand Data Type		DL430 Range	DL440 Range	DL450 Range
A		aaa	aaa	aaa
Vmemory	V	All (See p. 3-40)	All (See p. 3-41)	All (See p. 3-42)
Pointer	P	All (See p. 3-40)	All (See p. 3-41)	All (See p. 3-42)
Constant	K	0-FFFF	0-FFFF	0-FFFF

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

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

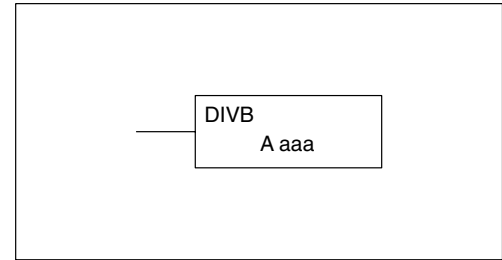
In the following example, when X1 is on, the value in V1400 will be loaded into the accumulator using the Load instruction. The binary value in V1420 is multiplied by the binary value in the accumulator using the Multiply Binary instruction. The value in the accumulator is copied to V1500 using the Out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



Divide Binary (DIVB)

✓ ✓ ✓
430 440 450

Divide Binary is a 16 bit instruction that divides the unsigned 2's complement binary value in the accumulator by a binary value (Aaaa), which is either a V memory location or a 16-bit unsigned 2's complement binary constant. The first part of the quotient resides in the accumulator and the remainder resides in the first stack location.

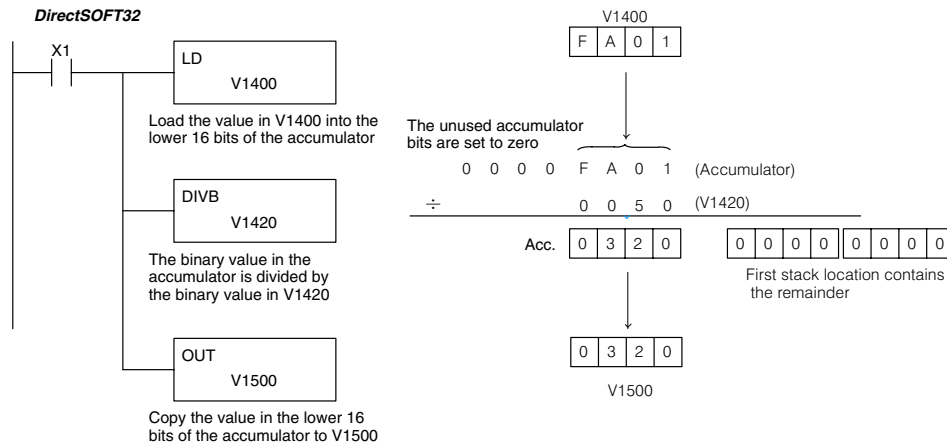


Operand Data Type		DL430 Range	DL440 Range	DL450 Range
A		aaa	aaa	aaa
Vmemory	V	All (See p. 3-40)	All (See p. 3-41)	All (See p. 3-42)
Pointer	P	All (See p. 3-40)	All (See p. 3-41)	All (See p. 3-42)
Constant	K	0-FFFF	0-FFFF	0-FFFF

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

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

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



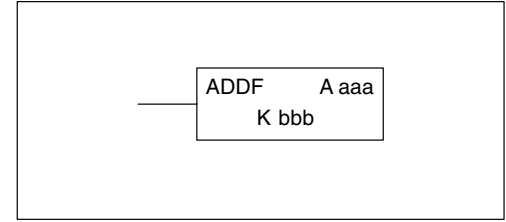
Handheld Programmer Keystrokes

STR	X(IN)	1	←
LD	V	1	4 0 0 ←
DIV	SHFT	B	SHFT V 1 4 2 0 ←
OUT	V	1	5 0 0 ←

Add Formatted (ADDF)

☒ ☒ ☒
 430 440 450

Add Formatted is a 32 bit instruction that adds the BCD value in the accumulator with the BCD value (Aaaa) which is a range of discrete bits. The specified range (Kbbb) can be 1 to 32 consecutive bits. The result resides in the accumulator.



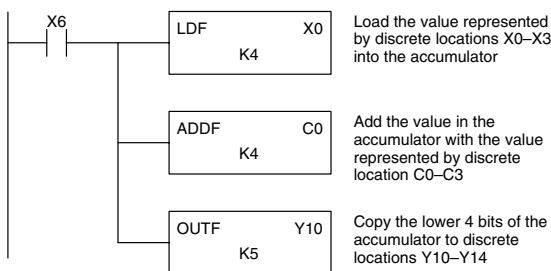
Operand Data Type	A	DL440 Range		DL450 Range	
		aaa	bbb	aaa	bbb
Inputs	X	0–477	—	0–1777	—
Outputs	Y	0–477	—	0–1777	—
Control Relays	C	0–1777	—	0–3777	—
Stage Bits	S	0–1777	—	0–1777	—
Timer Bits	T	0–377	—	0–377	—
Counter Bits	CT	0–177	—	0–377	—
Special Relays	SP	0–137 320–717	—	0–137 320–717	—
Global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

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

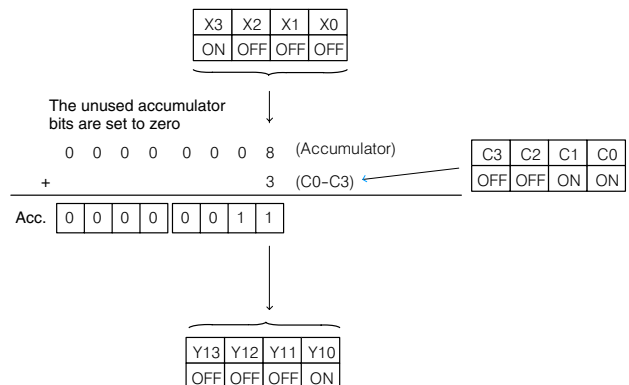
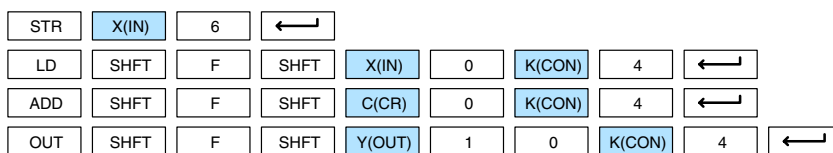
NOTE: Status flags are valid only until another instruction uses the same flag.

In the following example, when X6 is on, the value formed by discrete locations X0–X3 is loaded into the accumulator using the Load Formatted instruction. The value formed by discrete locations C0–C3 is added to the value in the accumulator using the Add Formatted instruction. The value in the lower four bits of the accumulator is copied to Y10–Y13 using the Out Formatted instruction.

DirectSOFT32



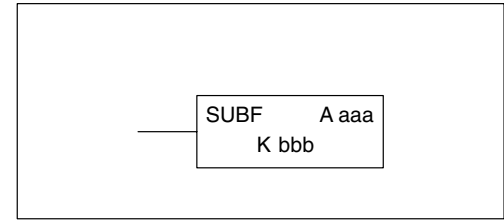
Handheld Programmer Keystrokes



Subtract Formatted (SUBF)

☐ ☒ ☒
 430 440 450

Subtract Formatted is a 32 bit instruction that subtracts the BCD value (Aaaa), which is a range of discrete bits, from the BCD value in the accumulator. The specified range (Kbbb) can be 1 to 32 consecutive bits. The result resides in the accumulator.

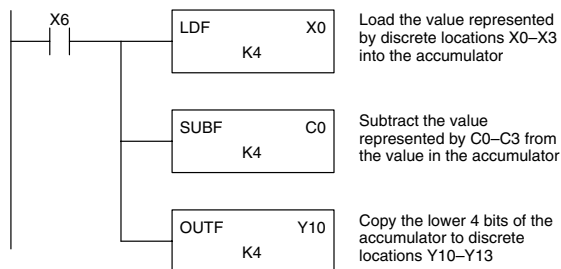


Operand Data Type		DL440 Range		DL450 Range	
	A	aaa	bbb	aaa	bbb
Inputs	X	0-477	—	0-1777	—
Outputs	Y	0-477	—	0-1777	—
Control Relays	C	0-1777	—	0-3777	—
Stage Bits	S	0-1777	—	0-1777	—
Timer Bits	T	0-377	—	0-377	—
Counter Bits	CT	0-177	—	0-377	—
Special Relays	SP	0-137 320-717	—	0-137 320-717	—
Global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

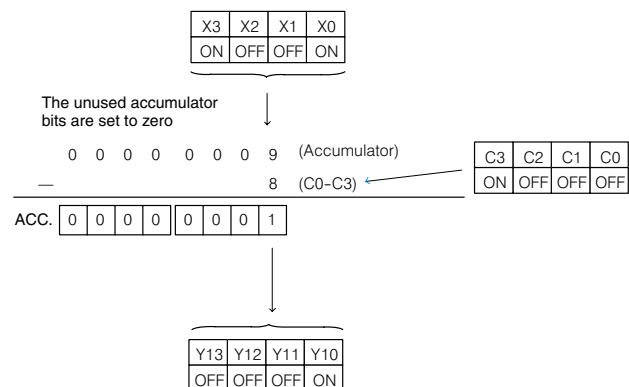
Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP64	On when the 16 bit subtraction instruction results in a borrow.
SP65	On when the 32 bit subtraction instruction results in a borrow.
SP70	On anytime the value in the accumulator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

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

In the following example, when X6 is on, the value formed by discrete locations X0-X3 is loaded into the accumulator using the Load Formatted instruction. The value formed by discrete location C0-C3 is subtracted from the value in the accumulator using the Subtract Formatted instruction. The value in the lower four bits of the accumulator is copied to Y10-Y13 using the Out Formatted instruction.

DirectSOFT32**Handheld Programmer Keystrokes**

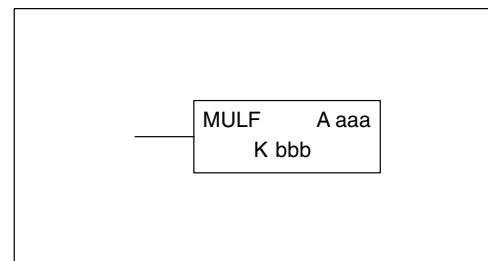
STR	X(IN)	6	←
LD	SHFT	F	SHFT X(IN) 0 K(CON) 4 ←
SUBF	SHFT	F	SHFT C(CR) 0 K(CON) 4 ←
OUT	SHFT	F	SHFT Y(OUT) 1 0 K(CON) 4 ←



Multiply Formatted (MULF)

430 440 450

Multiply Formatted is a 16 bit instruction that multiplies the BCD value in the accumulator by the BCD value (Aaaa) which is a range of discrete bits. The specified range (Kbbb) can be 1 to 16 consecutive bits. The result resides in the accumulator.

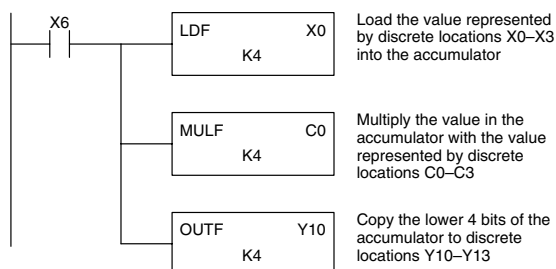


Operand Data Type		DL440 Range		DL450 Range	
	A/B	aaa	bbb	aaa	bbb
Inputs	X	0-477	—	0-1777	—
Outputs	Y	0-477	—	0-1777	—
Control Relays	C	0-1777	—	0-3777	—
Stage Bits	S	0-1777	—	0-1777	—
Timer Bits	T	0-377	—	0-377	—
Counter Bits	CT	0-177	—	0-377	—
Special Relays	SP	0-137 320-717	—	0-137 320-717	—
Global I/O	GX	0-1777	—	0-2777	—
Constant	K	—	1-16	—	1-16

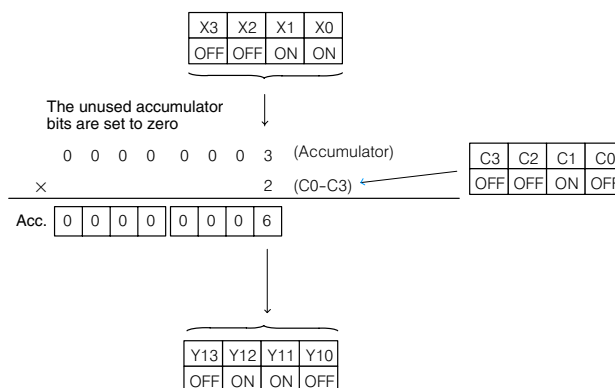
Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.
SP75	On when a BCD instruction is executed and a NON-BCD number was encountered.

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

In the following example, when X6 is on, the value formed by discrete locations X0-X3 is loaded into the accumulator using the Load Formatted instruction. The value formed by discrete locations C0-C3 is multiplied by the value in the accumulator using the Multiply Formatted instruction. The value in the lower four bits of the accumulator is copied to Y10-Y13 using the Out Formatted instruction.

DirectSOFT32**Handheld Programmer Keystrokes**

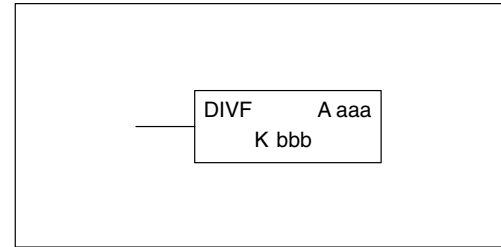
STR	X(IN)	6	←
LD	SHFT	F	SHFT
MUL	SHFT	F	SHFT
OUT	SHFT	F	SHFT
	X(IN)	0	K(CON)
	C(CR)	0	K(CON)
	Y(OUT)	1	0
			K(CON)



**Divide Formatted
(DIVF)**

☐ ☒ ☒
 430 440 450

Divide Formatted is a 16 bit instruction that divides the BCD value in the accumulator by the BCD value (Aaaa), a range of discrete bits. The specified range (Kbbb) can be 1 to 16 consecutive bits. The first part of the quotient resides in the accumulator and the remainder resides in the first stack location.

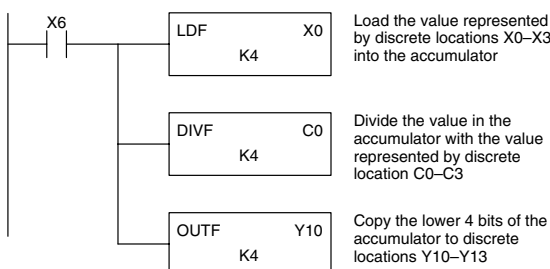


Operand Data Type	A/B	DL440 Range		DL450 Range	
		aaa	bbb	aaa	bbb
Inputs	X	0-477	—	0-477	—
Outputs	Y	0-477	—	0-477	—
Control Relays	C	0-1777	—	0-1777	—
Stage Bits	S	0-1777	—	0-1777	—
Timer Bits	T	0-377	—	0-377	—
Counter Bits	CT	0-177	—	0-177	—
Special Relays	SP	0-137 320-717	—	0-137 320-717	—
Global I/O	GX	0-1777	—	0-1777	—
Constant	K	—	1-16	—	1-16

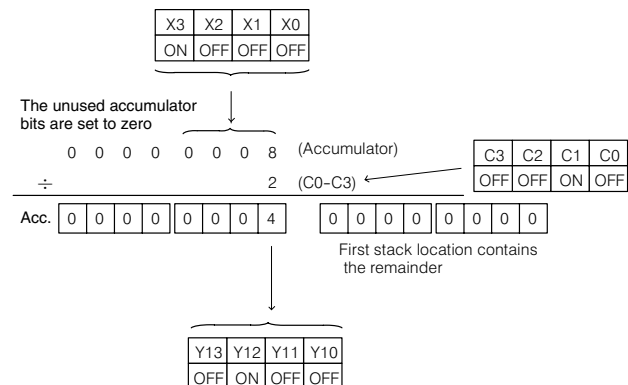
Discrete Bit Flags	Description
SP53	On when the value of the operand is larger than the accumulator can work with.
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.
SP75	On when a BCD instruction is executed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

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

In the following example, when X6 is on, the value formed by discrete locations X0-X3 is loaded into the accumulator using the Load Formatted instruction. The value in the accumulator is divided by the value formed by discrete location C0-C3 using the Divide Formatted instruction. The value in the lower four bits of the accumulator is copied to Y10-Y13 using the Out Formatted instruction.

DirectSOFT32**Handheld Programmer Keystrokes**

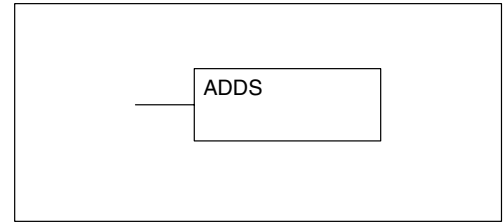
STR	X(IN)	6	←
LD	SHFT	F	SHFT
DIV	SHFT	F	SHFT
OUT	SHFT	F	SHFT
	X(IN)	0	K(CON)
	C(CR)	0	K(CON)
	Y(OUT)	1	0
			K(CON)



Add Top of Stack (ADDS)

✓ ✓ ✓
430 440 450

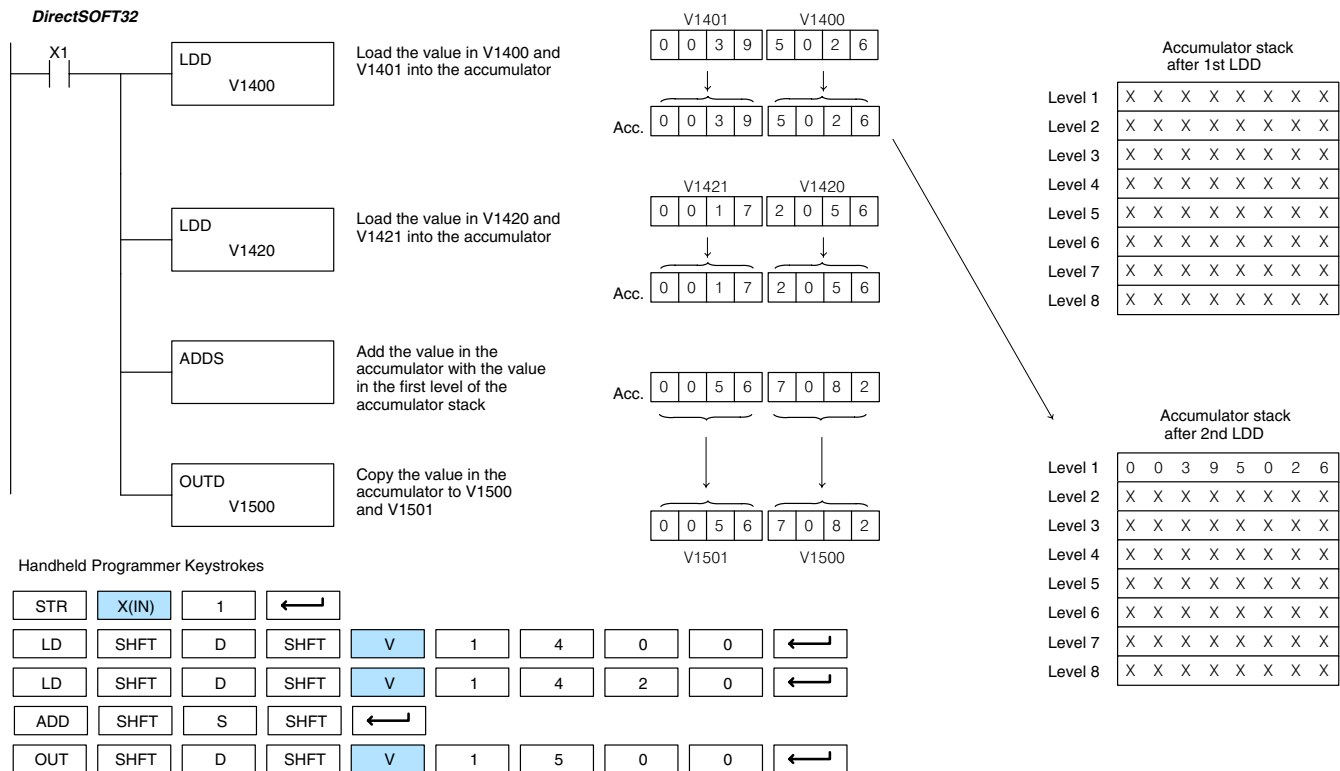
Add Top of Stack is a 32 bit instruction that adds the BCD value in the accumulator with the BCD value in the first level of the accumulator stack. The result resides in the accumulator. The value in the first level of the accumulator stack is removed and all stack values are moved up one level.



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

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

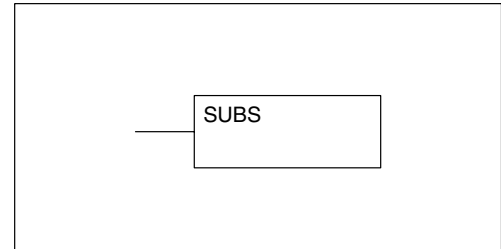
In the following example, when X1 is on, the value in V1400 and V1401 will be loaded into the accumulator using the Load Double instruction. The value in V1420 and V1421 is loaded into the accumulator using the Load Double instruction, pushing the value previously loaded in the accumulator onto the accumulator stack. The value in the first level of the accumulator stack is added with the value in the accumulator using the Add Stack instruction. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.



Subtract Top of Stack (SUBS)

✓ ✓ ✓
430 440 450

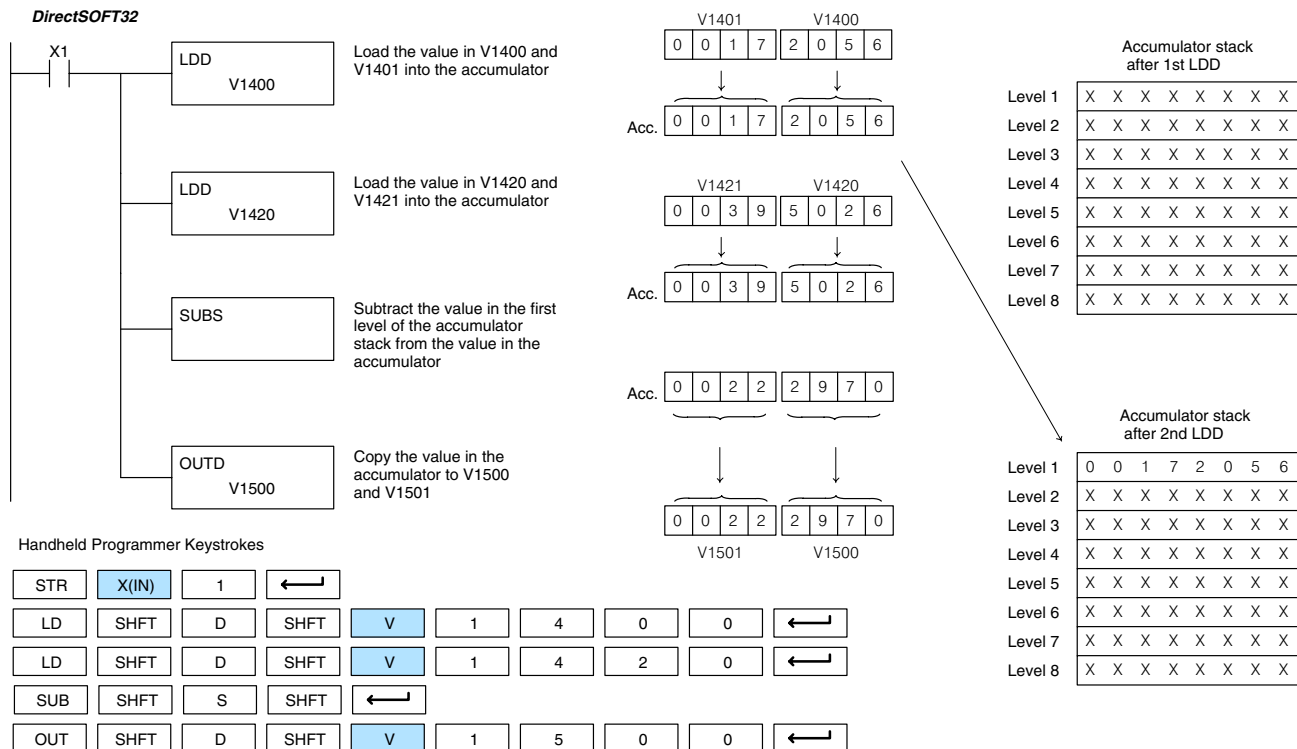
Subtract Top of Stack is a 32 bit instruction that subtracts the BCD value in the first level of the accumulator stack from the BCD value in the accumulator. The result resides in the accumulator. The value in the first level of the accumulator stack is removed and all stack values are moved up one level.



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

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

In the following example, when X1 is on, the value in V1400 and V1401 will be loaded into the accumulator using the Load Double instruction. The value in V1420 and V1421 is loaded into the accumulator using the Load Double instruction, pushing the value previously loaded into the accumulator onto the accumulator stack. The BCD value in the first level of the accumulator stack is subtracted from the BCD value in the accumulator using the Subtract Stack instruction. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.



Multiply Top of Stack (MULS)

✓ ✓ ✓
430 440 450

Multiply Top of Stack is a 16 bit instruction that multiplies a 4-digit BCD value in the first level of the accumulator stack by a 4-digit BCD value in the accumulator. The result resides in the accumulator. The value in the first level of the accumulator stack is removed and all stack values are moved up one level.

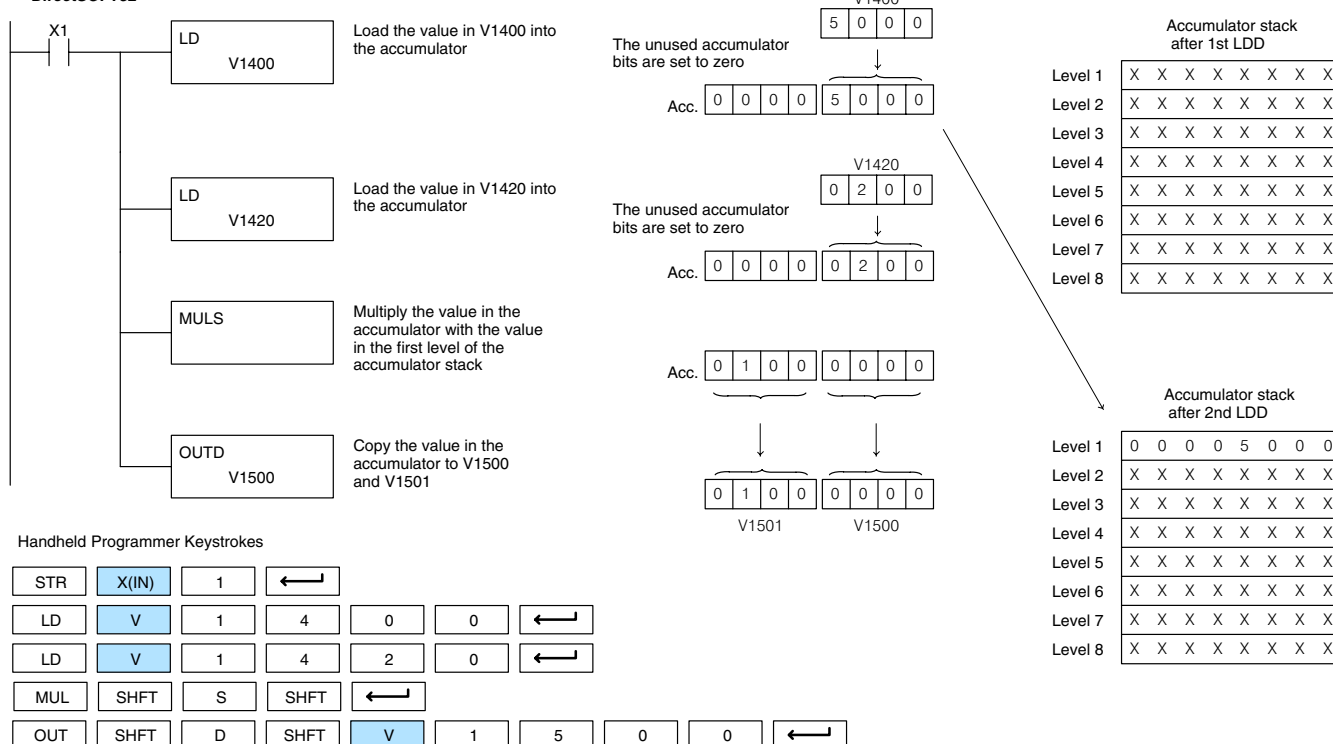
MULS

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

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

In the following example, when X1 is on, the value in V1400 will be loaded into the accumulator using the Load instruction. The value in V1420 is loaded into the accumulator using the Load Double instruction, pushing the value previously loaded in the accumulator onto the accumulator stack. The BCD value in the first level of the accumulator stack is multiplied by the BCD value in the accumulator using the Multiply Stack instruction. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.

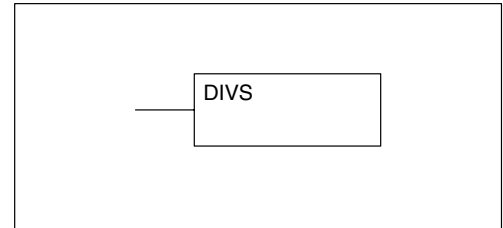
DirectSOFT32



Divide by Top of Stack (DIVS)

430 440 450

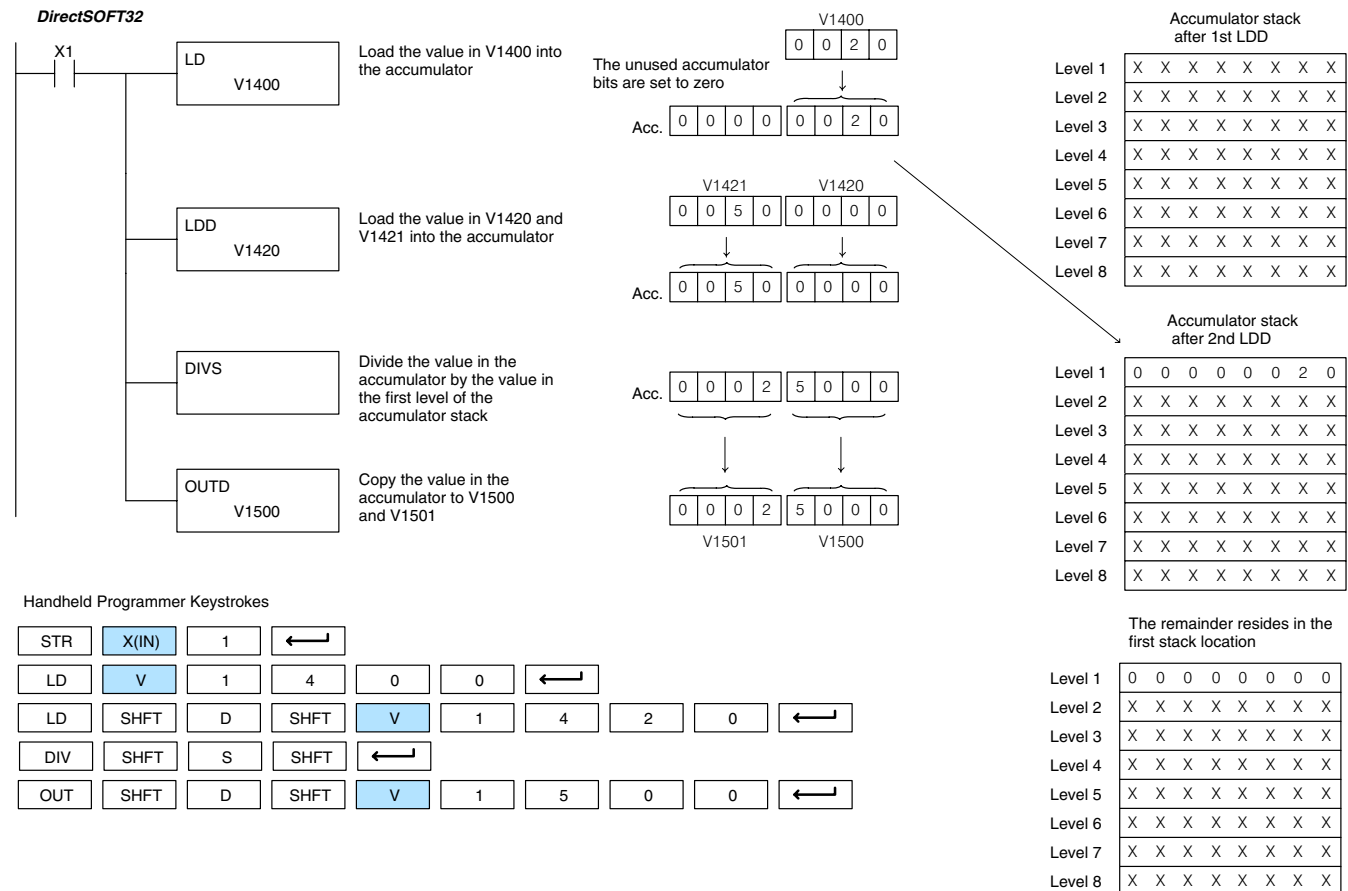
Divide Top of Stack is a 32 bit instruction that divides the 8-digit BCD value in the accumulator by a 4-digit BCD value in the first level of the accumulator stack. The result resides in the accumulator and the remainder resides in the first level of the accumulator stack.



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

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

In the following example, when X1 is on, the Load instruction loads the value in V1400 into the accumulator. The value in V1420 is loaded into the accumulator using the Load Double instruction, pushing the value previously loaded in the accumulator onto the accumulator stack. The BCD value in the accumulator is divided by the BCD value in the first level of the accumulator stack using the Divide Stack instruction. The Out Double instruction copies the value in the accumulator to V1500 and V1501.



Add Binary Top of Stack (ADDBS)



430 440 450

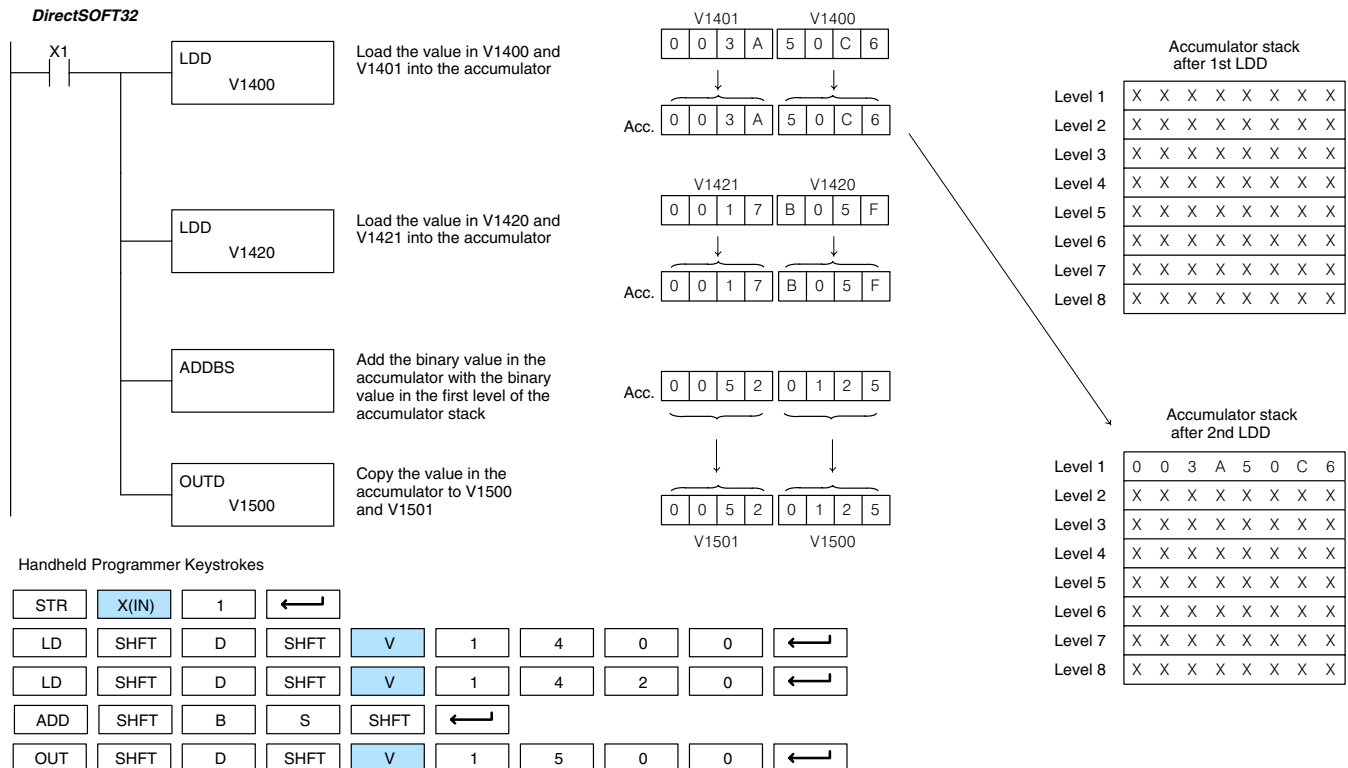
Add Binary Top of Stack instruction is a 32 bit instruction that adds the binary value in the accumulator with the binary value in the first level of the accumulator stack. The result resides in the accumulator. The value in the first level of the accumulator stack is removed and all stack values are moved up one level.

ADDBS

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP66	On when the 16 bit addition instruction results in a carry.
SP67	On when the 32 bit addition instruction results in a carry.
SP70	On anytime the value in the accumulator is negative.
SP73	on when a signed addition or subtraction results in a incorrect sign bit.

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

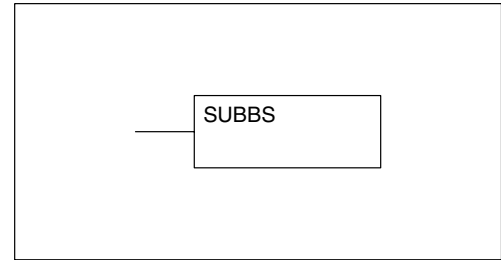
In the following example, when X1 is on, the value in V1400 and V1401 will be loaded into the accumulator using the Load Double instruction. The value in V1420 and V1421 is loaded into the accumulator using the Load Double instruction, pushing the value previously loaded in the accumulator onto the accumulator stack. The binary value in the first level of the accumulator stack is added with the binary value in the accumulator using the Add Stack instruction. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.



Subtract Binary Top of Stack (SUBBS)

✗ ✓ ✓
430 440 450

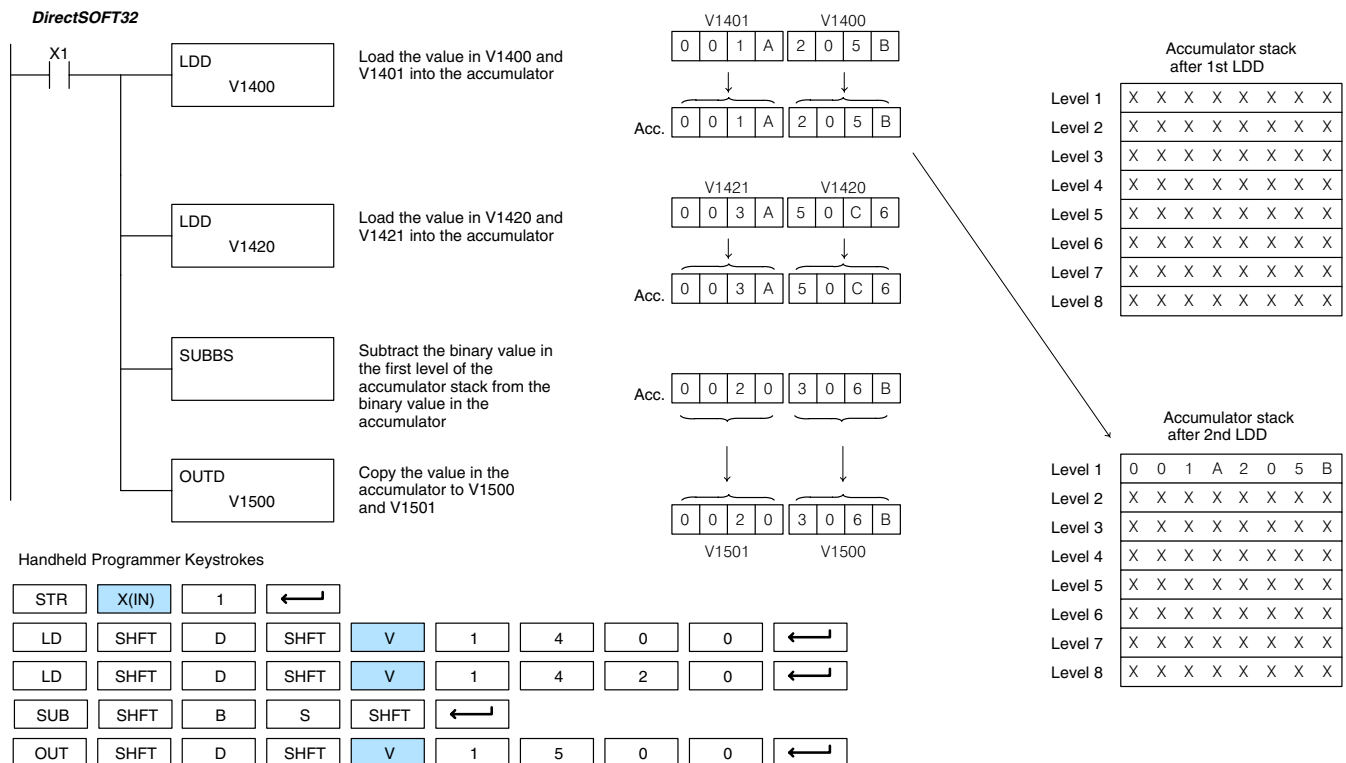
Subtract Binary Top of Stack is a 32 bit instruction that subtracts the binary value in the first level of the accumulator stack from the binary value in the accumulator. The result resides in the accumulator. The value in the first level of the accumulator stack is removed and all stack locations are moved up one level.



Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP64	On when the 16 bit subtraction instruction results in a borrow.
SP65	On when the 32 bit subtraction instruction results in a borrow.
SP70	On anytime the value in the accumulator is negative.

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

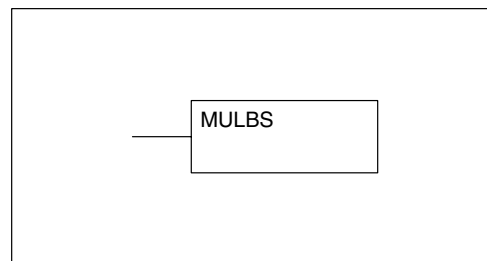
In the following example, when X1 is on, the value in V1400 and V1401 will be loaded into the accumulator using the Load Double instruction. The value in V1420 and V1421 is loaded into the accumulator using the Load Double instruction, pushing the value previously loaded in the accumulator onto the accumulator stack. The binary value in the first level of the accumulator stack is subtracted from the binary value in the accumulator using the Subtract Stack instruction. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.



Multiply Binary Top of Stack (MULBS)

✗ ✓ ✓
430 440 450

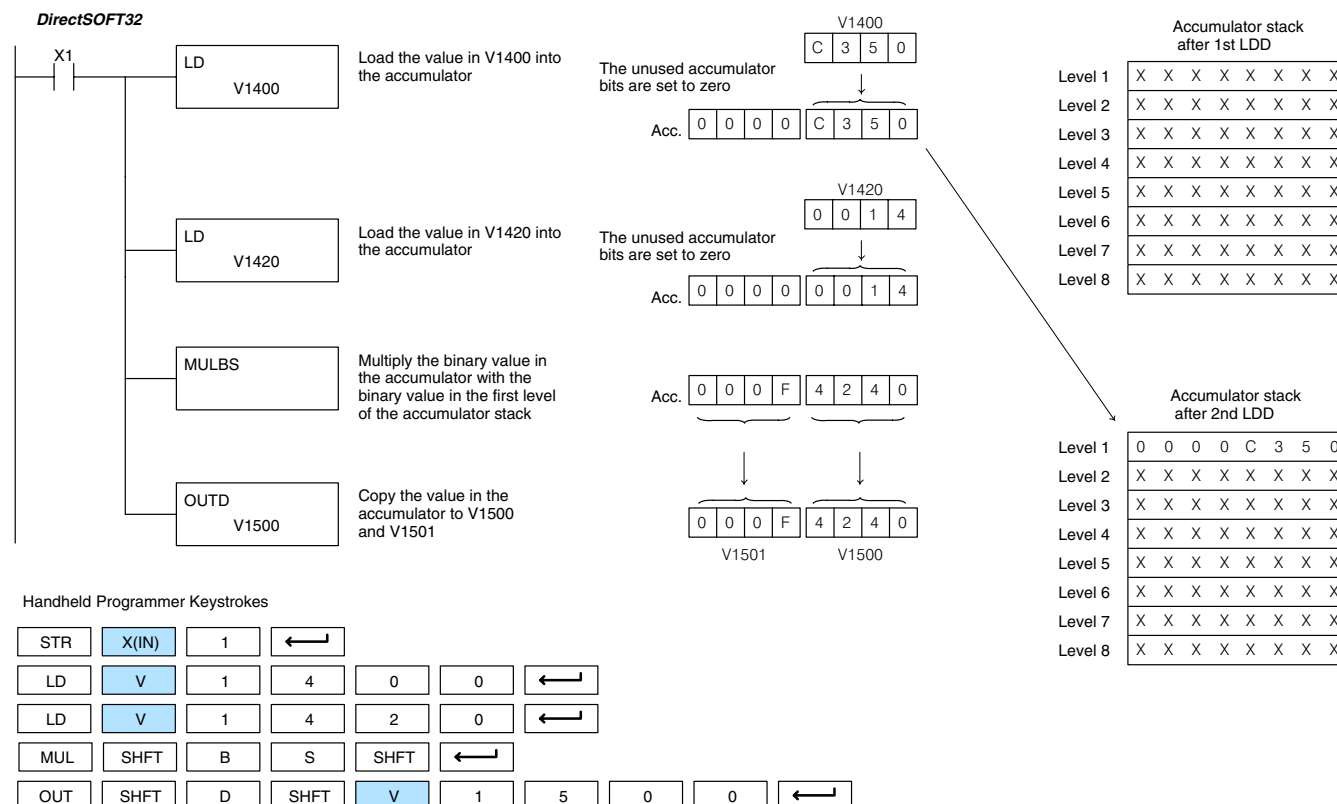
Multiply Binary Top of Stack is a 16 bit instruction that multiplies the 16 bit binary value in the first level of the accumulator stack by the 16 bit binary value in the accumulator. The result resides in the accumulator and can be 32 bits (8 digits max.). The value in the first level of the accumulator stack is removed and all stack locations are moved up one level.



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

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

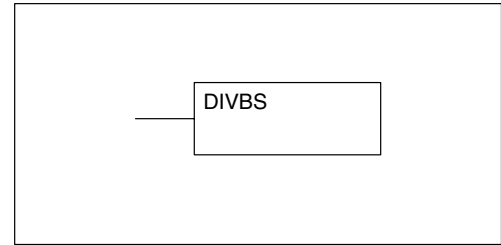
In the following example, when X1 is on, the Load instruction moves the value in V1400 into the accumulator. The value in V1420 is loaded into the accumulator, pushing the value previously loaded in the accumulator onto the stack. The binary value in the accumulator stack's first level is multiplied by the binary value in the accumulator using the Multiply Binary Stack instruction. The Out Double instruction copies the value in the accumulator to V1500 and V1501.



Divide Binary by Top OF Stack (DIVBS)

✕ ✓ ✓
430 440 450

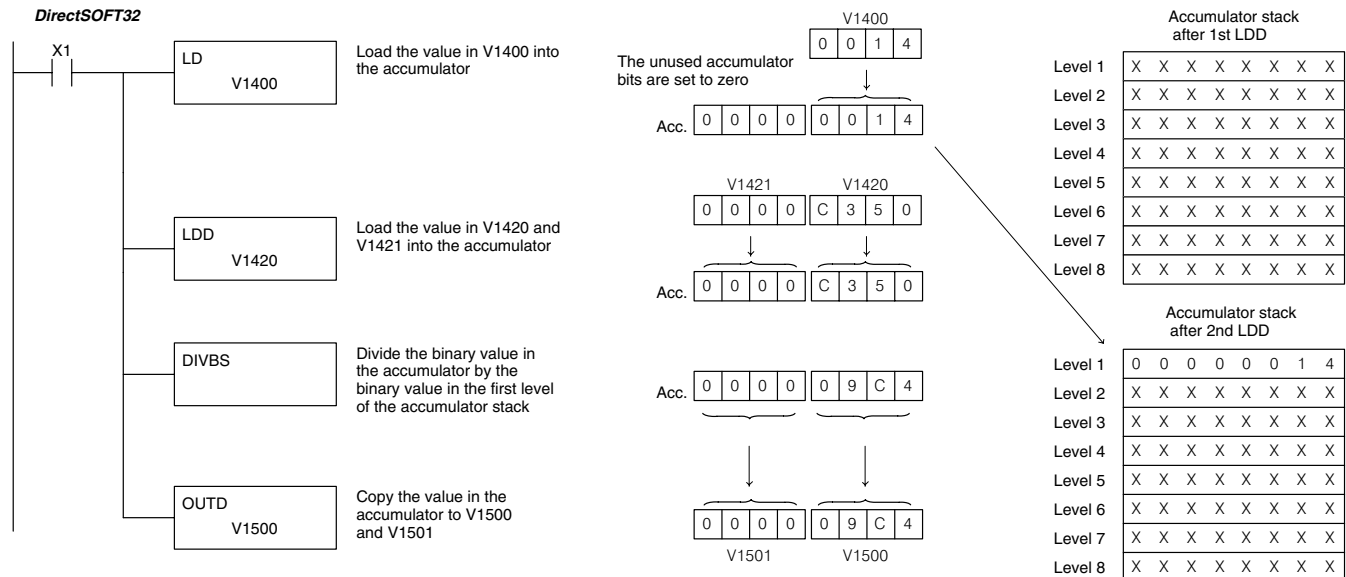
Divide Binary Top of Stack is a 32 bit instruction that divides the 32 bit binary value in the accumulator by the 16 bit binary value in the first level of the accumulator stack. The result resides in the accumulator and the remainder resides in the first level of the accumulator stack.



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

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

In the following example, when X1 is on, the value in V1400 will be loaded into the accumulator using the Load instruction. The value in V1420 and V1421 is loaded into the accumulator using the Load Double instruction also, pushing the value previously loaded in the accumulator onto the accumulator stack. The binary value in the accumulator is divided by the binary value in the first level of the accumulator stack using the Divide Binary Stack instruction. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.



Handheld Programmer Keystrokes

STR	X(IN)	1	←
LD	V	1	4 0 0 ←
LD	SHFT	D	SHFT V 1 4 2 0 ←
DIV	SHFT	B	S SHFT ←
OUT	SHFT	D	SHFT V 1 5 0 0 ←

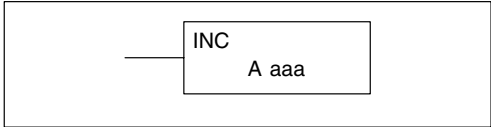
The remainder resides in the first stack location

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

Increment
(INC)

✓ ✓ ✓
430 440 450

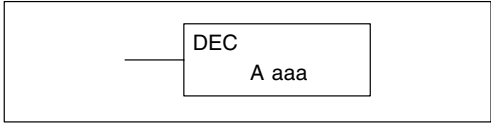
The Increment instruction increments a BCD value in a specified V memory location by “1” each time the instruction is executed.



Decrement
(DEC)

✓ ✓ ✓
430 440 450

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

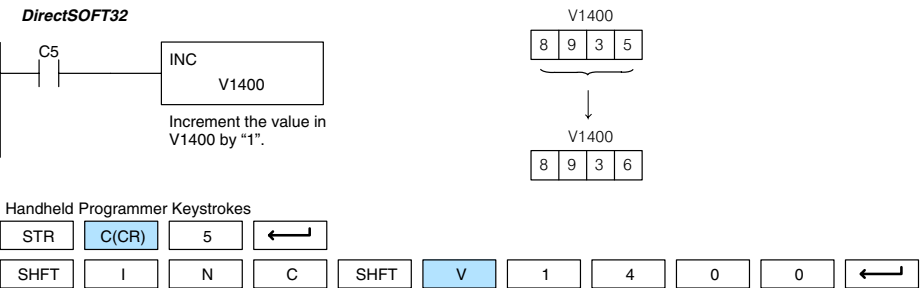


Operand Data Type		DL430 Range	DL440 Range	DL450 Range
A		aaa	aaa	aaa
Vmemory	V	All (See p. 3-40)	All (See p. 3-41)	All (See p. 3-42)
Pointer	P	All (See p. 3-40)	All (See p. 3-41)	All (See p. 3-42)

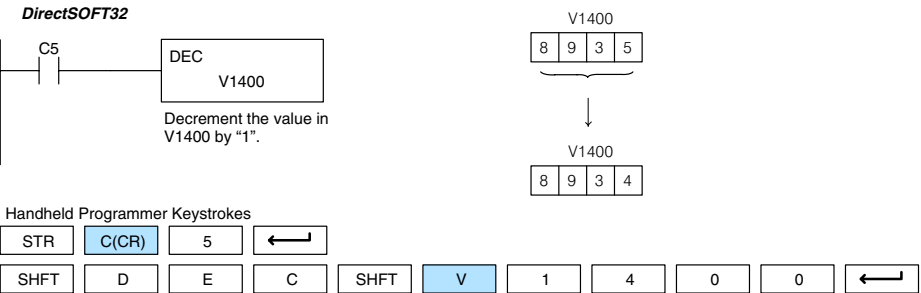
Discrete Bit Flags	Description
SP63	on when the result of the instruction causes the value in the accumulator to be zero.
SP75	on when a BCD instruction is executed and a NON-BCD number was encountered.

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

In the following increment example, when C5 is on the value in V1400 increases by one.



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



Transcendental Functions

The DL450 CPU features special numerical functions to complement its real number capability. The transcendental functions include the trigonometric sine, cosine, and tangent, and also their inverses (arc sine, arc cosine, and arc tangent). The square root function is also grouped with these other functions.

The transcendental math instructions operate on a real number in the accumulator (it cannot be BCD or binary). The real number result resides in the accumulator. The square root function operates on the full range of positive real numbers. The sine, cosine and tangent functions require numbers expressed in radians. You can work with angles expressed in degrees by first converting them to radians with the Radian (RAD) instruction, then performing the trig function. All transcendental functions utilize the following flag bits.

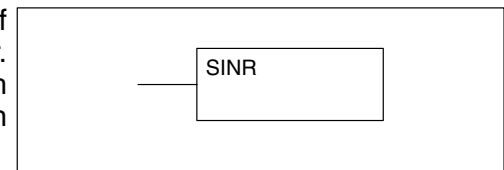
Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.
SP72	On anytime the value in the accumulator is a valid floating point number.
SP73	on when a signed addition or subtraction results in a incorrect sign bit.
SP75	On when a real number instruction is executed and a non-real number was encountered.

Math Function	Range of Argument
SP53	On when the value of the operand is larger than the accumulator can work with.

Sine Real (SINR)



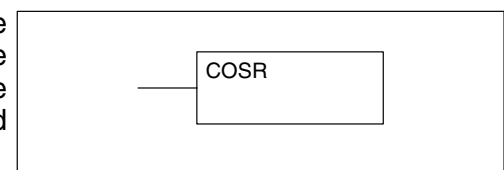
The Sine Real instruction takes the sine of the real number stored in the accumulator. The result resides in the accumulator. Both the original number and the result are in IEEE 32-bit format.



Cosine Real (COSR)



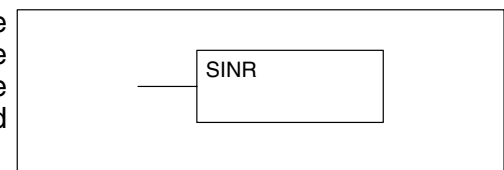
The Cosine Real instruction takes the cosine of the real number stored in the accumulator. The result resides in the accumulator. Both the original number and the result are in IEEE 32-bit format.



Tangent Real (TANR)



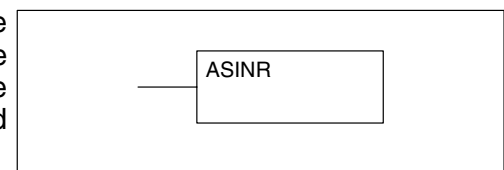
The Tangent Real instruction takes the tangent of the real number stored in the accumulator. The result resides in the accumulator. Both the original number and the result are in IEEE 32-bit format.



Arc Sine Real (ASINR)



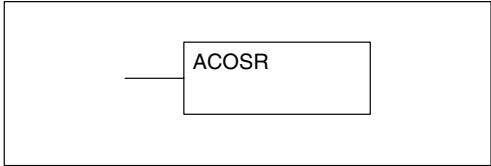
The Arc Sine Real instruction takes the inverse sine of the real number stored in the accumulator. The result resides in the accumulator. Both the original number and the result are in IEEE 32-bit format.



Arc Cosine Real
(ACOSR)

✕ ✕ ✓
430 440 450

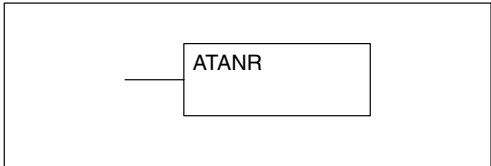
The Arc Cosine Real instruction takes the inverse cosine of the real number stored in the accumulator. The result resides in the accumulator. Both the original number and the result are in IEEE 32-bit format.



Arc Tangent Real
(ATANR)

✕ ✕ ✓
430 440 450

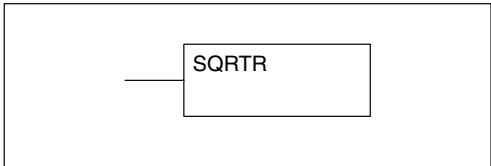
The Arc Tangent Real instruction takes the inverse tangent of the real number stored in the accumulator. The result resides in the accumulator. Both the original number and the result are in IEEE 32-bit format.



Square Root Real
(SQRTR)

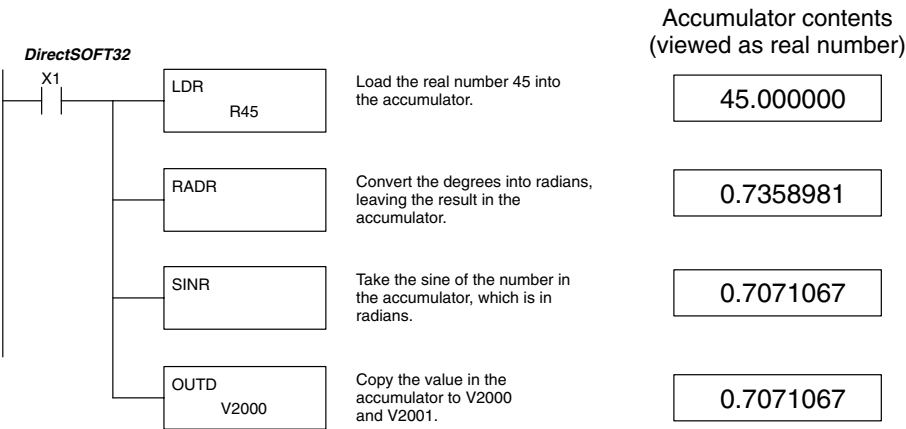
✕ ✕ ✓
430 440 450

The Square Root Real instruction takes the square root of the real number stored in the accumulator. The result resides in the accumulator. Both the original number and the result are in IEEE 32-bit format.



NOTE: The square root function can be useful in several situations. However, if you are trying to do the square-root extract function for an orifice flow meter measurement as the PV to a PID loop, note that the PID loop already has the square-root extract function built in.

The following example takes the **sine** of 45 degrees. Since these transcendental functions operate only on real numbers, we do a LDR (load real) 45. The trig functions operate only in radians, so we must convert the degrees to radians by using the RADR command. After using the SINR (Sine Real) instruction, we use an OUTD (Out Double) instruction to move the result from the accumulator to V-memory. The result is 32-bits wide, requiring the Out Double to move it.

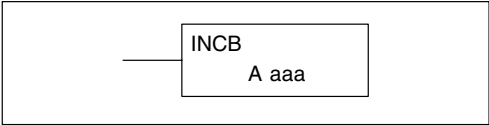


NOTE: The current HPP does not support real number entry with automatic conversion to the 32-bit IEEE format. You must use **DirectSOFT32** for entering real numbers, using the LDR (Load Real) instruction.

Increment Binary
(INCB)

✓ ✓ ✓
430 440 450

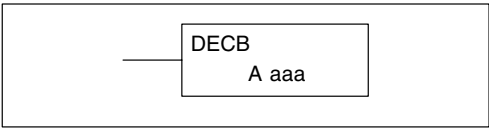
The Increment Binary instruction increments a binary value in a specified V memory location by “1” each time the instruction is executed.



Decrement Binary
(DECB)

✓ ✓ ✓
430 440 450

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

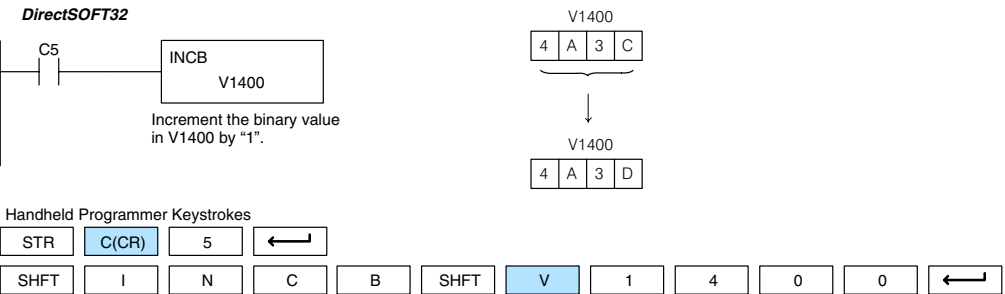


Operand Data Type		DL430 Range	DL440 Range	DL450 Range
A		aaa	aaa	aaa
Vmemory	V	All (See p. 3-40)	All (See p. 3-40)	All (See p. 3-42)
Pointer	P	All (See p. 3-40)	All (See p. 3-41)	All (See p. 3-42)

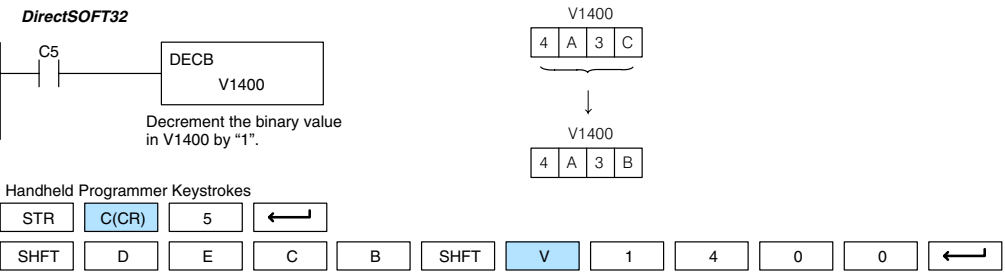
Discrete Bit Flags	Description
SP63	on when the result of the instruction causes the value in the accumulator to be zero.

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

In the following increment binary example, when C5 is on the binary value in V1400 is increased by one.



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

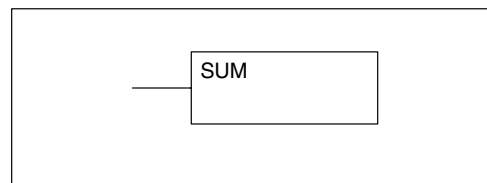


Bit Operation Instructions

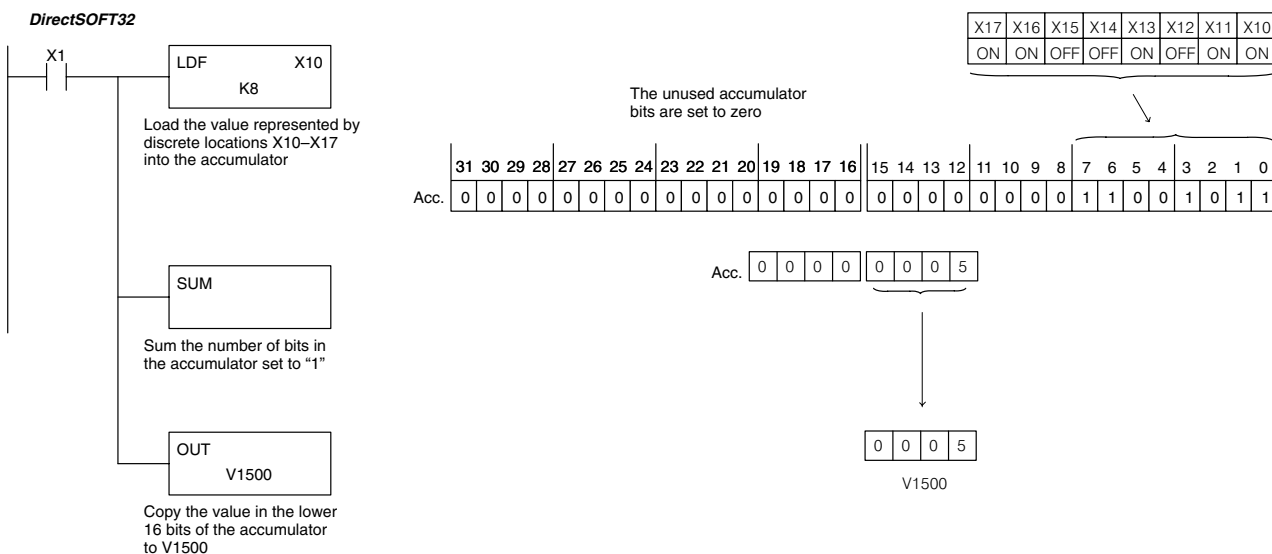
Sum (SUM)

✓ ✓ ✓
430 440 450

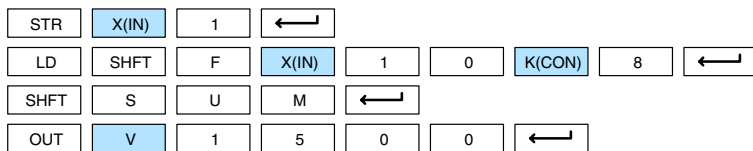
The Sum instruction counts number of bits that are set to “1” in the accumulator. The HEX result resides in the accumulator.



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.



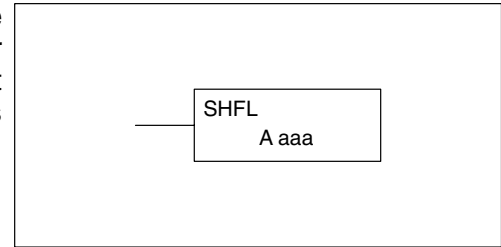
Handheld Programmer Keystrokes



Shift Left (SHFL)

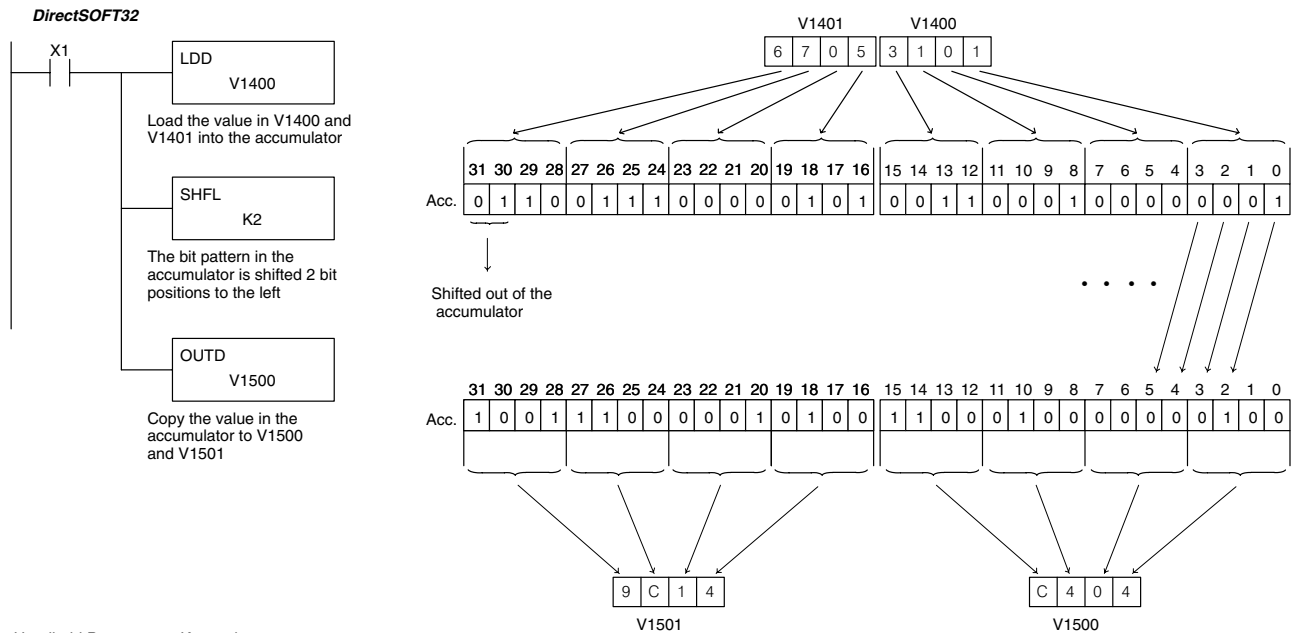
✓ ✓ ✓
430 440 450

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		DL430 Range	DL440 Range	DL450 Range
A		aaa	aaa	aaa
Vmemory	V	All (See p. 3-40)	All (See p. 3-41)	All (See p. 3-42)
Constant	K	1-32	1-32	1-32

In the following example, when X1 is on, the value in V1400 and V1401 will be loaded into the accumulator using the Load Double instruction. The bit pattern in the accumulator is shifted 2 bits to the left using the Shift Left instruction. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.



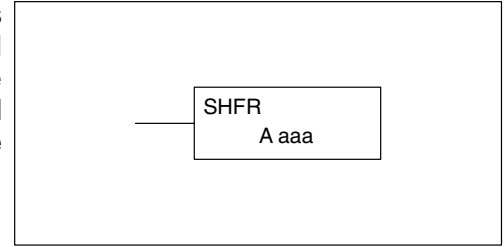
Handheld Programmer Keystrokes

STR	X(IN)	1	←						
LD	SHFT	D	SHFT	V	1	4	0	0	←
SHFT	S	H	F	L	SHFT	K(CON)	2	←	
OUT	SHFT	D	SHFT	V	1	5	0	0	←

Shift Right (SHFR)

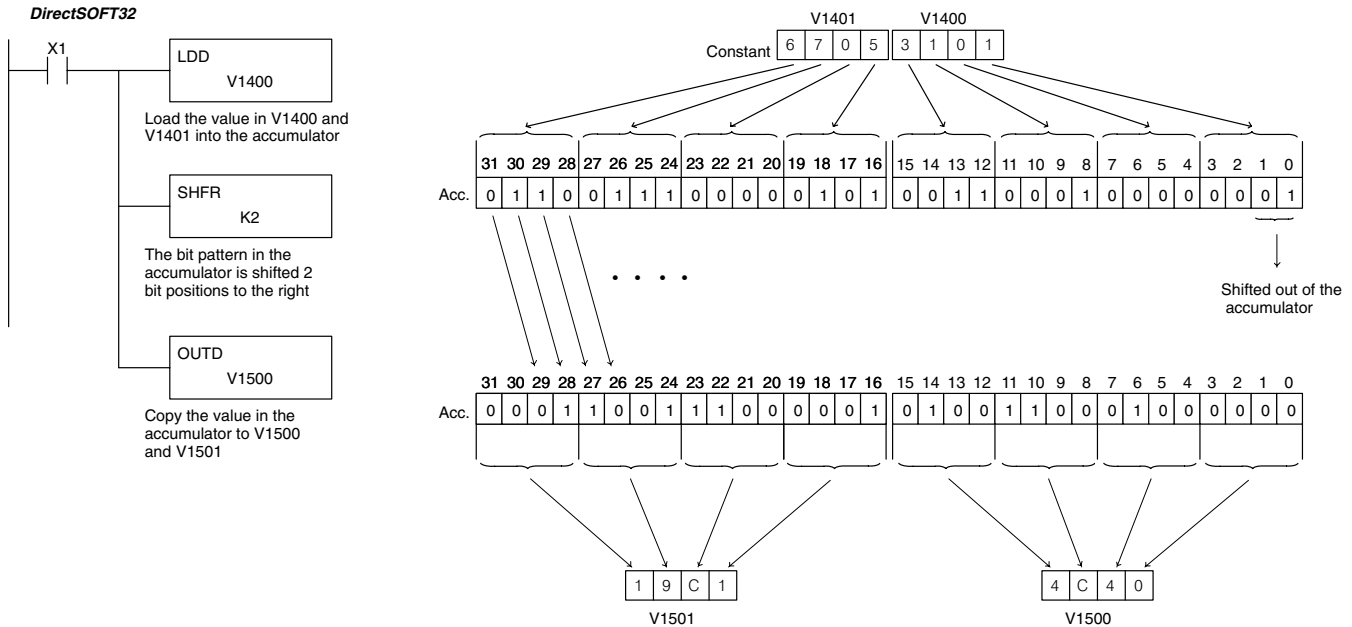
✓ ✓ ✓
430 440 450

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		DL430 Range	DL440 Range	DL450 Range
A		aaa	aaa	aaa
Vmemory	V	All (See p. 3-40)	All (See p. 3-41)	All (See p. 3-42)
Constant	K	1-32	1-32	1-32

In the following example, when X1 is on, the value in V1400 and V1401 will be loaded into the accumulator using the Load Double instruction. The bit pattern in the accumulator is shifted 2 bits to the right using the Shift Right instruction. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.



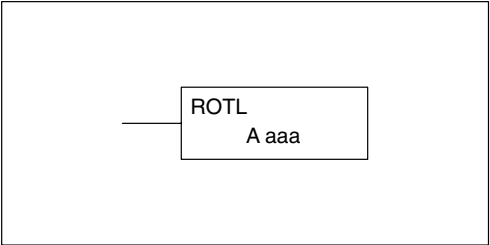
Handheld Programmer Keystrokes

STR	X(IN)	1	←						
LD	SHFT	D	SHFT	V	1	4	0	0	←
SHFT	S	H	F	R	SHFT	K(CON)	2	←	
OUT	SHFT	D	SHFT	V	1	5	0	0	←

Rotate Left
(ROTL)

✓✓✓
430 440 450

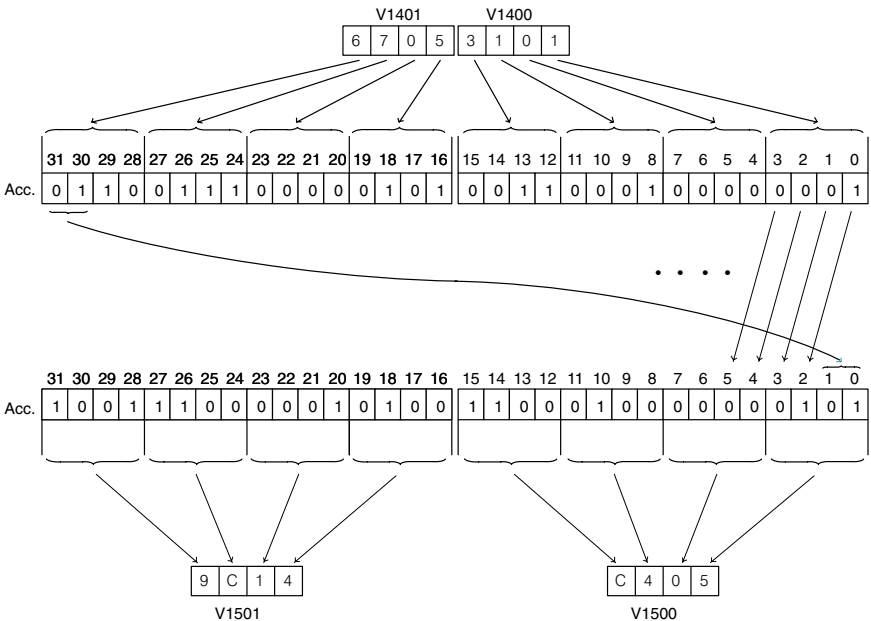
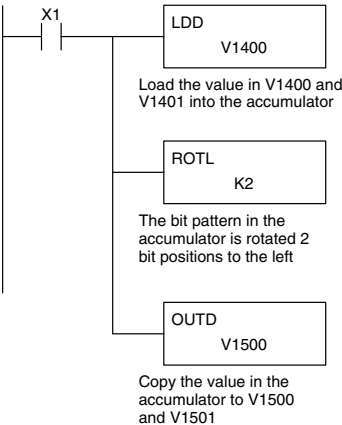
Rotate Left is a 32 bit instruction that rotates the bits in the accumulator a specified number (Aaaa) of places to the left.



Operand Data Type		DL430 Range	DL440 Range	DL450 Range
A		aaa	aaa	aaa
Vmemory	V	All (See p. 3-40)	All (See p. 3-41)	All (See p. 3-42)
Constant	K	1-32	1-32	1-32

In the following example, when X1 is on, the value in V1400 and V1401 will be loaded into the accumulator using the Load Double instruction. The bit pattern in the accumulator is rotated 2 bit positions to the left using the Rotate Left instruction. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.

DirectSOFT32



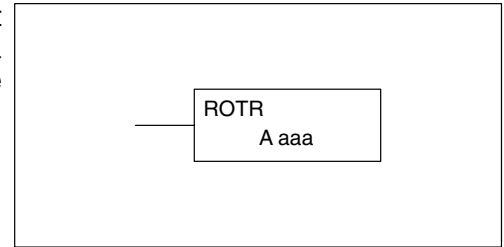
Handheld Programmer Keystrokes

STR	X(IN)	1	↩							
LD	SHFT	D	SHFT	V	1	4	0	0	↩	
SHFT	R	O	T	L	SHFT	K(CON)	2	↩		
OUT	SHFT	D	SHFT	V	1	5	0	0	↩	

Rotate Right (ROTR)

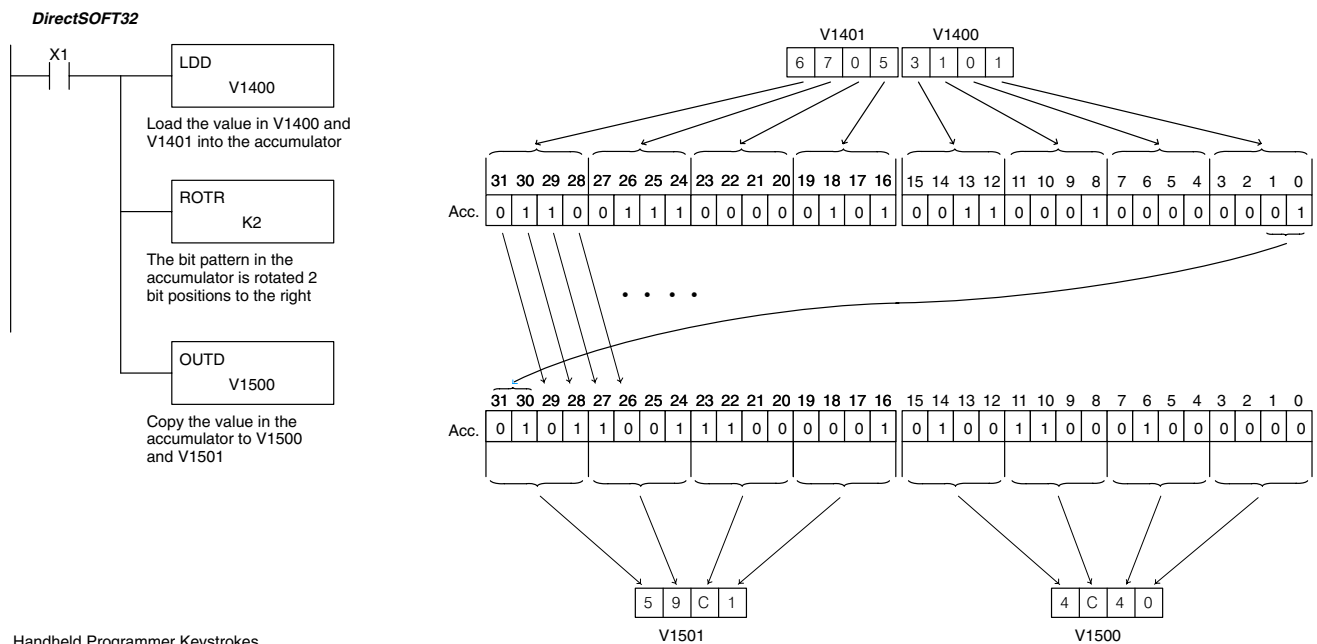
✓ ✓ ✓
430 440 450

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		DL430 Range	DL440 Range	DL450 Range
A		aaa	aaa	aaa
Vmemory	V	All (See p. 3-40)	All (See p. 3-41)	All (See p. 3-42)
Constant	K	1-32	1-32	1-32

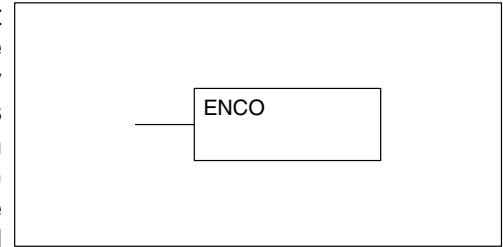
In the following example, when X1 is on, the value in V1400 and V1401 will be loaded into the accumulator using the Load Double instruction. The bit pattern in the accumulator is rotated 2 bit positions to the right using the Rotate Right instruction. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.



Encode (ENCO)

✓ ✓ ✓
430 440 450

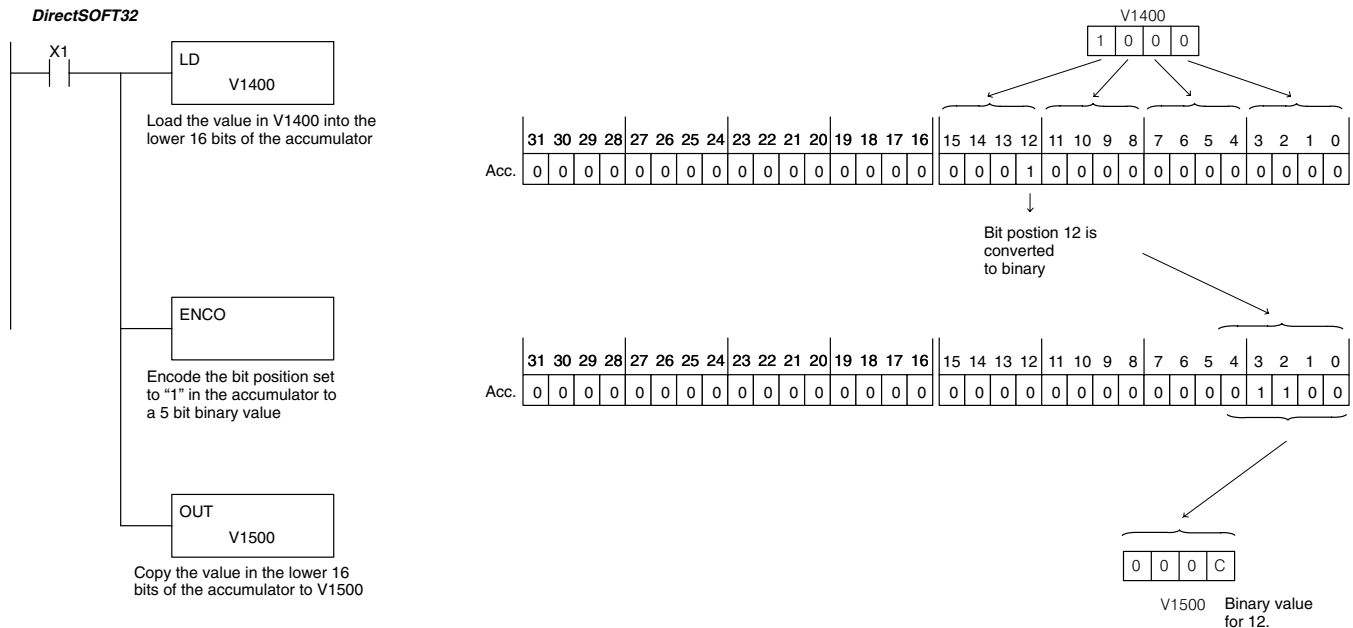
The Encode instruction encodes the bit position in the accumulator having a value of 1, and returns the appropriate binary representation. If the most significant bit is set to 1 (Bit 31), the Encode instruction would place the value HEX 1F (decimal 31) in the accumulator. If the value to be encoded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



Discrete Bit Flags	Description
SP53	On when the value of the operand is larger than the accumulator can work with. SP53 will also come on when the accumulator is zero.

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

In the following example, when X1 is on, The value in V1400 is loaded into the accumulator using the Load instruction. The bit position set to a “1” in the accumulator is encoded to the corresponding 5 bit binary value using the Encode instruction. The value in the lower 16 bits of the accumulator is copied to V1500 using the Out instruction.



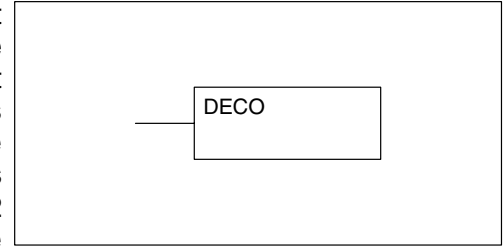
Handheld Programmer Keystrokes

STR	X(IN)	1	←
LD	V	1	4 0 0 ←
SHFT	E	N	C O ←
OUT	V	1	5 0 0 ←

Decode (DECO)

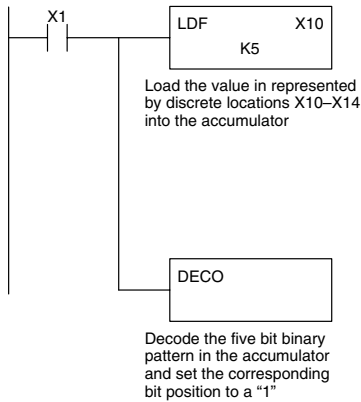
✓ ✓ ✓
430 440 450

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.

DirectSOFT32



X14	X13	X12	X11	X10
OFF	ON	OFF	ON	ON

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

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

The binary value is converted to bit position 11.

Handheld Programmer Keystrokes

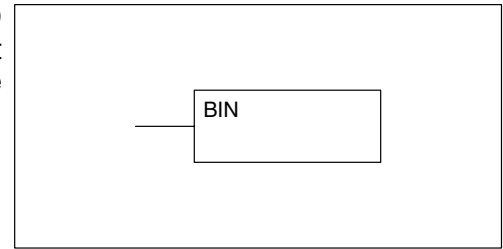
STR	X(IN)	1	←							
LD	SHFT	F	SHFT	X(IN)	1	0	SHFT	K(CON)	5	←
SHFT	D	E	C	O	←					

Number Conversion Instructions

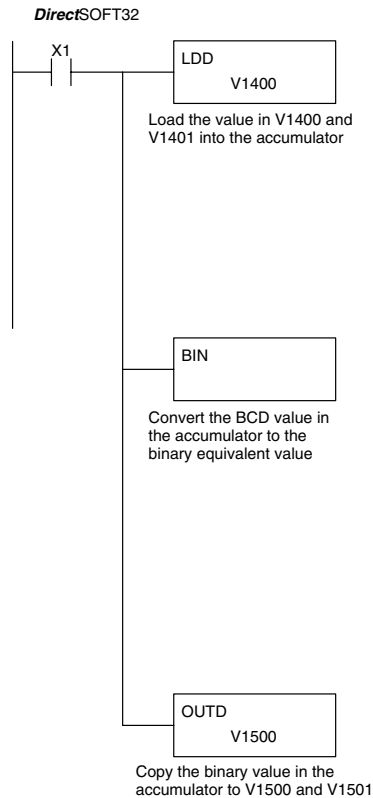
Binary (BIN)

✓ ✓ ✓
430 440 450

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 V1400 and V1401 is loaded into the accumulator using the Load Double instruction. The BCD value in the accumulator is converted to the binary (HEX) equivalent using the BIN instruction. The binary value in the accumulator is copied to V1500 and V1501 using the Out Double instruction. The handheld programmer would display the binary value in V1500 and V1501 as a HEX value.



V1401				V1400			
0	0	0	2	8	5	2	9

8 4 2 1	8 4 2 1	8 4 2 1	8 4 2 1	8 4 2 1	8 4 2 1	8 4 2 1	8 4 2 1
0 0 0 0	0 0 0 0	0 0 0 0	0 0 1 0	1 0 0 0	0 1 0 1	0 0 1 0	1 0 0 1

BCD Value

$$28529 = 16384 + 8192 + 2048 + 1024 + 512 + 256 + 64 + 32 + 16 + 1$$

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

V1501				V1500			
0	0	0	0	6	F	7	1

The binary (HEX) value copied to V1500

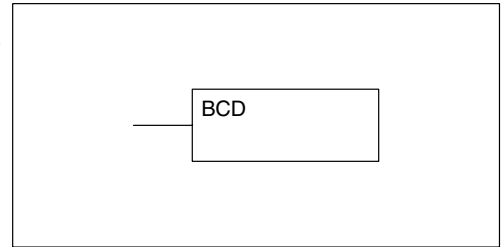
Handheld Programmer Keystrokes

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

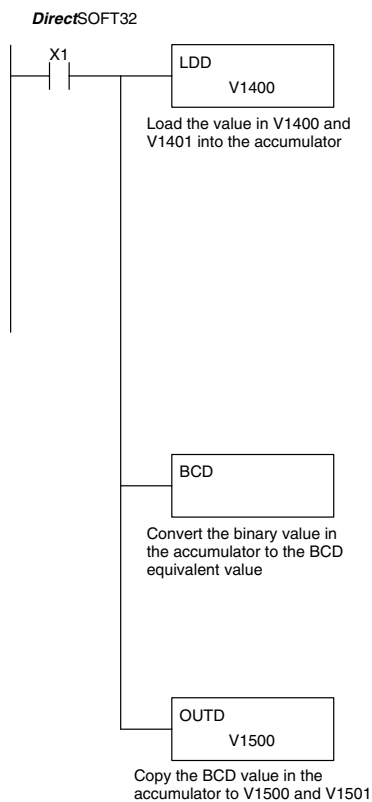
**Binary Coded
Decimal
(BCD)**

430 440 450

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



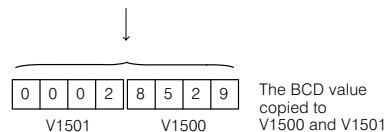
In the following example, when X1 is on, the binary (HEX) value in V1400 and V1401 is loaded into the accumulator using the Load Double instruction. The binary value in the accumulator is converted to the BCD equivalent value using the BCD instruction. The BCD value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.



V1401																V1400															
0 0 0 0																6	F	7	1												
Binary Value																															

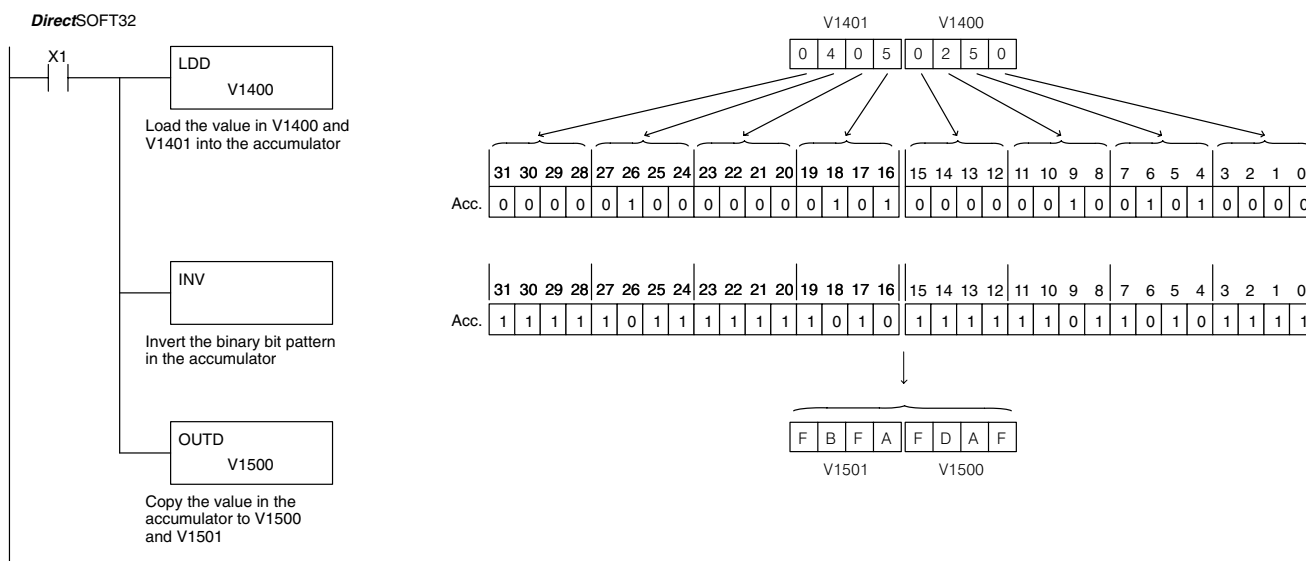
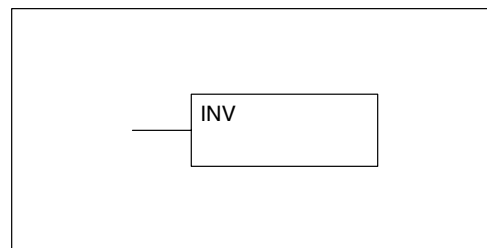
$$16384 + 8192 + 2048 + 1024 + 512 + 256 + 64 + 32 + 16 + 1 = 28529$$

BCD Equivalent Value																																	
	8	4	2	1	8	4	2	1	8	4	2	1	8	4	2	1		8	4	2	1	8	4	2	1	8	4	2	1	8	4	2	1
Acc.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0		1	0	0	0	0	1	0	1	0	0	1	0	1	0	0	1

**Handheld Programmer Keystrokes**

STR	X(IN)	1	←							
LD	SHFT	D	SHFT	V	1	4	0	0	←	
BCD	←									
OUT	SHFT	D	SHFT	V	1	5	0	0	←	

430 440 450



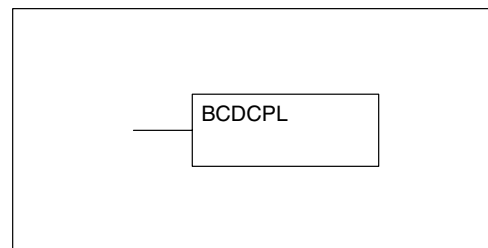
STR	X(IN)	1	←							
LD	SHFT	D	SHFT	V	1	4	0	0	←	
SHFT	I	N	V	←						
OUT	SHFT	D	SHFT	V	1	5	0	0	←	

Ten's Complement (BCDCPL)

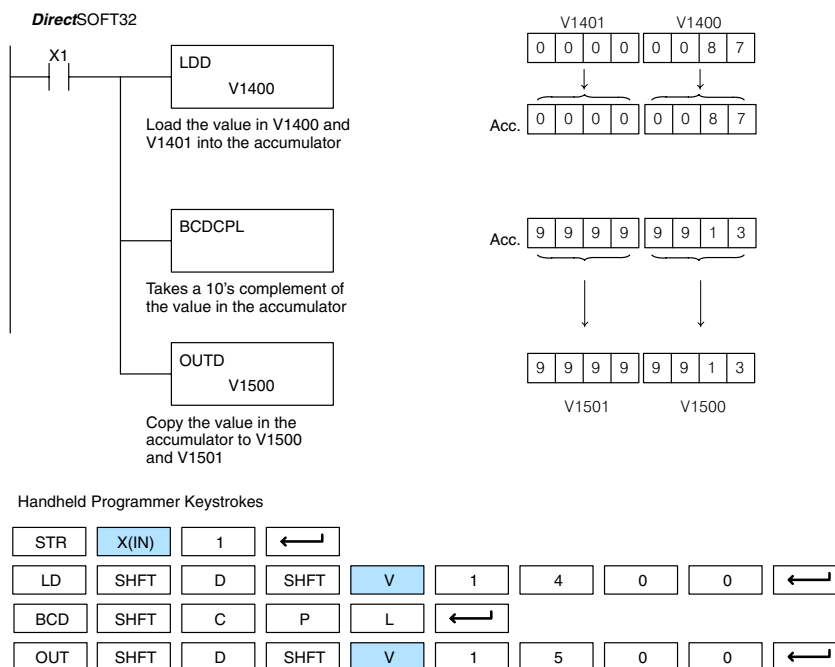
✓ ✓ ✓
430 440 450

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 V1400 and V1401 is loaded into the accumulator. The 10's complement is taken for the 8 digit accumulator using the Ten's Complement instruction. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.

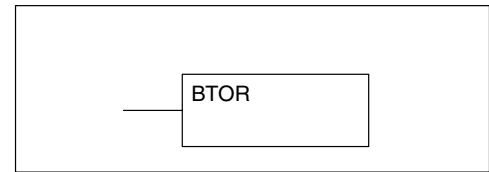


NOTE: If your program has a subtraction calculation which results in a borrowed digit (noted by the status flag), the BCDCPL instruction can be used to find the absolute difference between the two values in the subtraction.

Binary to Real Conversion (BTOR)

✕ ✕ ✓
430 440 450

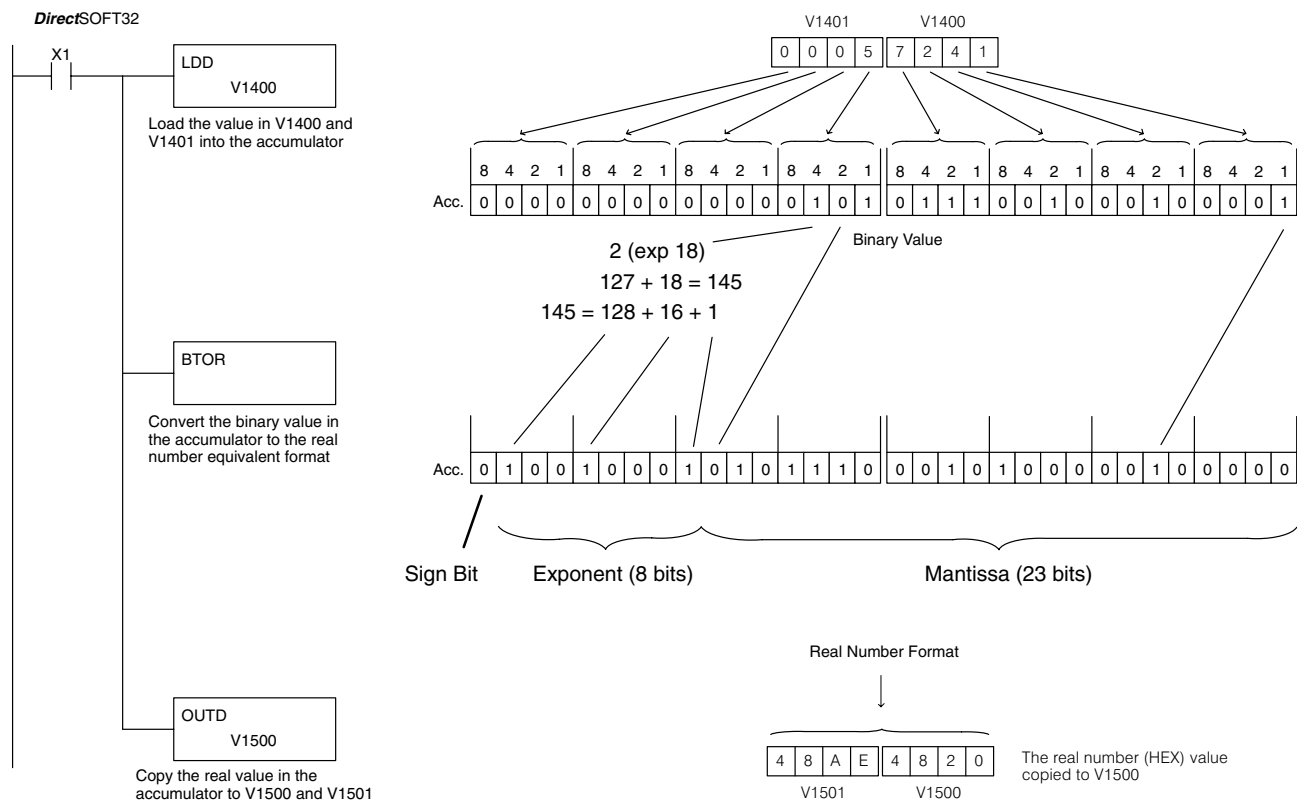
The Binary-to-Real instruction converts a binary value in the accumulator to its equivalent real number (floating point) format. The result resides in the accumulator. Both the binary and the real number may use all 32 bits of the accumulator



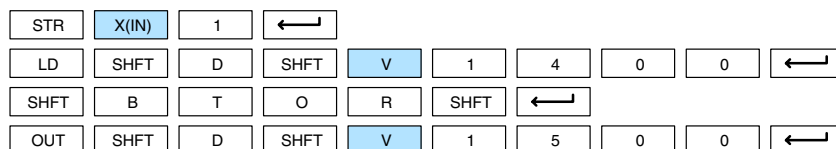
NOTE: This instruction will not work for a signed decimal.

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

In the following example, when X1 is on, the value in V1400 and V1401 is loaded into the accumulator using the Load Double instruction. The BTOR instruction converts the binary value in the accumulator to the equivalent real number format. The binary weight of the MSB is converted to the real number exponent by adding it to 127 (decimal). Then the remaining bits are copied to the mantissa as shown. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction. The handheld programmer would display the binary value in V1500 and V1501 as a HEX value.



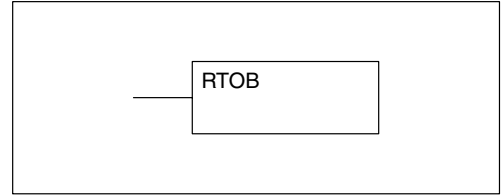
Handheld Programmer Keystrokes



Real to Binary Conversion (RTOB)

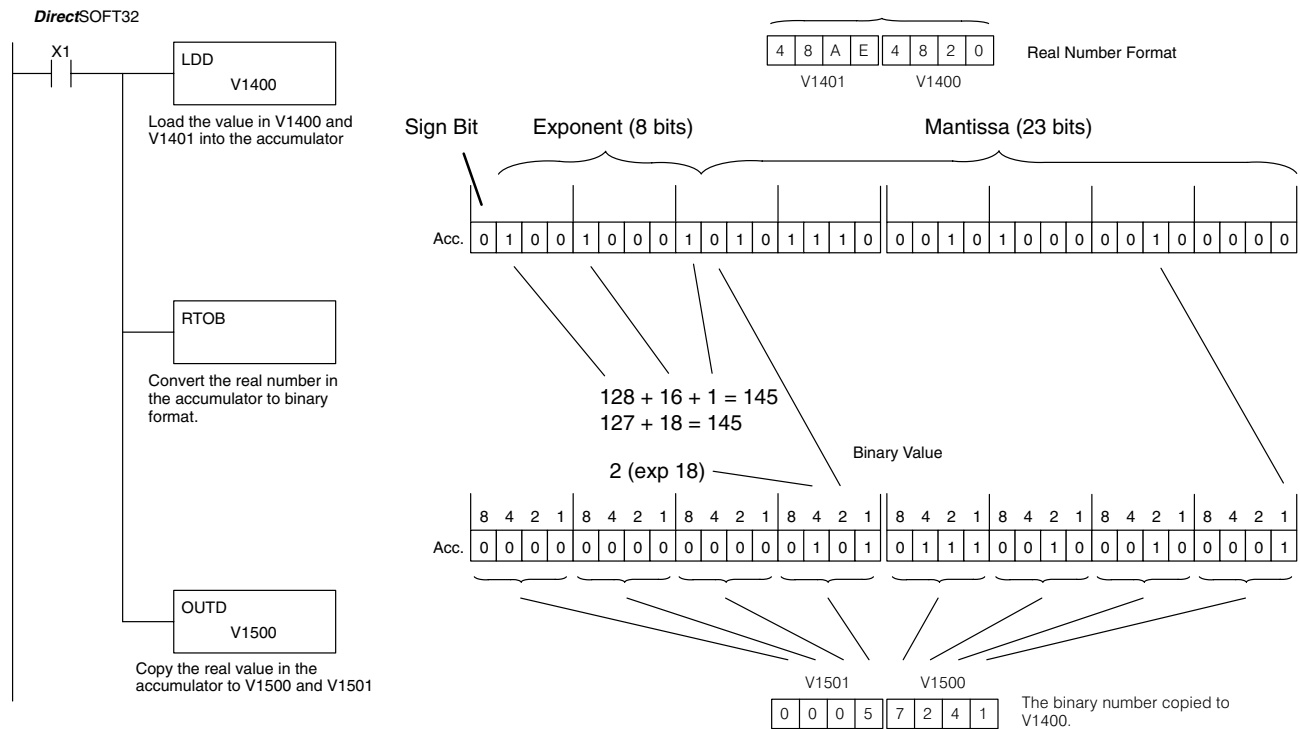
☐ 430
 ☐ 440
 ☒ 450

The Real-to-Binary instruction converts the real number in the accumulator to a binary value. If the real number is negative, it will become a signed decimal. The result resides in the accumulator. Both the binary and the real number may use all 32 bits of the accumulator. Any decimal portion will be truncated.



Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.
SP72	On anytime the value in the accumulator is a valid floating point number.
SP73	on when a signed addition or subtraction results in a incorrect sign bit.
SP75	On when a BCD instruction is executed and a NON-BCD number was encountered.

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



Handheld Programmer Keystrokes

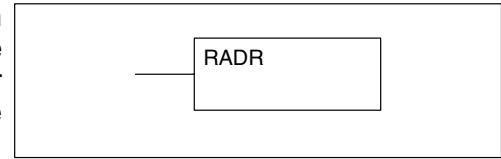
STR	X(IN)	1	←
LD	SHFT	D	SHFT
SHFT	R	T	O
OUT	SHFT	D	SHFT

V 1 4 0 0 ←
 B ←
 V 1 5 0 0 ←

Radian Real Conversion (RADR)

✗ ✗ ✓
430 440 450

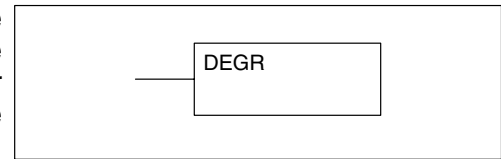
The Radian Real Conversion instruction converts the real degree value stored in the accumulator to the equivalent real number in radians. The result resides in the accumulator.



Degree Real Conversion (DEGR)

✗ ✗ ✓
430 440 450

The Degree Real instruction converts the degree real radian value stored in the accumulator to the equivalent real number in degrees. The result resides in the accumulator.

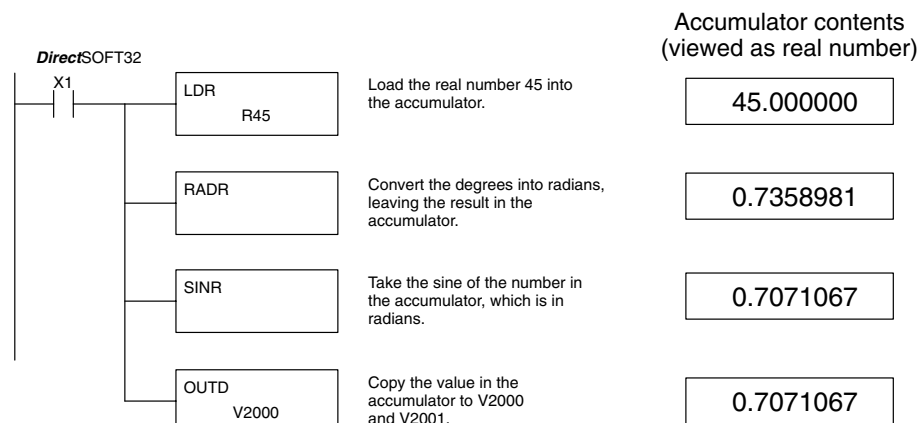


The two instructions described above convert real numbers into the accumulator from degree format to radian format, and visa-versa. In degree format, a circle contains 360 degrees. In radian format, a circle contains 2π radians. These convert between both positive and negative real numbers, and for angles greater than a full circle. These functions are very useful when combined with the transcendental trigonometric functions (see the section on math instructions).

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.
SP71	On anytime the V-memory specified by a pointer (P) is not valid.
SP72	On anytime the value in the accumulator is a valid floating point number.
SP74	On anytime a floating point math operation results in an underflow error.
SP75	On when a BCD instruction is executed and a NON-BCD number was encountered.

NOTE: The current HPP does not support real number entry with automatic conversion to the 32-bit IEEE format. You must use **DirectSOFT32** for entering real numbers, using the LDR (Load Real) instruction.

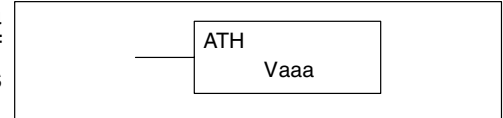
The following example takes the sine of 45 degrees. Since transcendental functions operate only on real numbers, we do a LDR (load real) 45. The trig functions operate only in radians, so we must convert the degrees to radians by using the RADR command. After using the SINR (Sine Real) instruction, we use an OUTD (Out Double) instruction to move the result from the accumulator to V-memory. The result is 32-bits wide, requiring the Out Double to move it.



ASCII to HEX (ATH)

☐ ☒ ☒
 430 440 450

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.

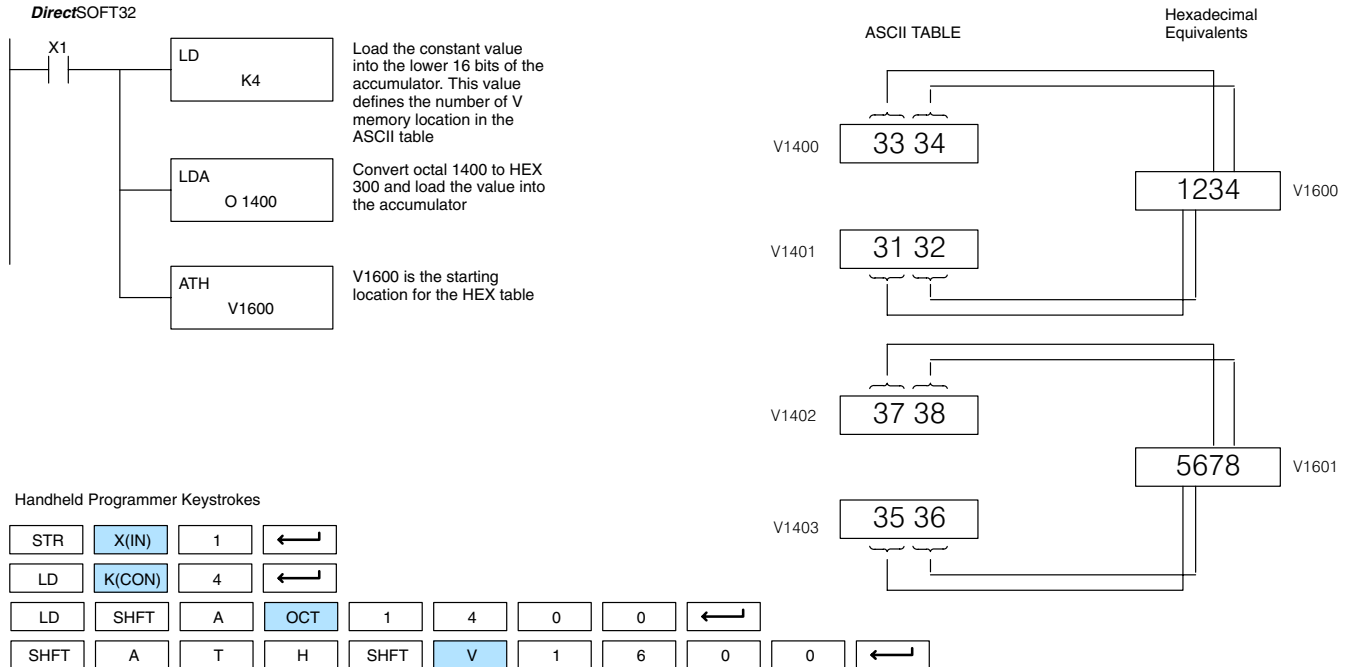
Step 3: — Specify the starting V memory location (Vaaa) for the HEX table in the ATH instruction.

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	DL440 Range	DL450 Range
	aaa	aaa
Vmemory V	All (See p. 3-41)	All (See p. 3-42)

In the example on the following page, when X1 is ON the constant (K4) is loaded into the accumulator using the Load instruction and will be placed in the first level of the accumulator stack when the next Load instruction is executed. The starting location for the ASCII table (V1400) is loaded into the accumulator using the Load Address instruction. The starting location for the HEX table (V1600) is specified in the ASCII to HEX instruction. The table below lists valid ASCII values for ATH conversion.

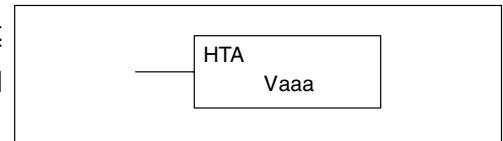
ASCII Values Valid for ATH Conversion			
ASCII Value	Hex Value	ASCII Value	Hex Value
30	0	38	8
31	1	39	9
32	2	41	A
33	3	42	B
34	4	43	C
35	5	44	D
36	6	45	E
37	7	46	F



HEX to ASCII (HTA)

430 440 450

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.

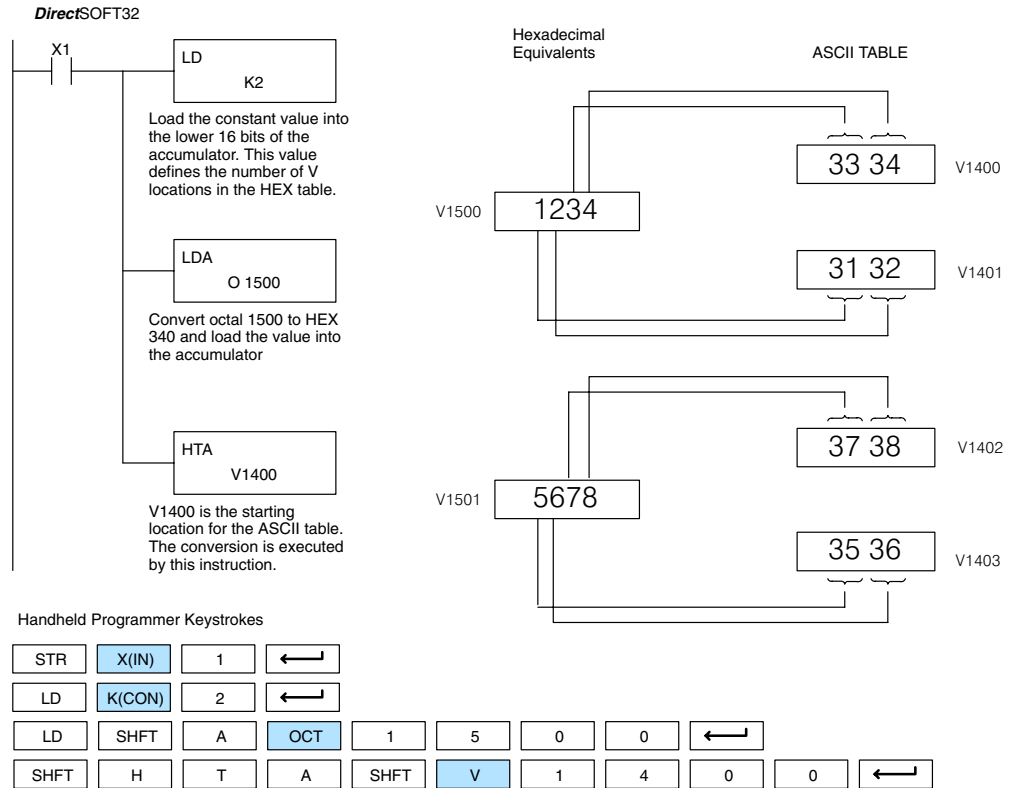
Step 2: — Load the starting V memory location for the HEX table into the accumulator. This parameter must be a HEX value.

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	DL440 Range	DL440 Range
	aaa	aaa
Vmemory V	All (See p. 3-41)	All (See p. 3-42)

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.



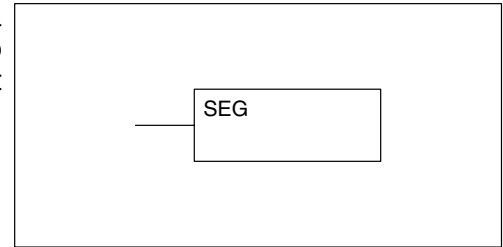
The table below lists valid ASCII values for HTA conversion.

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

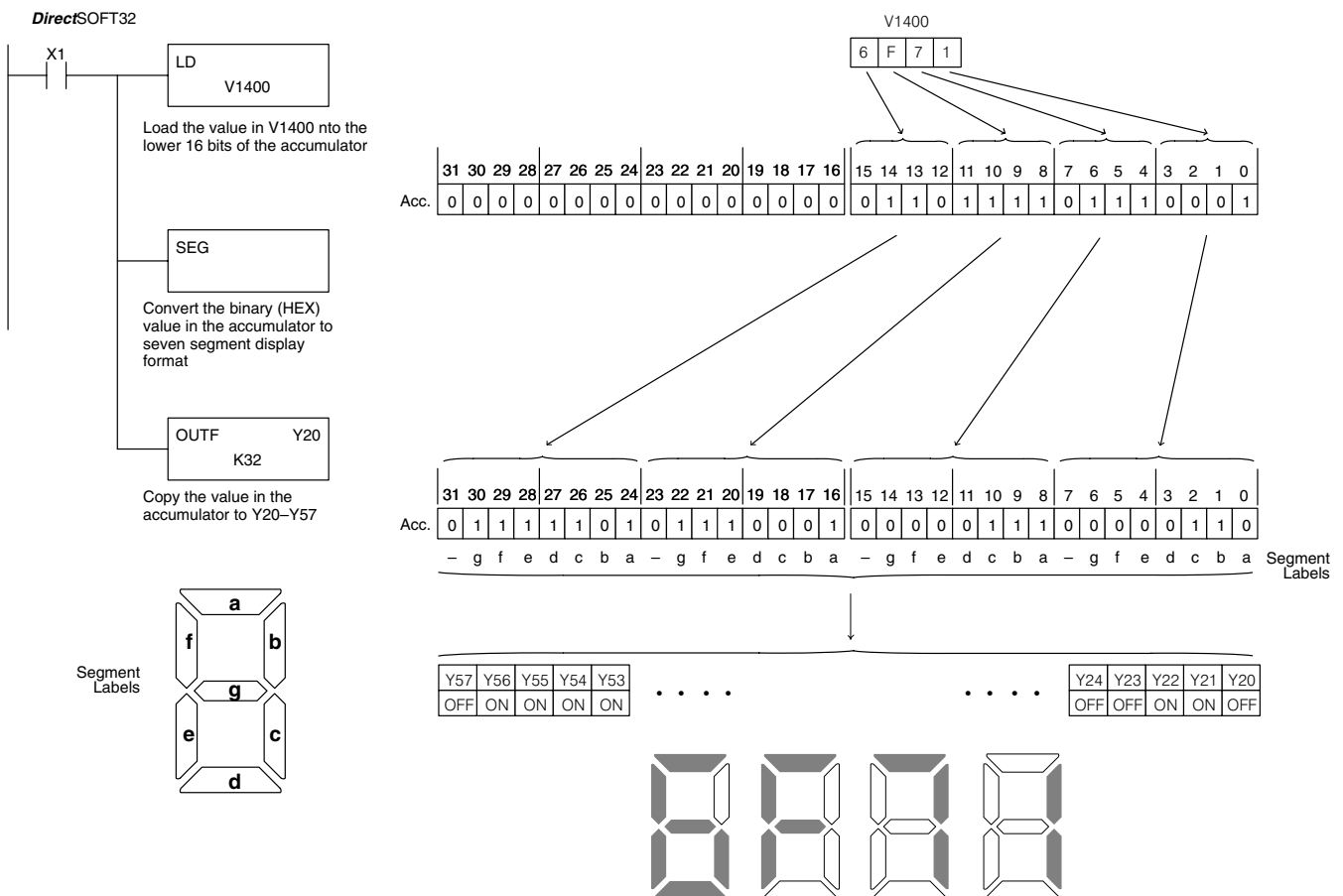
Segment (SEG)

✓ ✓ ✓
430 440 450

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.



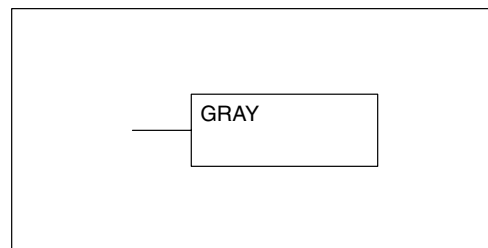
Handheld Programmer Keystrokes

STR	X(IN)	1	←
LD	SHFT	V	1 4 0 0 ←
SHFT	S	E	G ←
OUT	SHFT	F	SHFT Y(OUT) 2 0 K(CON) 3 2 ←

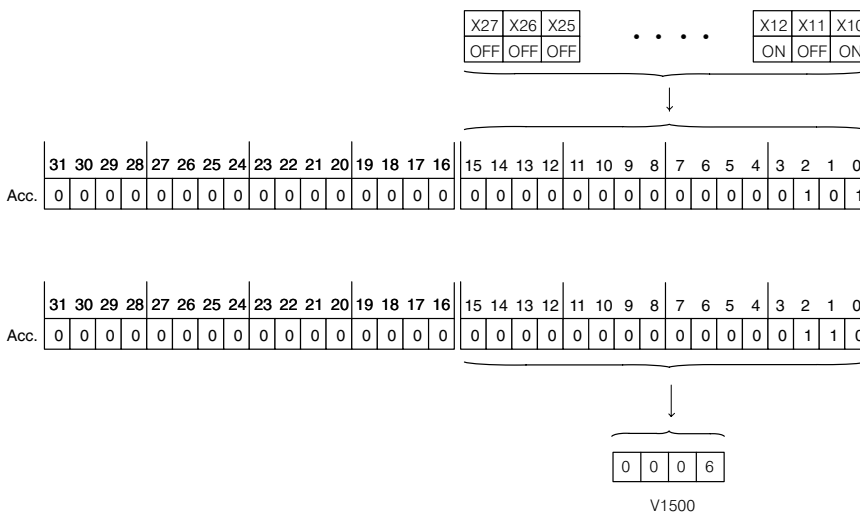
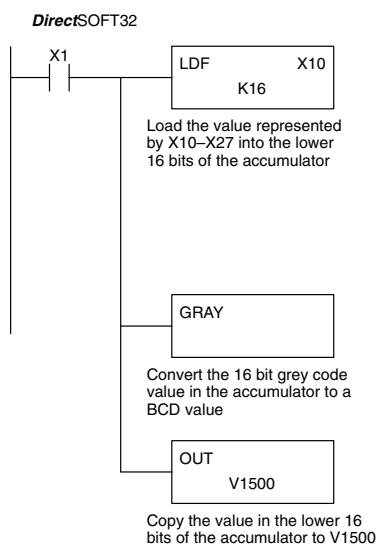
Gray Code (GRAY)

☐ ☒ ☒
 430 440 450

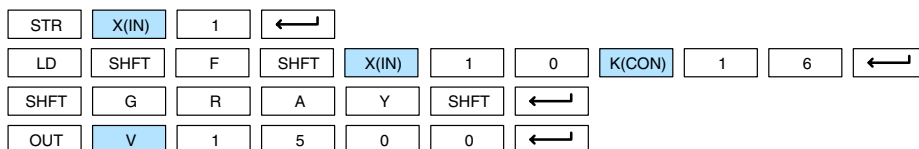
The Gray code instruction converts a 16 bit gray code value to a BCD value. The BCD conversion requires 10 bits of the accumulator. The upper 22 bits are set to "0". This instruction is designed for use with devices (typically encoders) that use the grey code numbering scheme. The Gray Code instruction 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 V1500.



Handheld Programmer Keystrokes

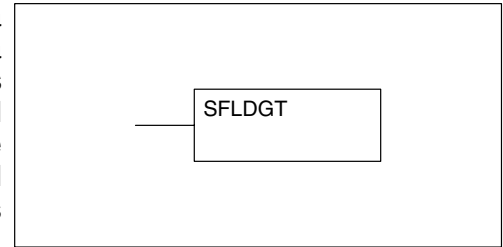


Gray Code	BCD
000000000	0000
000000001	0001
000000011	0002
000000010	0003
000000110	0004
000000111	0005
000000101	0006
000000100	0007
•	•
•	•
•	•
100000001	1022
100000000	1023

Shuffle Digits (SFLDGT)

✕ ✓ ✓
430 440 450

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 the digits will be shuffled to into the accumulator, numbered 1 through 8 (“1” being the least significant digit, and “8” being the most significant digit).

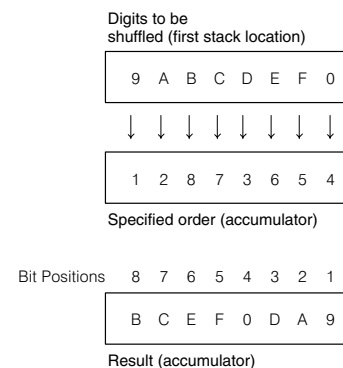
Note:— If the number used to specify the order contains a 0 or 9–F, the corresponding position will be set to 0.
See example

Note:—If the number used to specify the order contains duplicate numbers, the most significant duplicate number is valid. The result resides in the accumulator.
See example

Step 3:— Insert the SFLDGT instruction.

Shuffle Digits Block Diagram

There are a maximum of 8 digits that can be shuffled. The bit positions in the first level of the accumulator stack defines the digits to be shuffled. They correspond to the bit positions in the accumulator that define the order the digits will be shuffled. The digits are shuffled and the result resides in the accumulator.

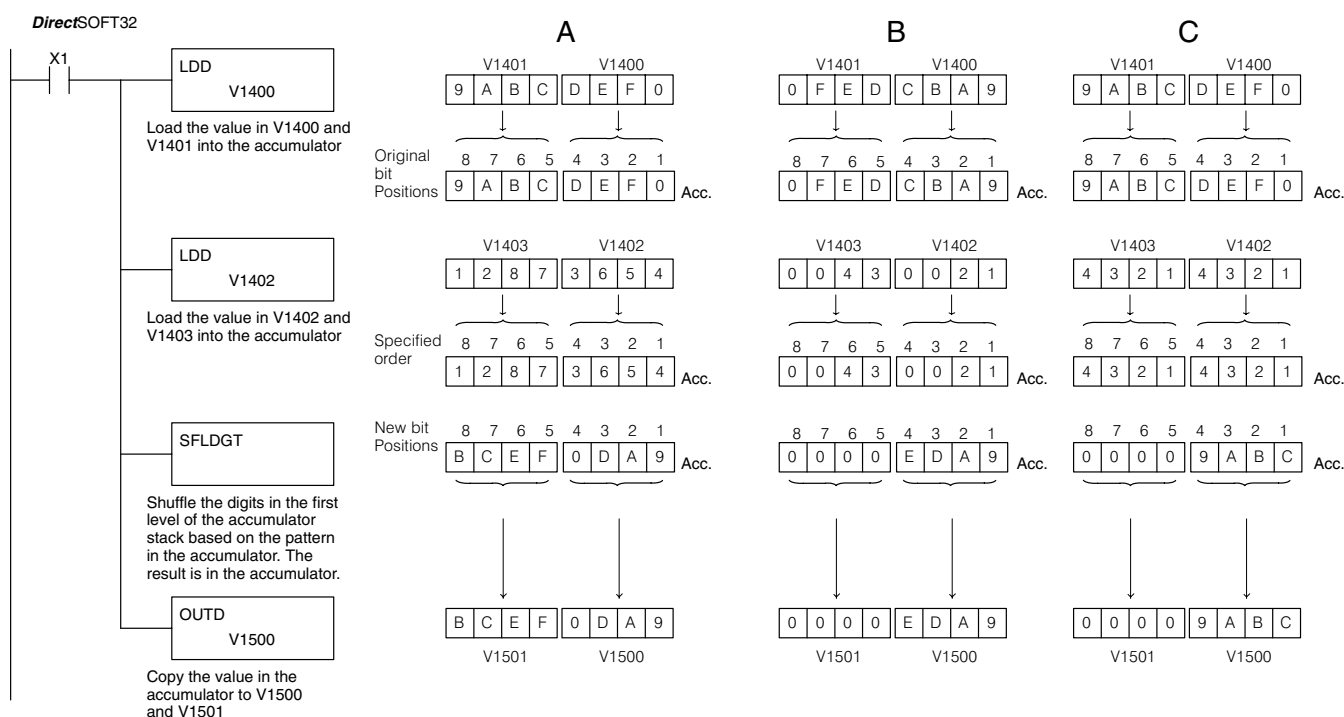


In the following example when X1 is on, The value in the first level of the accumulator stack will be reorganized in the order specified by the value in the accumulator.

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

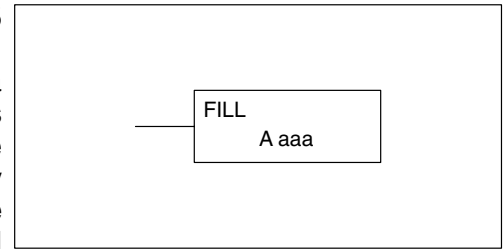
STR	X(IN)	1	←
LD	SHFT	D	SHFT V 1 4 0 0 ←
LD	SHFT	D	SHFT V 1 4 0 2 ←
SHFT	S	F	L D G T ←
OUT	SHFT	D	SHFT V 1 5 0 0 ←

Table Instructions

Fill (FILL)



The Fill instruction fills a table of up to 255 V-memory locations with a value (Aaaa), which is either a V-memory location or a 4-digit constant. The function parameters are loaded into the first level of the accumulator stack and the accumulator by two additional instructions. Listed below are the steps necessary to program the Fill function.



Step 1:— Load the number of V-memory locations to be filled into the first level of the accumulator stack. This parameter must be a HEX value, 0-FF.

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

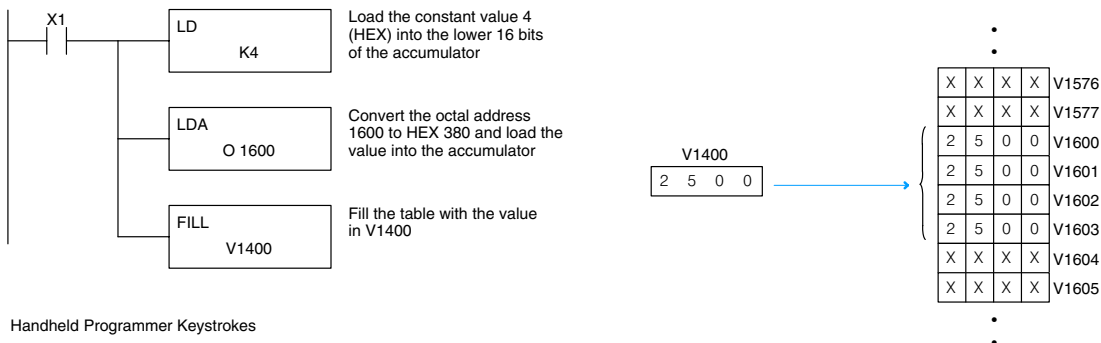
Step 3:— Insert the Fill instructions which specifies the value to fill the table with.

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

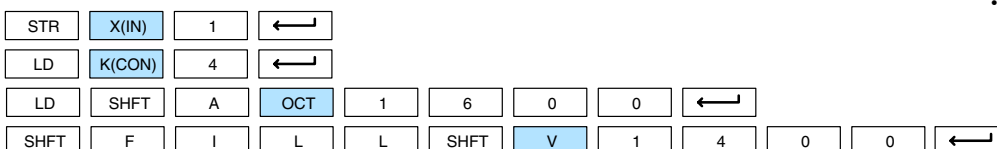
Operand Data Type		DL430 Range	DL440 Range	DL450 Range
A		aaa	aaa	aaa
Vmemory	V	All (See p. 3-40)	All (See p. 3-41)	All (See p. 3-42)
Pointer	P	All (See p. 3-40)	All (See p. 3-41)	All (See p. 3-42)
Constant	K	0-FF	0-FF	0-FF

In the following example, when X1 is on, the constant value (K4) is loaded into the accumulator using the Load instruction. This value specifies the length of the table and is placed on the first level of the accumulator stack when the Load Address instruction is executed. The octal address 1600 (V1600) is the starting location for the table and is loaded into the accumulator using the Load Address instruction. The value to fill the table with (V1400) is specified in the Fill instruction.

DirectSOFT32



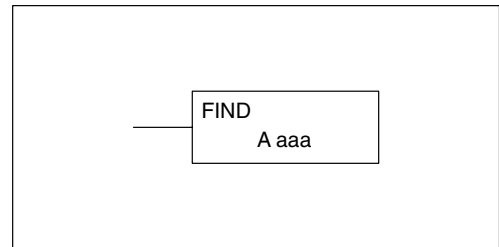
Handheld Programmer Keystrokes



**Find
(FIND)**

✕	✓	✓
430	440	450

The Find instruction is used to search for a specified value in a V-memory table of up to 255 locations. The function parameters are loaded into the first and second levels of the accumulator stack and the accumulator by three additional instructions. Listed below are the steps necessary to program the Find function.



Step 1:— Load the length of the table (number of V-memory locations) into the second level of the accumulator stack. This parameter must be a HEX value, 0-FF.

Step 2:— Load the starting V-memory location for the table into the first level of the accumulator stack. This parameter must be a HEX value.

Step 3:— Load the offset from the starting location to begin the search. This parameter must be a HEX value.

Step 4:— Insert the Find instruction which specifies the first value to be found in the table.

Results:— The offset from the starting address to the first V-memory location which contains the search value is returned to the accumulator. SP53 will be set on if an address outside the table is specified in the offset or the value is not found. If the value is not found 0 will be returned in the accumulator. The result will be a HEX value.

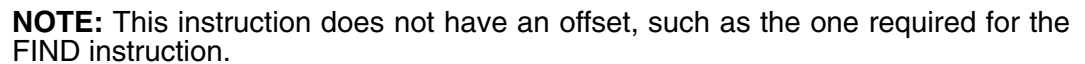
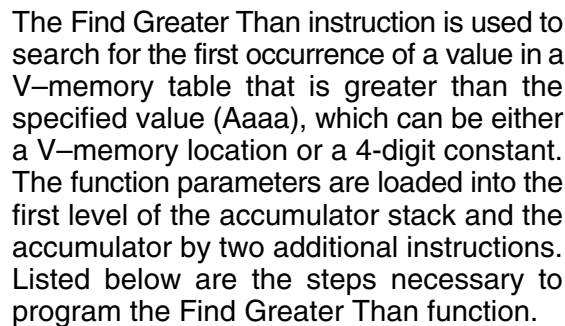
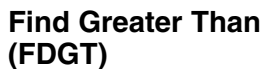
Helpful Hint: — For parameters that require HEX values when referencing memory locations, the LDA instruction can be used to convert an octal address to the HEX equivalent and load the value into the accumulator.

Operand Data Type		DL440 Range	DL450 Range
A		aaa	aaa
Vmemory	V	All (See p. 3-41)	All (See p. 3-42)
Constant	K	0-FFFF	0-FFFF

Discrete Bit Flags	Description
SP53	ON if there is no value in the table that is equal to the search value.

NOTE: Status flags are only valid until another instruction that uses the same flags is executed. The pointer for this instruction starts at 0 and resides in the accumulator.

In the following example, when X1 is on, the constant value (K6) is loaded into the accumulator using the Load instruction. This value specifies the length of the table and is placed in the second stack location when the following Load Address and Load instruction is executed. The octal address 1400 (V1400) is the starting location for the table and is loaded into the accumulator. This value is placed in the first level of the accumulator stack when the following Load instruction is executed. The offset (K2) is loaded into the lower 16 bits of the accumulator using the Load instruction. The value to be found in the table is specified in the Find instruction. If a value is found equal to the search value, the offset (from the starting location of the table) where the value is located will reside in the accumulator.



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

Results:— The offset from the starting address to the first V-memory location which contains the greater than search value is returned to the accumulator. SP53 will be set on if the value is not found and 0 will be returned in the accumulator. The result will be a HEX value.

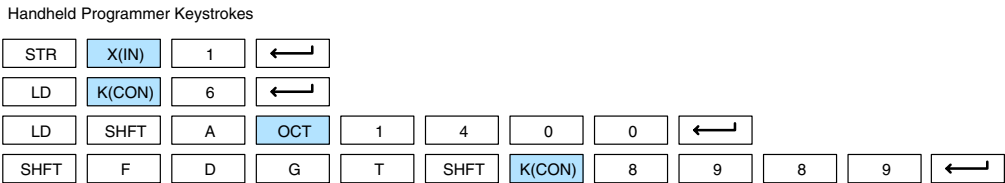
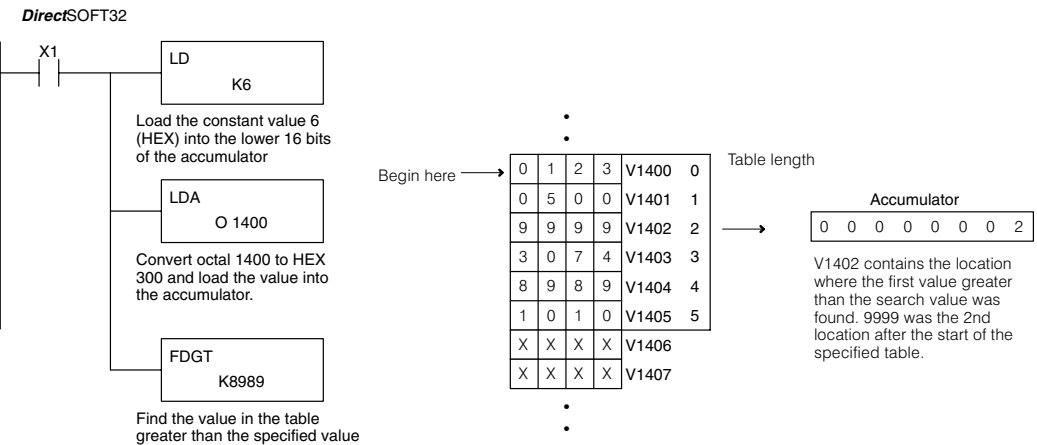
Helpful Hint: — For parameters that require HEX values when referencing memory locations, the LDA instruction can be used to convert an octal address to the HEX equivalent and load the value into the accumulator.

Operand Data Type		DL440 Range	DL450 Range
A		aaa	aaa
Vmemory	V	All (See p. 3-41)	All (See p. 3-42)
Constant	K	0-FFFF	0-FFFF

Discrete Bit Flags	Description
SP53	on if there is no value in the table that is greater than the search value.

NOTE: Status flags are only valid until another instruction that uses the same flags is executed.
The pointer for this instruction starts at 0 and resides in the accumulator.

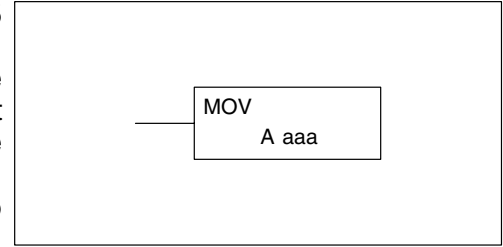
In the following example, when X1 is on, the constant value (K6) is loaded into the accumulator using the Load instruction. This value specifies the length of the table and is placed in the first stack location after the Load Address instruction is executed. The octal address 1400 (V1400) is the starting location for the table and is loaded into the accumulator. The greater than search value is specified in the Find Greater Than instruction. If a value is found greater than the search value, the offset (from the starting location of the table) where the value is located will reside in the accumulator. If there is no value in the table that is greater than the search value, a zero is stored in the accumulator and SP53 will come ON.



Move (MOV)

✕ ✓ ✓
430 440 450

The Move instruction moves up to 4095 values from a V-memory table to another V-memory table the same length. The function parameters are loaded into the first level of the accumulator stack and the accumulator by two additional instructions. Listed below are the steps necessary to program the Move function.



Step 1:— Load the number of V-memory locations to be moved into the first level of the accumulator stack. This parameter must be a HEX value, 0–FFF.

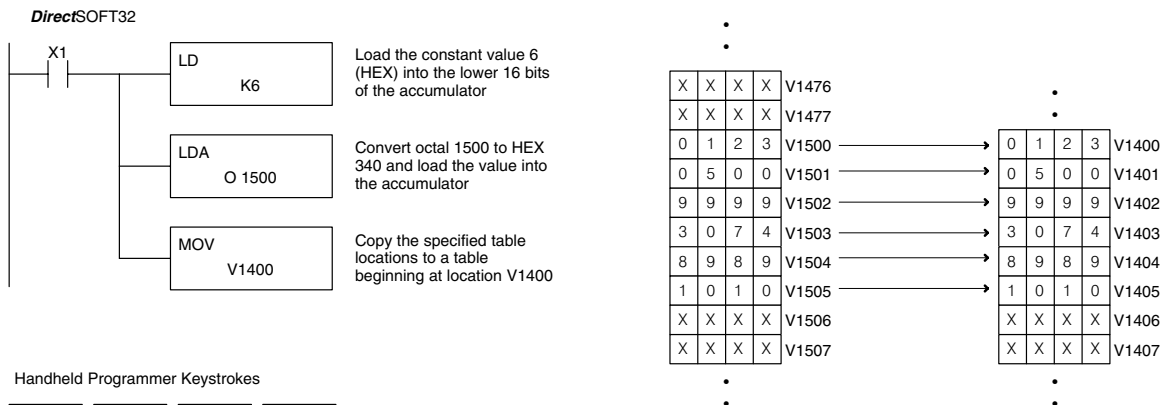
Step 2:— Load the starting V-memory location for the locations to be moved into the accumulator. This parameter must be a HEX value.

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

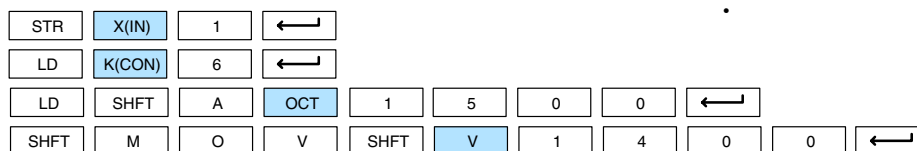
Helpful Hint: — For parameters that require HEX values when referencing memory locations, the LDA instruction can be used to convert an octal address to the HEX equivalent and load the value into the accumulator.

Operand Data Type		DL440 Range	DL450 Range
A		aaa	aaa
Vmemory	V	All (See p. 3-41)	All (See p. 3-42)
Pointer	P	All (See p. 3-41)	All (See p. 3-42)

In the following example, when X1 is on, the constant value (K6) is loaded into the accumulator using the Load instruction. This value specifies the length of the table and is placed in the first stack location after the Load Address instruction is executed. The octal address 1500 (V1500), the starting location for the source table is loaded into the accumulator. The destination table location (V1400) is specified in the Move instruction.



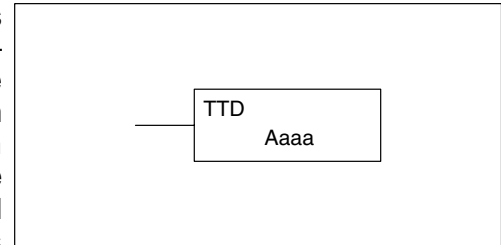
Handheld Programmer Keystrokes



**Table to
Destination
(TTD)**

✕	✓	✓
430	440	450

The Table To Destination instruction moves a value from a V-memory table to a V-memory location and increments the table pointer by 1. The first V-memory location in the table contains the table pointer which indicates the next location in the table to be moved. The instruction will be executed once per scan provided the input remains on. The table pointer will reset to 1 when the value equals the last location in the table. The function parameters are loaded into the first level of the accumulator stack and the accumulator by two additional instructions. Listed below are the steps necessary to program the Table To Destination function.



Step 1:— Load the length of the data table (number of V-memory locations) into the first level of the accumulator stack. This parameter must be a HEX value, 0 to FF.

Step 2:— Load the starting V-memory location for the table into the accumulator. (Remember, the starting location of the table is used as the table pointer.) This parameter must be a HEX value.

Step 3:— Insert the TTD instruction which specifies destination V-memory location (Vaaa).

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

Helpful Hint:— The instruction will be executed every scan if the input logic is on. If you do not want the instruction to execute for more than one scan, a one shot (PD) should be used in the input logic.

Helpful Hint: — The pointer location should be set to the value where the table operation will begin. The special relay SP0 or a one shot (PD) should be used so the value will only be set in one scan and will not affect the instruction operation.

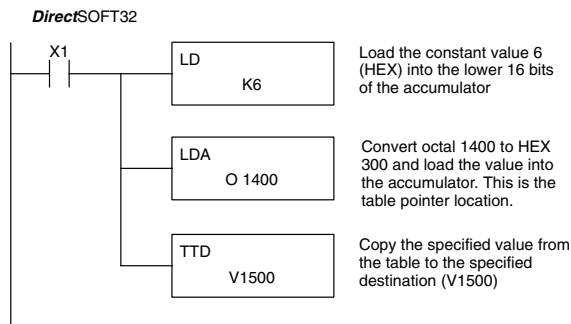
Operand Data Type	DL440 Range	DL450 Range
A	aaa	aaa
Vmemory V	All (See p. 3-41)	All (See p. 3-42)

Discrete Bit Flags	Description
SP56	ON when the table pointer equals the table length.

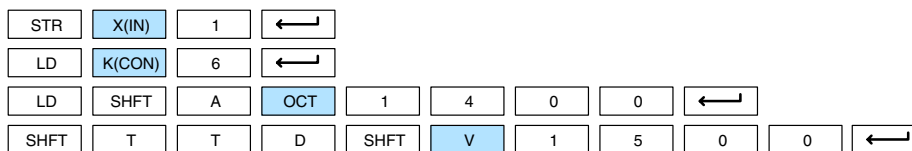
NOTE: Status flags (SPs) are only valid until:
— another instruction that uses the same flag is executed, or
— the end of the scan

The pointer for this instruction starts at 0 and resets when the table length is reached. At first glance it may appear that the pointer should reset to 0. However, it resets to 1, not 0.

In the following example, when X1 is on, the constant value (K6) is loaded into the accumulator using the Load instruction. This value specifies the length of the table and is placed in the first stack location after the Load Address instruction is executed. The octal address 1400 (V1400) is the starting location for the source table and is loaded into the accumulator. Remember, V1400 is used as the pointer location, and is not actually part of the table data source. The destination location (V1500) is specified in the Table to Destination instruction. The table pointer (V1400 in this case) will be increased by "1" after each execution of the TTD instruction.



Handheld Programmer Keystrokes

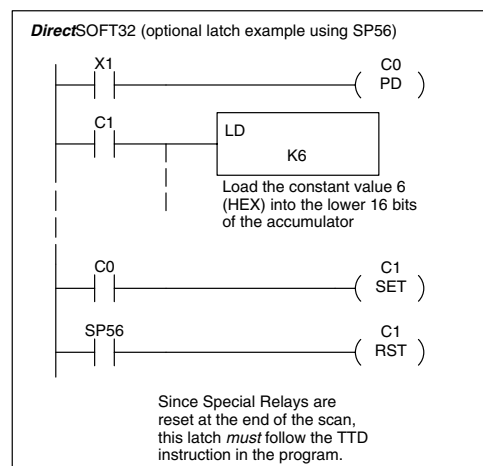


It is important to understand how the table locations are numbered. If you examine the example table, you'll notice that the first data location, V1401, will be used when the pointer is equal to zero, and again when the pointer is equal to six. Why? Because the pointer is only equal to zero before the very first execution. From then on, it increments from one to six, and then resets to one.

Table					Table Pointer	
V1401	0	5	0	0	0	6
V1402	9	9	9	9	1	
V1403	3	0	7	4	2	
V1404	8	9	8	9	3	
V1405	1	0	1	0	4	
V1406	2	0	4	6	5	
V1407	X	X	X	X		

Destination					V1500	
X	X	X	X	X		

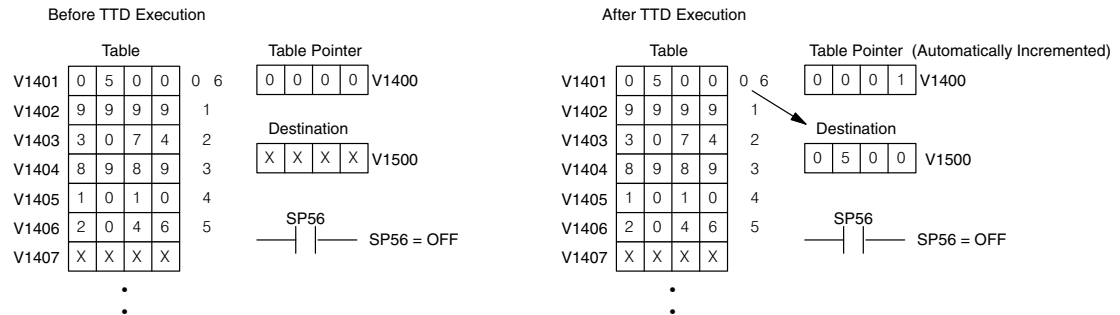
Also, our example uses a normal input contact (X1) to control the execution. Since the CPU scan is extremely fast, and the pointer increments automatically, the table would cycle through the locations very quickly. If this is a problem, you have an option of using SP56 in conjunction with a one-shot (PD) and a latch (C1 for example) to allow the table to cycle through all locations one time and then stop. The logic shown here is not required, it's just an optional method.



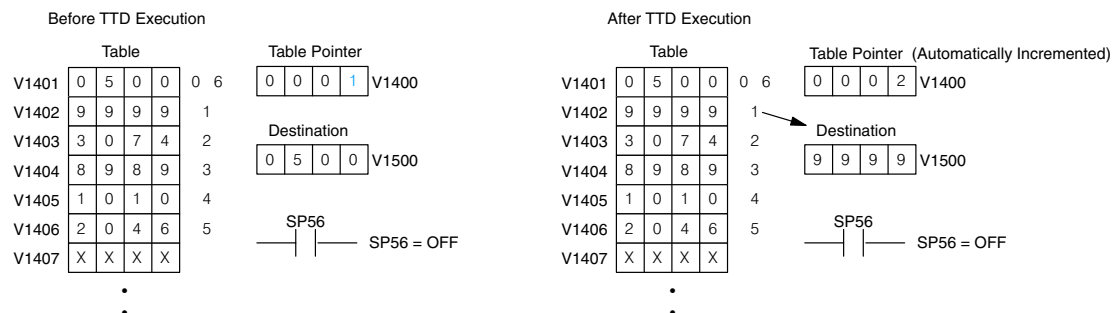
The following diagram shows the scan-by-scan results of the execution for our example program. Notice how the pointer automatically cycles from 0 – 6, and then starts over at 1 instead of 0. Also, notice how SP56 is only on until the end of the scan.

Example of Execution

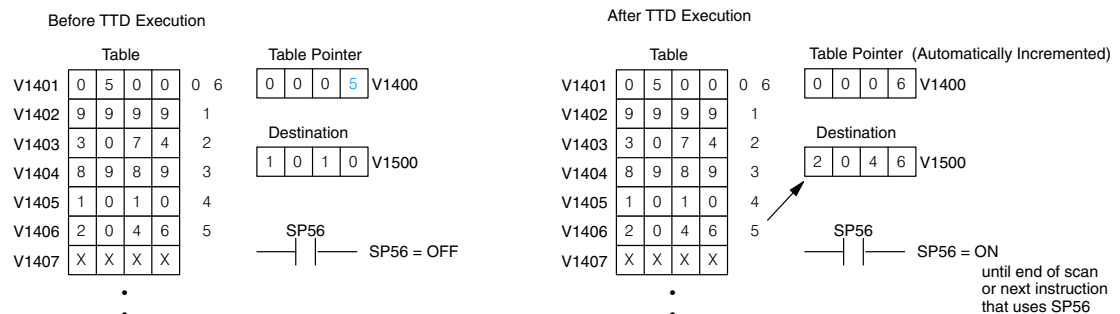
Scan N



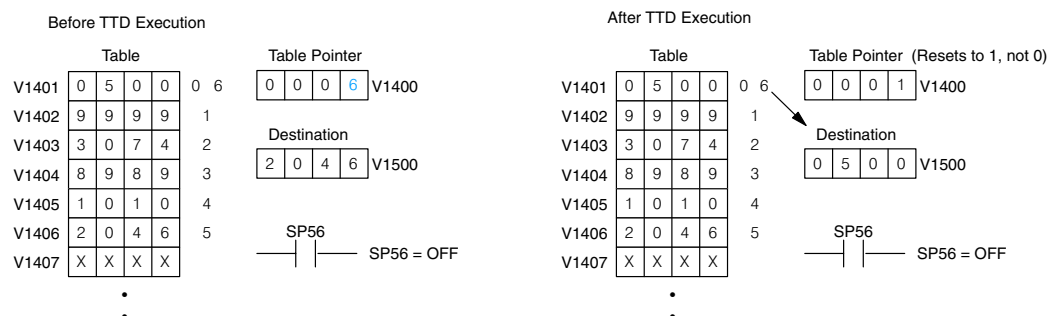
Scan N+1



Scan N+5



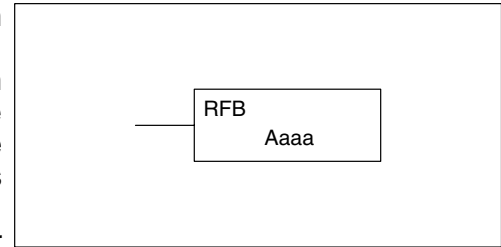
Scan N+6



Remove from Bottom (RFB)

✕	✓	✓
430	440	450

The Remove From Bottom instruction moves a value from the bottom of a V-memory table to a V-memory location and decrements a table pointer by 1. The first V-memory location in the table contains the table pointer which indicates the next location in the table to be moved. The instruction will be executed once per scan provided the input remains on. The instruction will stop operation when the pointer equals 0. The function parameters are loaded into the first level of the accumulator stack and the accumulator by two additional instructions. Listed below are the steps necessary to program the Remove From Bottom function.



Step 1:— Load the length of the table (number of V-memory locations) into the first level of the accumulator stack. This parameter must be a HEX value, 0 to FF.

Step 2:— Load the starting V-memory location for the table into the accumulator. (Remember, the starting location of the table blank is used as the table pointer.) This parameter must be a HEX value.

Step 3:— Insert the RFB instructions which specifies destination V-memory location (Vaaa).

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

Helpful Hint:— The instruction will be executed every scan if the input logic is on. If you do not want the instruction to execute for more than one scan, a one shot (PD) should be used in the input logic.

Helpful Hint: — The pointer location should be set to the value where the table operation will begin. The special relay SP0 or a one shot (PD) should be used so the value will only be set in one scan and will not affect the instruction operation.

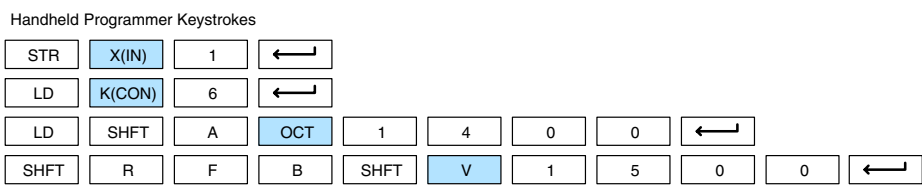
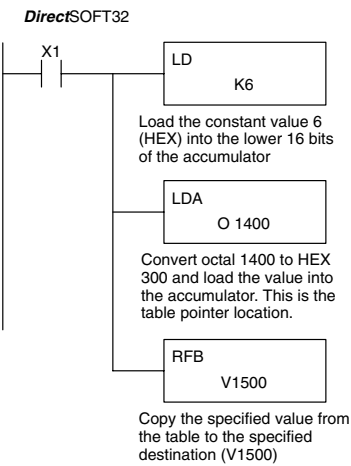
Operand Data Type	DL440 Range	DL450 Range
A	aaa	aaa
Vmemory V	All (See p. 3-41)	All (See p. 3-42)

Discrete Bit Flags	Description
SP56	on when the table pointer equals 0

NOTE: Status flags (SPs) are only valid until:
— another instruction that uses the same flag is executed, or
— the end of the scan

The pointer for this instruction can be set to start anywhere in the table. It is not set automatically. You have to load a value into the pointer somewhere in your program.

In the following example, when X1 is on, the constant value (K6) is loaded into the accumulator using the Load instruction. This value specifies the length of the table and is placed in the first stack location after the Load Address instruction is executed. The octal address 1400 (V1400) is the starting location for the source table and is loaded into the accumulator. Remember, V1400 is used as the pointer location, and is not actually part of the table data source. The destination location (V1500) is specified in the Remove From Bottom. The table pointer (V1400 in this case) will be decremented by “1” after each execution of the RFB instruction.

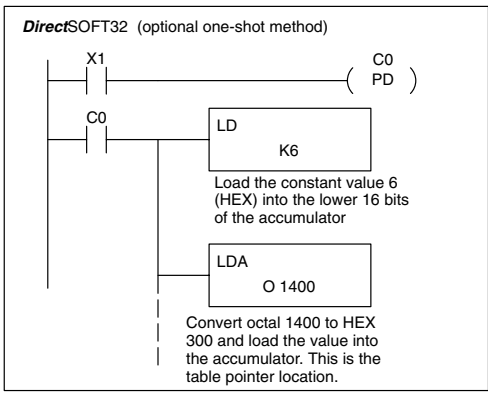


It is important to understand how the table locations are numbered. If you examine the example table, you'll notice that the first data location, V1401, will be used when the pointer is equal to one. The second data location, V1402, will be used when the pointer is equal to two, etc.

Table					Table Pointer				
V1401	0	5	0	0	1	0	0	0	V1400
V1402	9	9	9	9	2				
V1403	3	0	7	4	3				
V1404	8	9	8	9	4				
V1405	1	0	1	0	5				
V1406	2	0	4	6	6				
V1407	X	X	X	X					

Destination				
X	X	X	X	V1500

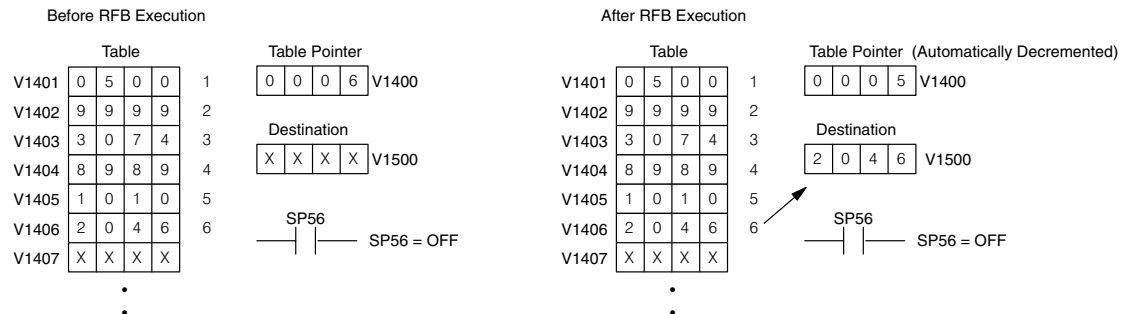
Also, our example uses a normal input contact (X1) to control the execution. Since the CPU scan is extremely fast, and the pointer decrements automatically, the table would cycle through the locations very quickly. If this is a problem for your applicaton, you have an option of using a one-shot (PD) to remove one value each time the input contact transitions from low to high.



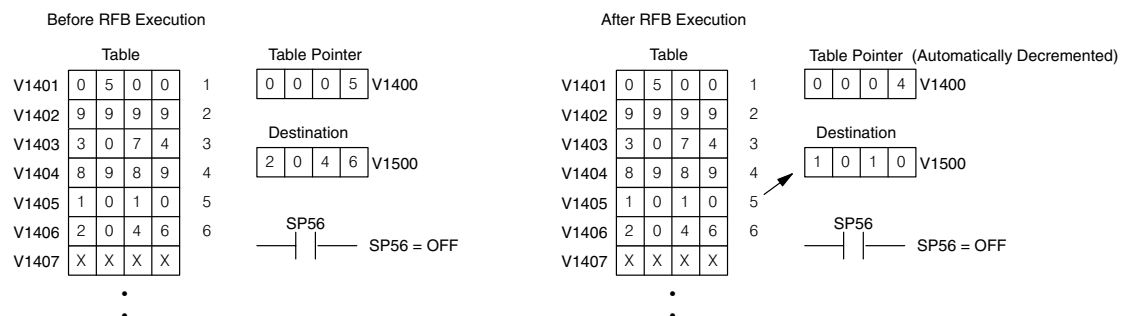
The following diagram shows the scan-by-scan results of the execution for our example program. Notice how the pointer automatically decrements from 6 – 0. Also, notice how SP56 is only on until the end of the scan.

Example of Execution

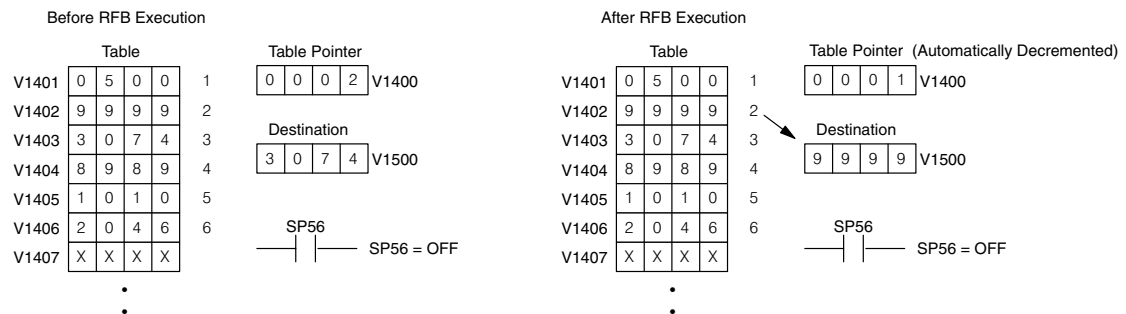
Scan N



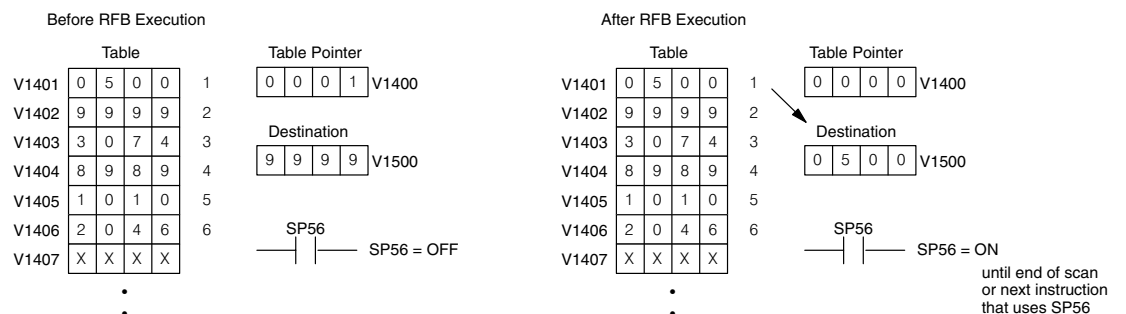
Scan N+1



Scan N+4



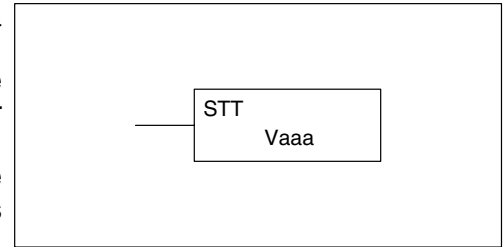
Scan N+5



**Source to Table
(STT)**

✕	✓	✓
430	440	450

The Source To Table instruction moves a value from a V-memory location into a V-memory table and increments a table pointer by 1. When the table pointer reaches the end of the table, it resets to 1. The first V-memory location in the table contains the table pointer which indicates the next location in the table to store a value. The instruction will be executed once per scan provided the input remains on. The function parameters are loaded into the first level of the accumulator stack and the accumulator with two additional instructions. Listed below are the steps necessary to program the Source To Table function.



Step 1:— Load the length of the table (number of V-memory locations) into the first level of the accumulator stack. This parameter must be a HEX value, 0 to FF.

Step 2:— Load the starting V-memory location for the table into the accumulator. (Remember, the starting location of the table is used as the table pointer.) This parameter must be a HEX value.

Step 3:— Insert the STT instruction which specifies the source V-memory location (Vaaa). This is where the value will be moved from.

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

Helpful Hint:— The instruction will be executed every scan if the input logic is on. If you do not want the instruction to execute for more than one scan, a one shot (PD) should be used in the input logic.

Helpful Hint: — The table counter value should be set to indicate the starting point for the operation. Also, it must be set to a value that is within the length of the table. For example, if the table is 6 words long, then the allowable range of values that could be in the pointer should be between 0 and 6. If the value is outside of this range, the data will not be moved. Also, a one shot (PD) should be used so the value will only be set in one scan and will not affect the instruction operation.

Operand Data Type	DL440 Range	DL450 Range
	aaa	aaa
Vmemory V	All (See p. 3-41)	All (See p. 3-42)

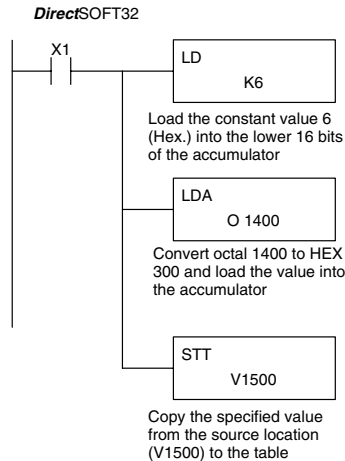
Discrete Bit Flags	Description
SP56	on when the table pointer equals the table length

NOTE: Status flags (SPs) are only valid until:

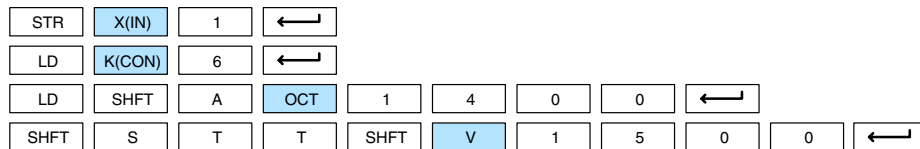
- another instruction that uses the same flag is executed, or
- the end of the scan

The pointer for this instruction starts at 0 and resets to 1 automatically when the table length is reached.

In the following example, when X1 is on, the constant value (K6) is loaded into the accumulator using the Load instruction. This value specifies the length of the table and is placed in the first stack location after the Load Address instruction is executed. The octal address 1400 (V1400), which is the starting location for the destination table and table pointer, is loaded into the accumulator. The data source location (V1500) is specified in the Source to Table instruction. The table pointer will be increased by "1" after each time the instruction is executed.



Handheld Programmer Keystrokes

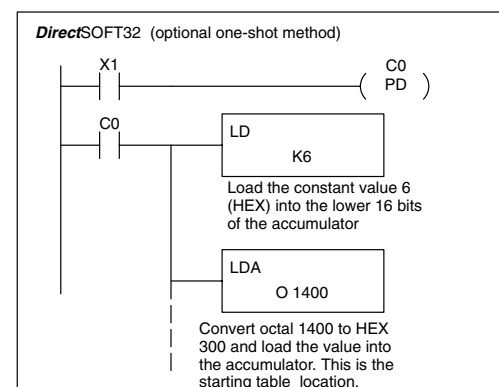


It is important to understand how the table locations are numbered. If you examine the example table, you'll notice that the first data storage location, V1401, will be used when the pointer is equal to zero, and again when the pointer is equal to six. Why? Because the pointer is only equal to zero before the very first execution. From then on, it increments from one to six, and then resets to one.

Also, our example uses a normal input contact (X1) to control the execution. Since the CPU scan is extremely fast, and the pointer increments automatically, the source data would be moved into all the table locations very quickly. If this is a problem for your application, you have an option of using a one-shot (PD) to move one value each time the input contact transitions from low to high.

Table					Table Pointer				
V1401	X	X	X	X	0	6	0	0	0
V1402	X	X	X	X	1				
V1403	X	X	X	X	2				
V1404	X	X	X	X	3				
V1405	X	X	X	X	4				
V1406	X	X	X	X	5				
V1407	X	X	X	X					

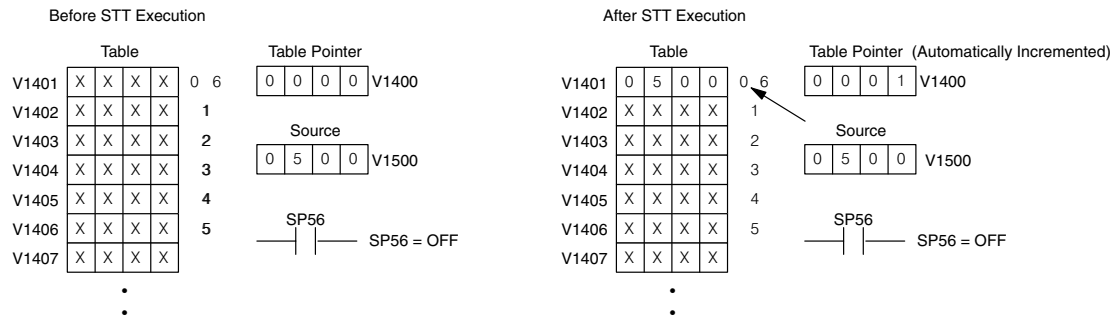
Data Source
0 5 0 0 V1500



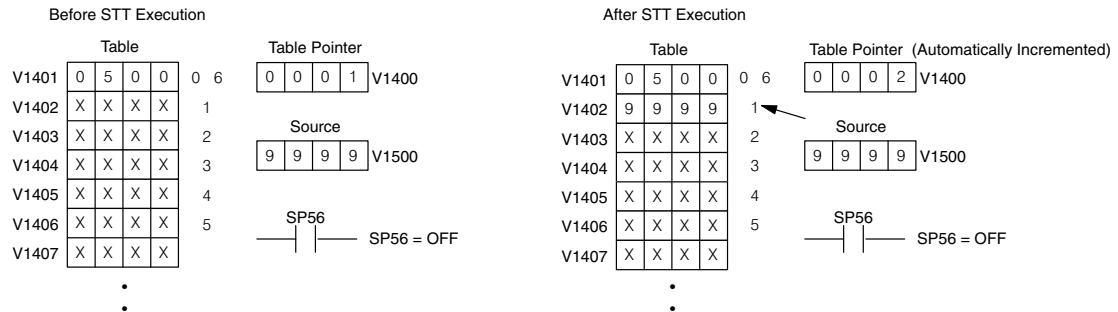
The following diagram shows the scan-by-scan results of the execution for our example program. Notice how the pointer automatically cycles from 0 – 6, and then starts over at 1 instead of 0. Also, notice how SP56 is affected by the execution. Although our example does not show it, we are assuming that there is another part of the program that changes the value in V1500 (data source) prior to the execution of the STT instruction. This is not required, but it makes it easier to see how the data source is copied into the table.

Example of Execution

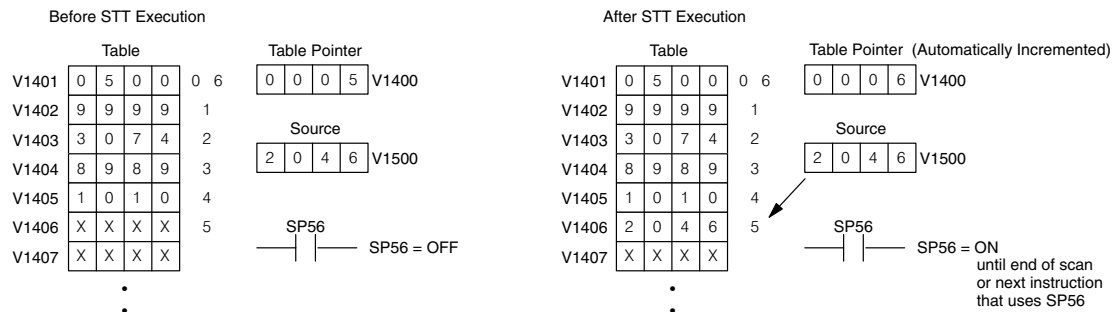
Scan N



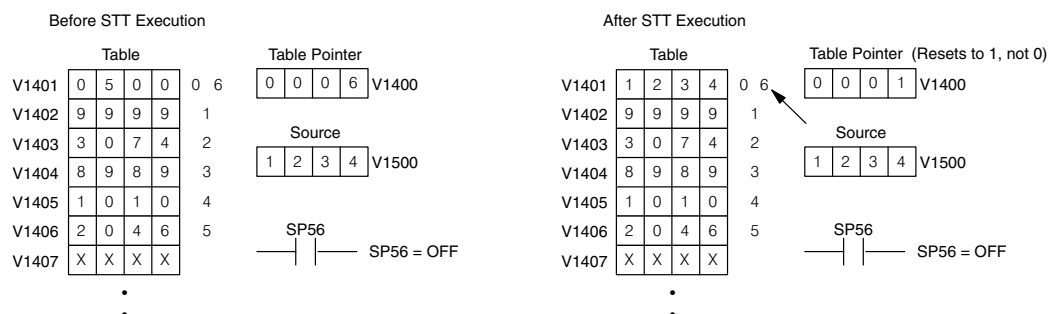
Scan N+1



Scan N+5



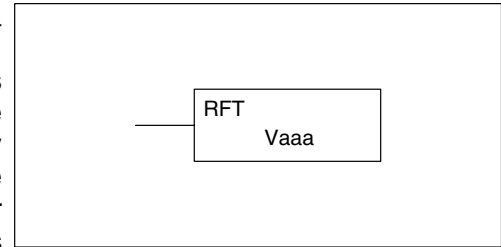
Scan N+6



Remove from Table (RFT)

×	✓	✓
430	440	450

The Remove From Table instruction pops a value off of a table and stores it in a V-memory location. When a value is removed from the table all other values are shifted up 1 location. The first V-memory location in the table contains the table length counter. The table counter decrements by 1 each time the instruction is executed. If the length counter is zero or greater than the maximum table length (specified in the first level of the accumulator stack) the instruction will not execute and SP56 will be on.



The instruction will be executed once per scan provided the input remains on. The function parameters are loaded into the first level of the accumulator stack and the accumulator by two additional instructions. Listed below are the steps necessary to program the Remove From Table function.

Step 1:— Load the length of the table (number of V-memory locations) into the first level of the accumulator stack. This parameter must be a HEX value, 0 to FF.

Step 2:— Load the starting V-memory location for the table into the accumulator. (Remember, the starting location of the table is used as the table length counter.) This parameter must be a HEX value.

Step 3:— Insert the RFT instructions which specifies destination V-memory location (Vaaa). This is where the value will be moved to.

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

Helpful Hint:— The instruction will be executed every scan if the input logic is on. If you do not want the instruction to execute for more than one scan, a one shot (PD) should be used in the input logic.

Helpful Hint: — The table counter value should be set to indicate the starting point for the operation. Also, it must be set to a value that is within the length of the table. For example, if the table is 6 words long, then the allowable range of values that could be in the table counter should be between 1 and 6. If the value is outside of this range or zero, the data will not be moved from the table. Also, a one shot (PD) should be used so the value will only be set in one scan and will not affect the instruction operation.

Operand Data Type	DL440 Range	DL450 Range
	aaa	aaa
Vmemory V	All (See p. 3-41)	All (See p. 3-42)

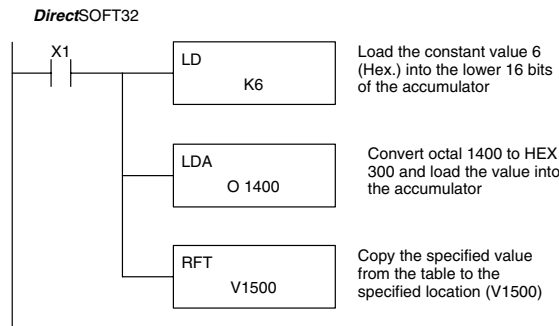
Discrete Bit Flags	Description
SP56	on when the table counter equals 0

NOTE: Status flags (SPs) are only valid until:

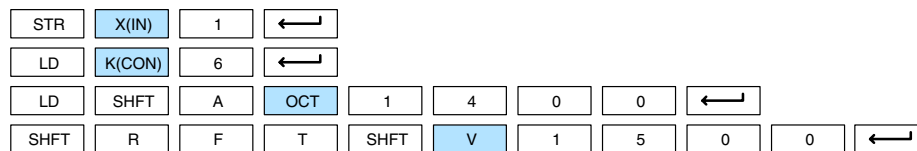
- another instruction that uses the same flag is executed, or
- the end of the scan

The pointer for this instruction can be set to start anywhere in the table. It is not set automatically. You have to load a value into the pointer somewhere in your program.

In the following example, when X1 is on, the constant value (K6) is loaded into the accumulator using the Load instruction. This value specifies the length of the table and is placed in the first stack location after the Load Address instruction is executed. The octal address 1400 (V1400) is the starting location for the source table and is loaded into the accumulator. The destination location (V1500) is specified in the Remove from Table instruction. The table counter will be decreased by "1" after the instruction is executed.



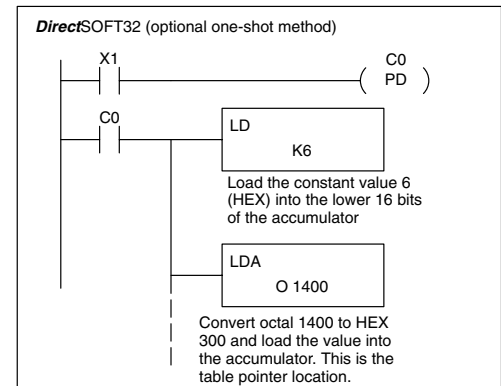
Handheld Programmer Keystrokes



Since the table counter specifies the range of data that will be removed from the table, it is important to understand how the table locations are numbered. If you examine the example table, you'll notice that the data locations are numbered from the top of the table. For example, if the table counter started at 6, then all six of the locations would be affected during the instruction execution.

Also, our example uses a normal input contact (X1) to control the execution. Since the CPU scan is extremely fast, and the pointer decrements automatically, the data would be removed from the table very quickly. If this is a problem for your application, you have an option of using a one-shot (PD) to remove one value each time the input contact transitions from low to high.

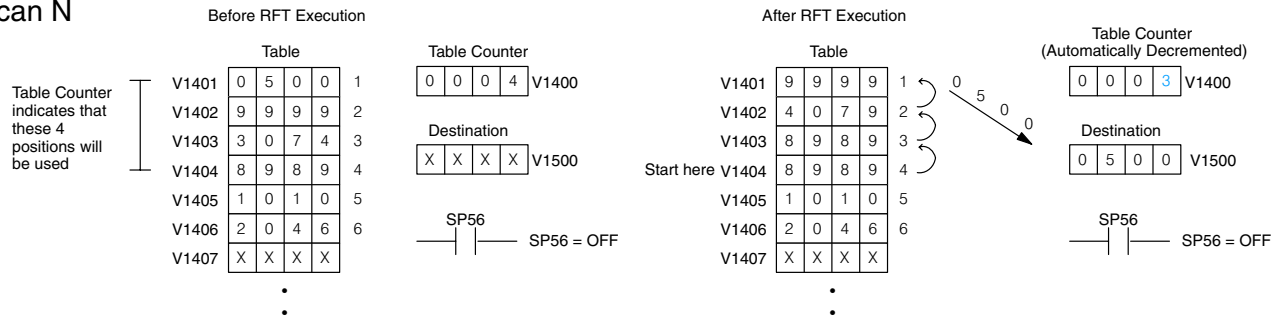
Table					Table Counter				
V1401	0	5	0	0	1	0	0	0	V1400
V1402	9	9	9	9	2				
V1403	3	0	7	4	3				
V1404	8	9	8	9	4				
V1405	1	0	1	0	5				
V1406	2	0	4	6	6				
V1407	X	X	X	X					



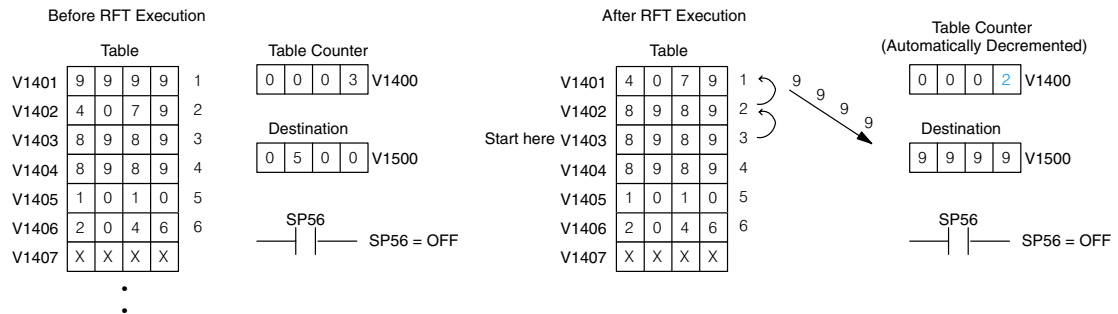
The following diagram shows the scan-by-scan results of the execution for our example program. In our example we're showing the table counter set to 4 initially. (Remember, you can set the table counter to any value that is within the range of the table.) The table counter automatically decrements from 4-0 as the instruction is executed. Notice how the last two table positions, 5 and 6, are not moved up through the table. Also, notice how SP56, which comes on when the table counter is zero, is only on until the end of the scan.

Example of Execution

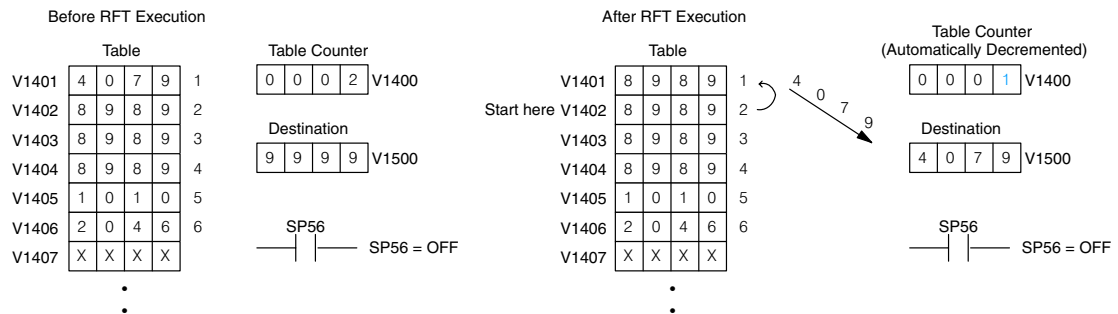
Scan N



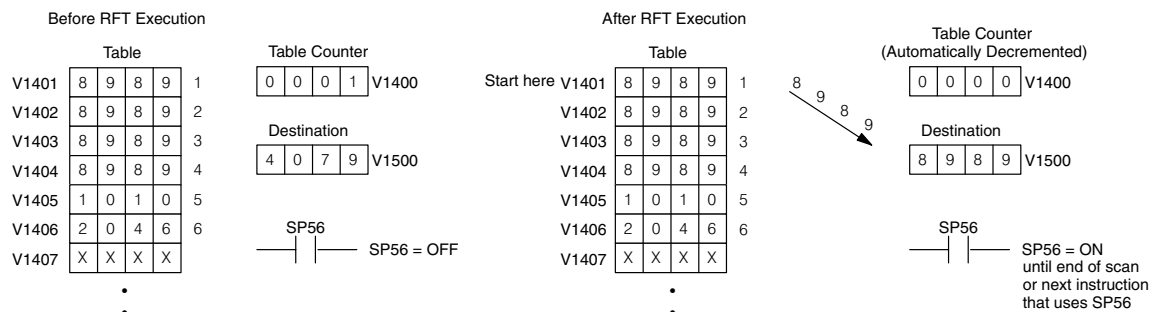
Scan N+1



Scan N+2



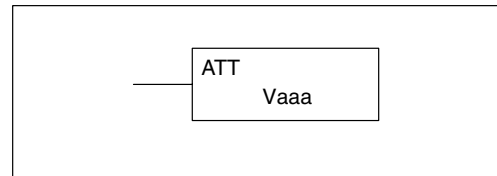
Scan N+3



**Add to Top
(ATT)**

430 440 450

The Add To Top instruction pushes a value on to a V-memory table from a V-memory location. When the value is added to the table all other values are pushed down 1 location.



The instruction will be executed once per scan provided the input remains on. The function parameters are loaded into the first level of the accumulator stack and the accumulator by two additional instructions. Listed below are the steps necessary to program the Add To Top function.

Step 1:— Load the length of the table (number of V-memory locations) into the first level of the accumulator stack. This parameter must be a HEX value, 0 to FF.

Step 2:— Load the starting V-memory location for the table into the accumulator. (Remember, the starting location of the table is used as the table length counter.) This parameter must be a HEX value.

Step 3:— Insert the ATT instructions which specifies source V-memory location (Vaaa). This is where the value will be moved from.

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

Helpful Hint:— The instruction will be executed every scan if the input logic is on. If you do not want the instruction to execute for more than one scan, a one shot (PD) should be used in the input logic.

Helpful Hint: — The table counter value should be set to indicate the starting point for the operation. Also, it must be set to a value that is within the length of the table. For example, if the table is 6 words long, then the allowable range of values that could be in the table counter should be between 1 and 6. If the value is outside of this range or zero, the data will not be moved into the table. Also, a one shot (PD) should be used so the value will only be set in one scan and will not affect the instruction operation.

Operand Data Type		DL440 Range	DL450 Range
		aaa	aaa
Vmemory	V	All (See p. 3-41)	All (See p. 3-42)

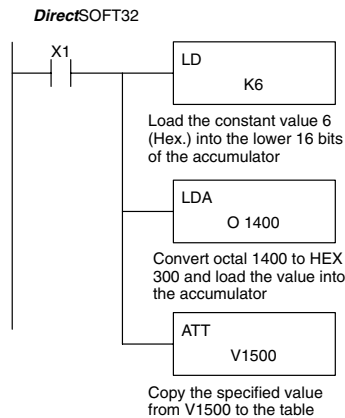
Discrete Bit Flags	Description
SP56	on when the table counter is equal to the table size

NOTE: Status flags (SPs) are only valid until:

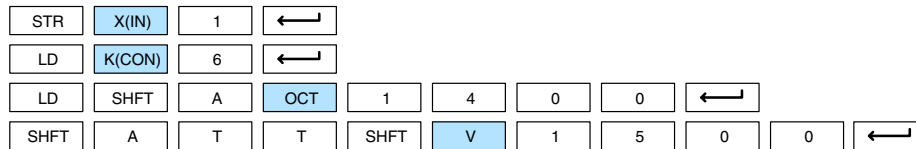
- another instruction that uses the same flag is executed, or
- the end of the scan

The pointer for this instruction can be set to start anywhere in the table. It is not set automatically. You have to load a value into the pointer somewhere in your program.

In the following example, when X1 is on, the constant value (K6) is loaded into the accumulator using the Load instruction. This value specifies the length of the table and is placed in the first stack location after the Load Address instruction is executed. The octal address 1400 (V1400), which is the starting location for the destination table and table counter, is loaded into the accumulator. The source location (V1500) is specified in the Add to Top instruction. The table counter will be increased by "1" after the instruction is executed.



Handheld Programmer Keystrokes



For the ATT instruction, the table counter determines the number of additions that can be made before the instruction will stop executing. So, it is helpful to understand how the system uses this counter to control the execution.

For example, if the table counter was set to 2, and the table length was 6 words, then there could only be 4 additions of data before the execution was stopped. This can easily be calculated by:

$$\text{Table length} - \text{table counter} = \text{number of executions}$$

Also, our example uses a normal input contact (X1) to control the execution. Since the CPU scan is extremely fast, and the table counter increments automatically, the data would be moved into the table very quickly. If this is a problem for your application, you have an option of using a one-shot (PD) to add one value each time the input contact transitions from low to high.

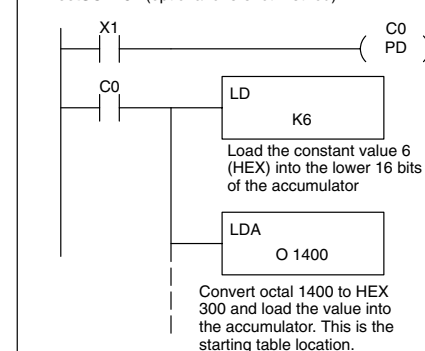
Table					
V1401	0	5	0	0	1
V1402	9	9	9	9	2
V1403	3	0	7	4	3
V1404	8	9	8	9	4
V1405	1	0	1	0	5
V1406	2	0	4	6	6
V1407	X	X	X	X	

Table Counter					
0	0	0	2		V1400

Data Source					
X	X	X	X		V1500

: (e.g., 6 – 2 = 4).

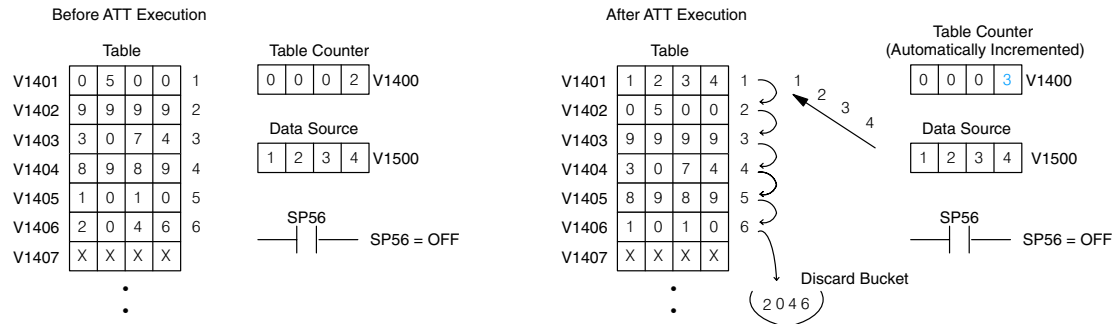
DirectSOFT32 (optional one-shot method)



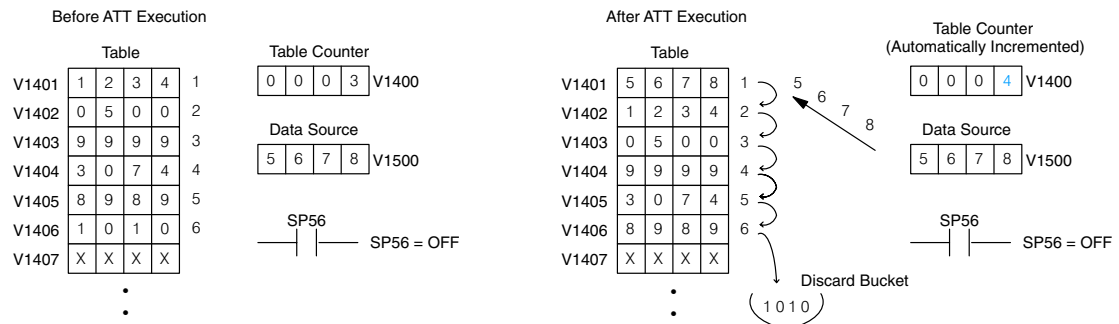
The following diagram shows the scan-by-scan results of the execution for our example program. The table counter is set to 2 initially, and it will automatically increment from 2 – 6 as the instruction is executed. Notice how SP56 comes on when the table counter is 6, which is equal to the table length. Plus, although our example does not show it, we are assuming that there is another part of the program that changes the value in V1500 (data source) prior to the execution of the ATT instruction.

Example of Execution

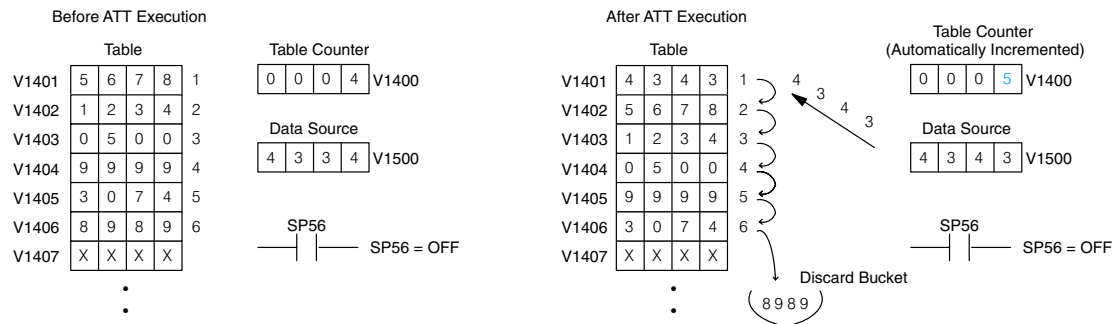
Scan N



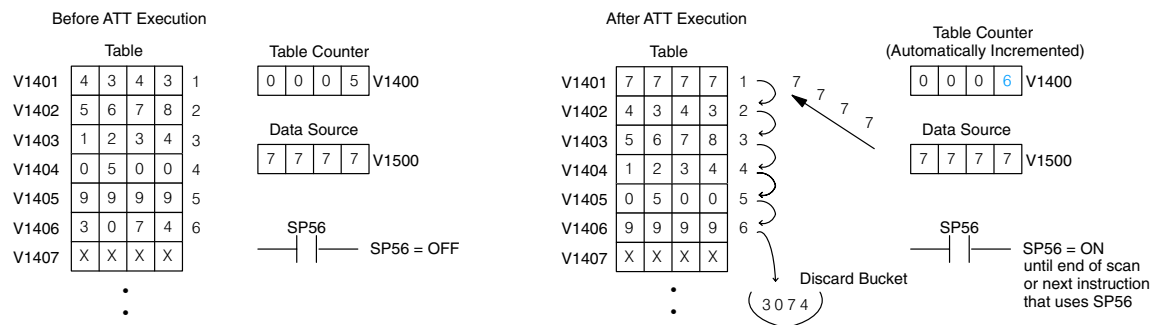
Scan N+1



Scan N+2



Scan N+3

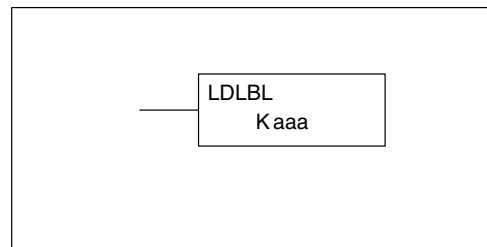
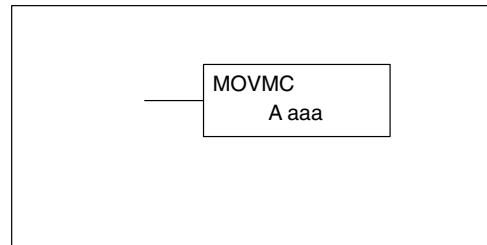


Move Memory Cartridge / Load Label (MOVMC / LDLBL)

☐ ☒ ☒
430 440 450

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 (255 maximum) to be copied into the second level of the accumulator stack. This must be a hex value, 0 to FF.

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

Step 3:— Load the *source data label* (LDLBL Kaaa) into the accumulator when copying data from ladder memory to V-memory. Load the *source address* into the accumulator when copying data from V-memory to ladder memory. 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		DL440 Range	DL450 Range
	A	aaa	aaa
Vmemory	V	All (See p. 3-41)	All (See p. 3-42)
Constant	K	1-FFFF	1-FFFF

Discrete Bit Flags	Description
SP53	on if there is a table pointer error.

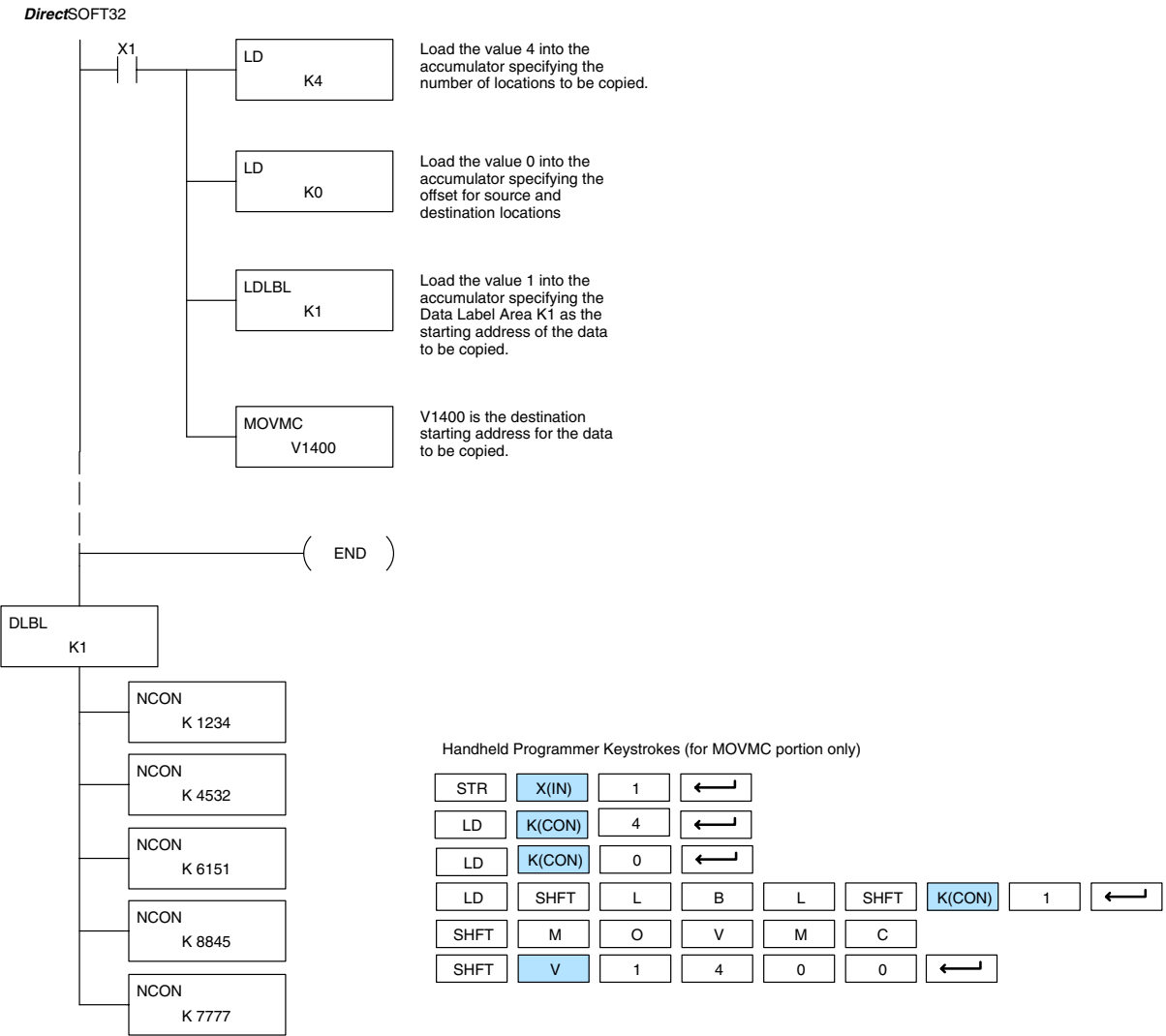
NOTE: Status flags are only valid until:
— the end of the scan
— or another instruction that uses the same flag is executed.

WARNING: The offset for this usage of the instruction starts at 0, but may be any number that *does not* result in data outside of the source data area being copied into the destination table. When an offset is outside of the source information boundaries, then unknown data values will be transferred into the destination table.

Copy Data From a Data Label Area to V-memory

☒ 430
 ☒ 440
 ☒ 450

In the following example, data is copied from a Data Label Area to V-memory. When X1 is on, the constant value (K4) is loaded into the accumulator using the Load instruction. This value specifies the length of the destination table and is placed in the second stack location after the next Load and Load Label (LDLBL) instructions are executed. The constant value (K0) is loaded into the accumulator using the Load instruction. This value specifies the offset for the source *and* the destination table, and is placed in the first stack location after the LDLBL instruction is executed. The source address where data is being copied from is loaded into the accumulator using the LDLBL instruction. The MOVMC instruction specifies the destination table starting location and executes the copying of data from the source Data Label Area to V-memory.

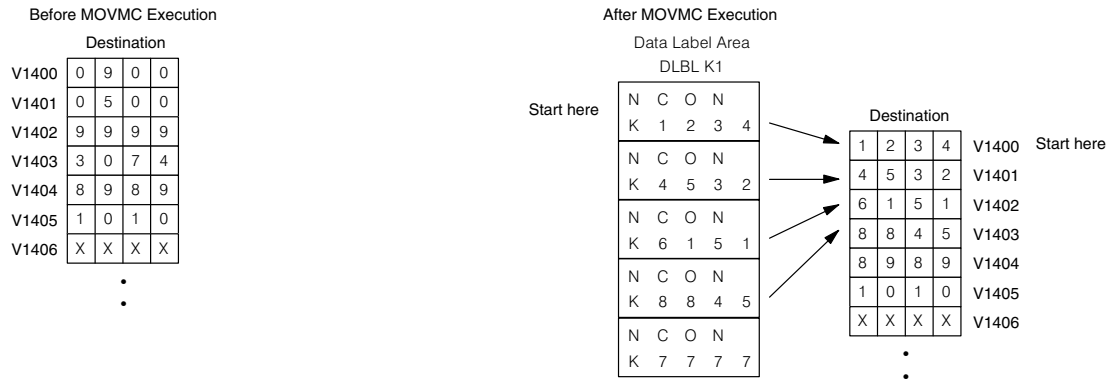


WARNING: The offset for this usage of the instruction starts at 0, but may be any number that *does not* result in data outside of the source data area being copied into the destination table. When an offset is outside of the source information boundaries, then unknown data values will be transferred into the destination table.

The following diagram shows the result of our example. The offset is equal to zero and four words will be copied into the Data Label area.

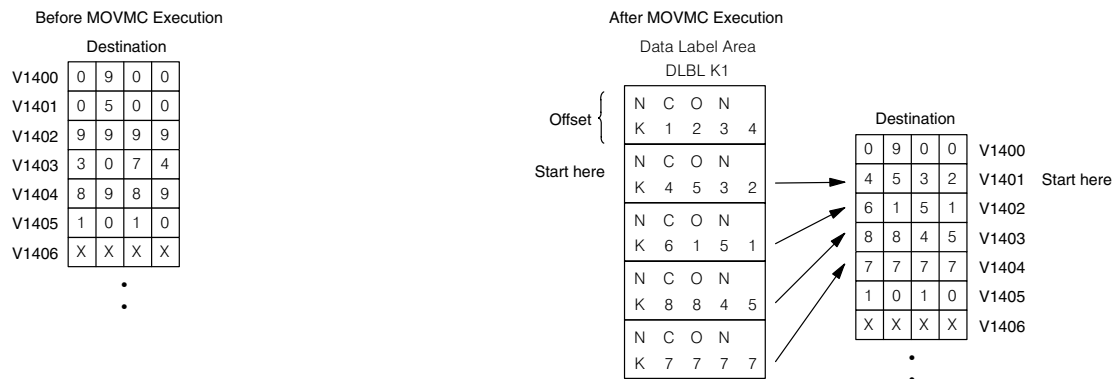
Example of Execution

Offset = 0, move 4 words

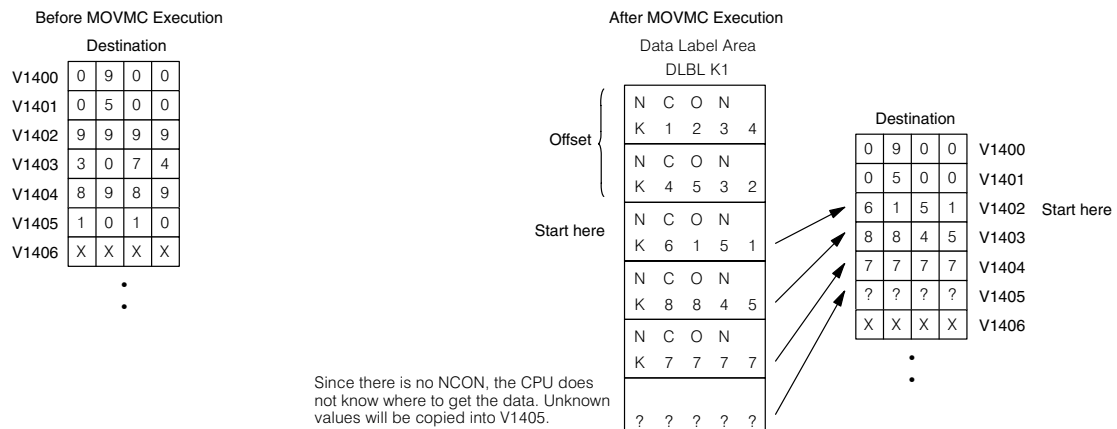


The example is fairly straightforward when an offset of zero is used. However, it is also helpful for you to understand the results that would have been obtained if different offset values (1 and 2) were used. Notice how the offset is used for both the data label (source) *and* the destination table. Also, notice how an improper offset (two in this case) can result in unknown values being copied into the destination table.

Offset = 1, move 4 words



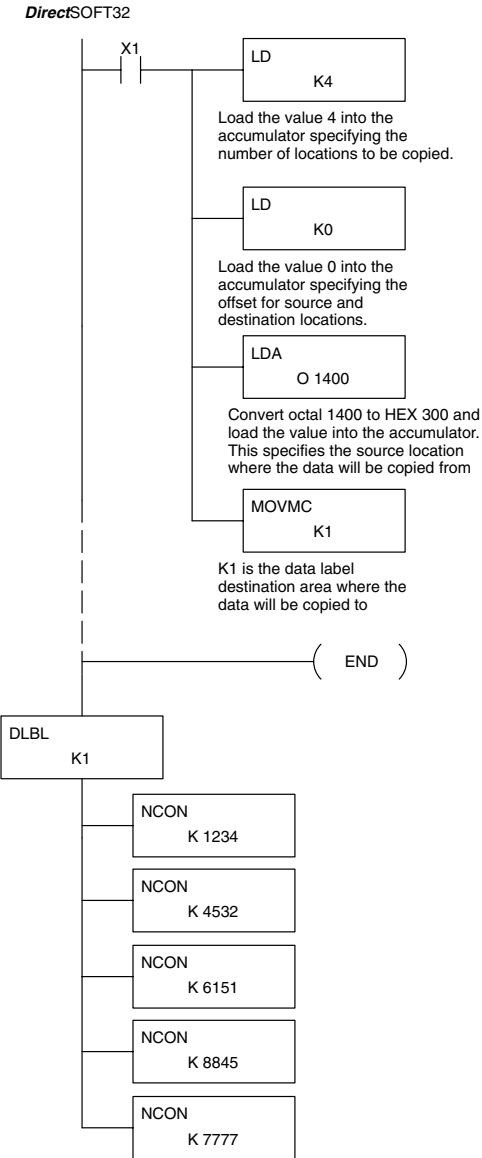
Offset = 2, move 4 words



Copy Data From
V-memory to a
Data Label Area

✕ ✓ ✕
430 440 450

NOTE: You must use a RAM cartridge for this example to work. In this example, data is copied from V-memory to a data label area. When X1 is on, the constant value (K4) is loaded into the accumulator using the Load instruction. This value specifies the length of the destination table and is placed in the second stack location after the next Load and Load Address instructions are executed. The constant value (K0) is loaded into the accumulator using the Load instruction. This value specifies the offset for the source *and* destination table, and is placed in the first stack location after the Load Address instruction is executed. The source address 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.



Handheld Programmer Keystrokes (for MOVMC portion only)

STR	X(IN)	1	←
LD	K(CON)	4	←
LD	K(CON)	0	←
LD	SHFT	A	OCT
SHFT	M	O	V
			1
			4
			0
			0
			←
			1

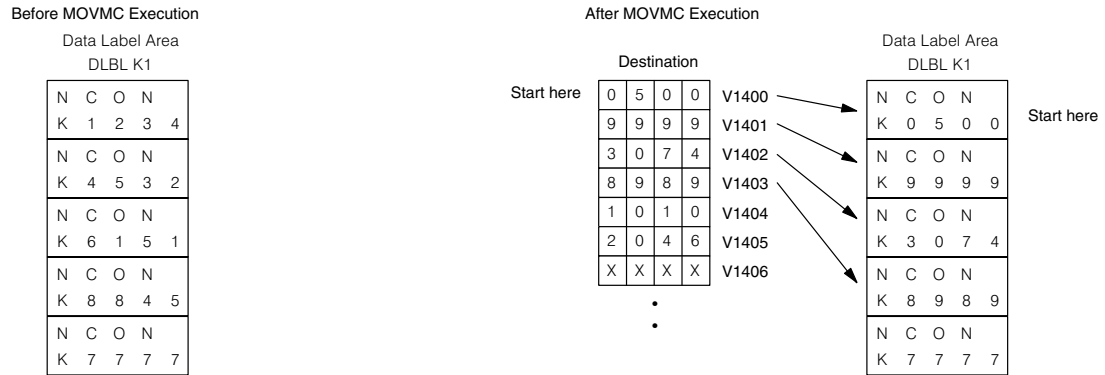
Note: This instruction works only with the RAM cartridge. It does not work with the EEPROM.

WARNING: The offset for this usage of the instruction starts at 0. If the offset (or the specified data table range) is large enough to cause data to be copied from V-memory to beyond the end of the DLBL area, then anything after the specified DLBL area will be replaced with invalid instructions.

The following diagram shows the result of our example. The offset is equal to zero and four words will be copied into the Data Label area.

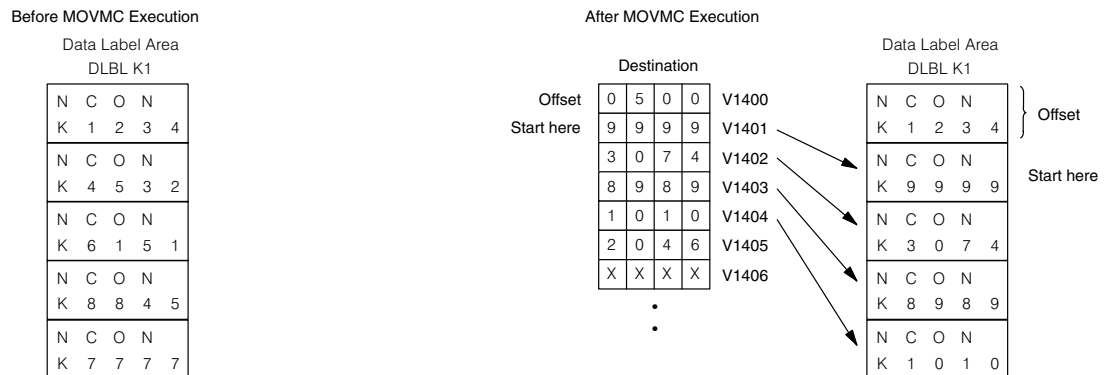
Example of Execution

Offset = 0, move 4 words

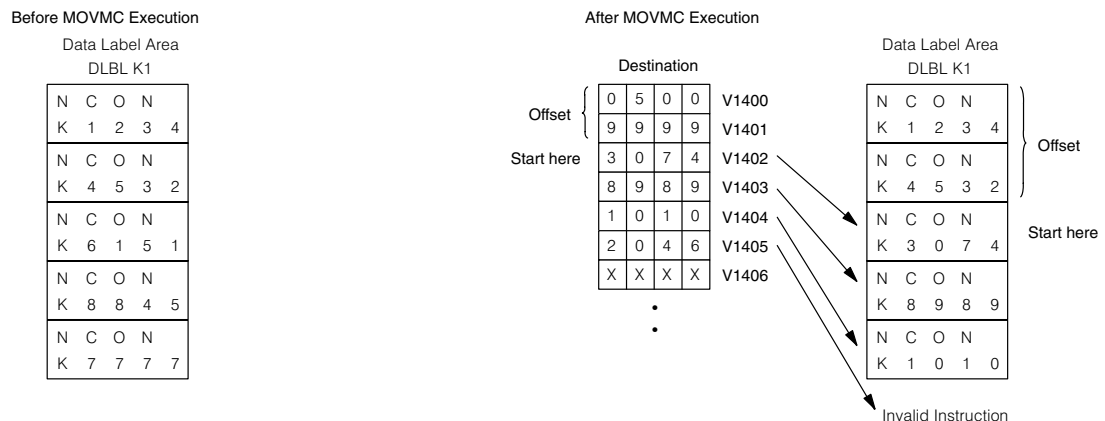


The example is fairly straightforward when an offset of zero is used. However, it is also helpful for you to understand the results that would have been obtained if different offset values (1 and 2) were used. Notice how the offset is used for both the V-memory data table (source) *and* the Data Label area. Also, notice how an improper offset (two in this case) can result in invalid instructions being written over any instructions that follow the Data Label.

Offset = 1, move 4 words



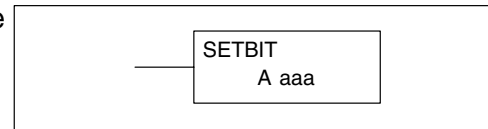
Offset = 2, move 4 words



Set Bit (SETBIT)

☐ ☐ ☒
 430 440 450

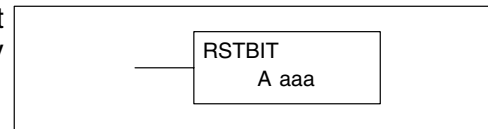
The Set Bit instruction sets a single bit to one within a range of V-memory locations.



Reset Bit (RSTBIT)

☐ ☐ ☒
 430 440 450

The Reset Bit instruction resets a single bit to zero within a range of V-memory locations.



The following description applies to both the Set Bit and Reset Bit table instructions.

Step 1: — Load the length of the table (number of V-memory locations) into the first level of the accumulator stack. This parameter must be a HEX value, 0 to FF.

Step 2: — Load the starting V-memory location for the table into the accumulator. This parameter must be a HEX value. You can use the LDA instruction to convert an octal address to hex.

Step 3: — Insert the Set Bit or Reset Bit instruction. This specifies the reference for the bit number of the bit you want to set or reset. The bit number is in octal, and the first bit in the table is number “0”.

Helpful hint: — Remember that each V-memory location contains 16 bits. So, the bits of the first word of the table are numbered from 0 to 17 octal. For example, if the table length is six words, then 6 words = (6 x 16) bits, = 96 bits (decimal), or 140 octal. The permissible range of bit reference numbers would be 0 to 137 octal. Flag 53 will be set if the bit specified is outside the range of the table.

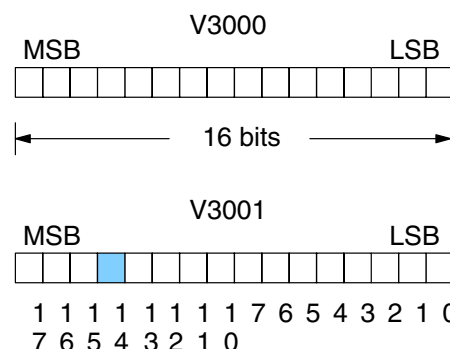
Operand Data Type		DL450 Range
		aaa
Vmemory	V	All (See p. 3-42)
Octal Address	O	0-7777

Discrete Bit Flags	Description
SP53	on when the bit number which is referenced in the Set Bit or Reset Bit exceeds the range of the table

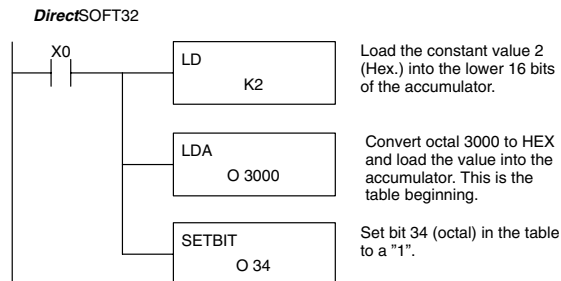
NOTE: Status flags are only valid until:

- the end of the scan
- or another instruction that uses the same flag is executed.

For example, suppose we have a table starting at V3000 that is two words long, as shown to the right. Each word in the table contains 16 bits, or 0 to 17 in octal. To set bit 12 in the second word, we use its octal reference (bit 14). Then we compute the bit's octal address from the start of the table, so $17 + 14 = 34$ octal. The following program shows how to set the bit as shown to a “1”.



In this ladder example, we will use input X0 to trigger the Set Bit operation. First, we will load the table length (2 words) into the accumulator stack. Next, we load the starting address into the accumulator. Since V3000 is an octal number we have to convert it to hex by using the LDA command. Finally, we use the Set Bit (or Reset Bit) instruction and specify the octal address of the bit (bit 34), referenced from the table beginning.



Handheld Programmer Keystrokes

STR	X(IN)	0	←						
LD	K(CON)	2	←						
LD	SHFT	A	OCT	3	0	0	0	←	
SET	SHFT	B	I	T	OCT	3	4	←	

Table Shift Left
(TSHFL)

✕

✕

✓

430440450

The Table Shift Left instruction shifts all the bits in a V-memory table to the left, the specified number of bit positions.

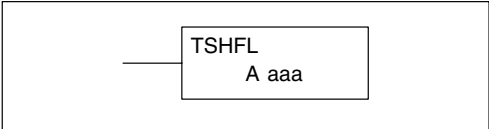


Table Shift Right
(TSHFR)

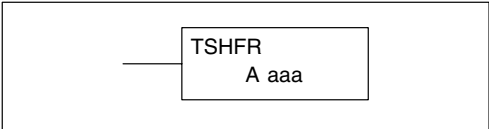
✕

✕

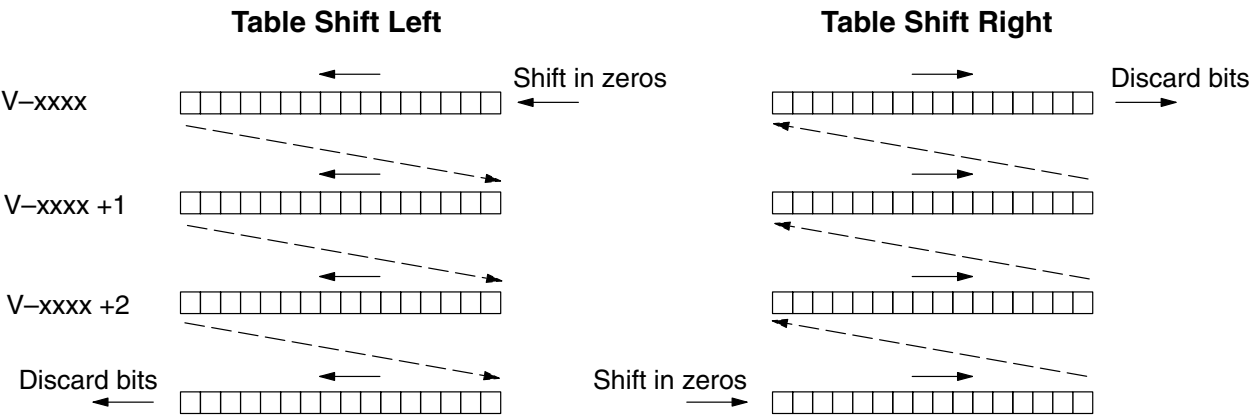
✓

430440450

The Table Shift Right instruction shifts all the bits in a V-memory table to the right, a specified number of bit positions.



The following description applies to both the Table Shift Left and Table Shift Right instructions. A table is just a range of V-memory locations. The Table Shift Left and Table Shift Right instructions shift bits serially throughout the entire table. Bits are shifted out the end of one word and into the opposite end of an adjacent word. At the ends of the table, bits are either discarded, or zeros are shifted into the table. The example tables below are arbitrarily four words long.



Step 1: — Load the length of the table (number of V-memory locations) into the first level of the accumulator stack. This parameter must be a HEX value, 0 to FF.

Step 2: — Load the starting V-memory location for the table into the accumulator. This parameter must be a HEX value. You can use the LDA instruction to convert an octal address to hex.

Step 3: —Insert the Table Shift Left or Table shift Right instruction. This specifies the number of bit positions you wish to shift the entire table. The number of bit positions must be in octal.

Helpful hint: — Remember that each V-memory location contains 16 bits. So, the bits of the first word of the table are numbered from 0 to 17 octal. If you want to shift the entire table by 20 bits, that is 24 octal. Flag 53 will be set if the number of bits to be shifted is larger than the total bits contained within the table. Flag 67 will be set if the last bit shifted (just before it is discarded) is a “1”.

Operand Data Type		DL450 Range
		aaa
Vmemory	V	All (See p. 3-42)

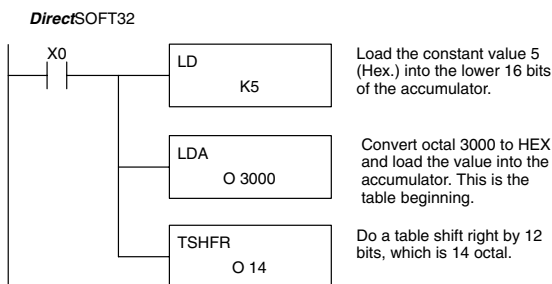
Discrete Bit Flags	Description
SP53	on when the number of bits to be shifted is larger than the total bits contained within the table
SP67	on when the last bit shifted (just before it is discarded) is a "1"

NOTE: Status flags are only valid until:
— the end of the scan
— or another instruction that uses the same flag is executed.

The example table to the right contains BCD data as shown (for demonstration purposes). Suppose we want to do a table shift right by 3 BCD digits (12 bits). Converting to octal, 12 bits is 14 octal. Using the Table Shift Right instruction and specifying a shift by octal 14, we have the resulting table shown at the far right. Notice that the 2-3-4 sequence has been discarded, and the 0-0-0 sequence has been shifted in at the bottom.

V3000	V3000
1 2 3 4	6 7 8 1
5 6 7 8	1 2 2 5
1 1 2 2	3 4 4 1
3 3 4 4	5 6 6 3
5 5 6 6	0 0 0 5

The following ladder example assumes the data at V3000 to V3004 already exists as shown above. We will use input X0 to trigger the Table Shift Right operation. First, we will load the table length (5 words) into the accumulator stack. Next, we load the starting address into the accumulator. Since V3000 is an octal number we have to convert it to hex by using the LDA command. Finally, we use the Table Shift Right instruction and specify the number of bits to be shifted (12 decimal), which is 14 octal.



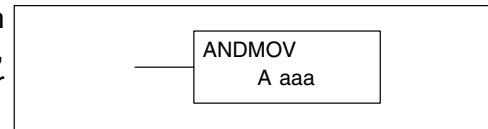
Handheld Programmer Keystrokes

STR	X(IN)	0	←
LD	K(CON)	5	←
LD	SHFT	A	OCT
SHFT	T	S	H
		F	R
		OCT	1
			4
			←

AND Move (ANDMOV)

☐ 430
 ☐ 440
 ☒ 450

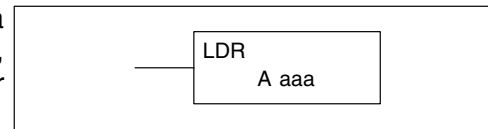
The AND Move instruction copies data from a table to the specified memory location, ANDing each word with the accumulator data as it is written.



OR Move (ORMOV)

☐ 430
 ☐ 440
 ☒ 450

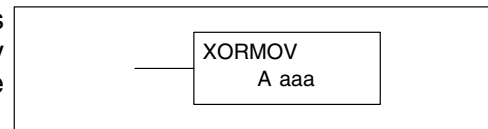
The Or Move instruction copies data from a table to the specified memory location, ORing each word with the accumulator contents as it is written.



Exclusive OR Move (XORMOV)

☐ 430
 ☐ 440
 ☒ 450

The Exclusive OR Move instruction copies data from a table to the specified memory location, XORing each word with the accumulator value as it is written.



The following description applies to the AND Move, OR Move, and Exclusive OR Move instructions. A table is just a range of V-memory locations. These instructions copy the data of a table to another specified location, performing a logical operation on each word with the accumulator contents as the new table is written.

Step 1: — Load the length of the table (number of V-memory locations) into the first level of the accumulator stack. This parameter must be a HEX value, 0 to FF.

Step 2: — Load the starting V-memory location for the table into the accumulator. This parameter must be a HEX value. You can use the LDA instruction to convert an octal address to hex.

Step 3: — Load the BCD/hex bit pattern into the accumulator which will be logically combined with the table contents as they are copied.

Step 4: — Insert the AND Move, OR Move, or XOR Move instruction. This specifies the starting location of the copy of the original table. This new table will automatically be the same length as the original table.

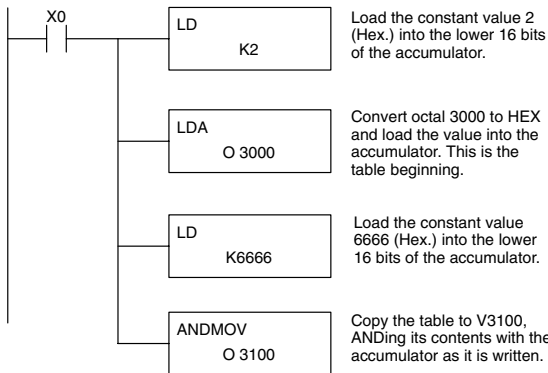
Operand Data Type		DL450 Range
		aaa
Vmemory	V	All (See p. 3-42)

The example table to the right contains BCD data as shown (for demonstration purposes). Suppose we want to move a table of two words at V3000 and AND it with K6666. The copy of the table at V3100 shows the result of the AND operation for each word.



The program on the next page performs the ANDMOV operation example above. It assumes that the data in the table at V3000 – V3001 already exists. First we load the table length (two words) into the accumulator. Next we load the starting address of the source table, using the LDA instruction. Then we load the data into the accumulator to be ANDed with the table. In the ANDMOV command, we specify the table destination, V3100.

DirectSOFT32



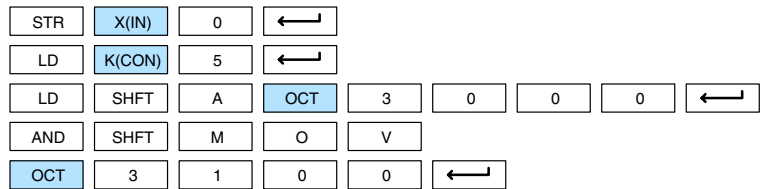
Load the constant value 2 (Hex.) into the lower 16 bits of the accumulator.

Convert octal 3000 to HEX and load the value into the accumulator. This is the table beginning.

Load the constant value 6666 (Hex.) into the lower 16 bits of the accumulator.

Copy the table to V3100, ANDing its contents with the accumulator as it is written.

Handheld Programmer Keystrokes

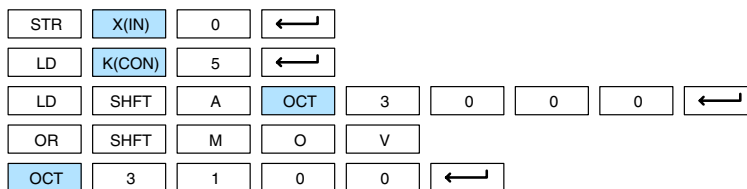


The example to the right shows a table of two words at V3000 and logically ORs it with K8888. The copy of the table at V3100 shows the result of the OR operation for each word.

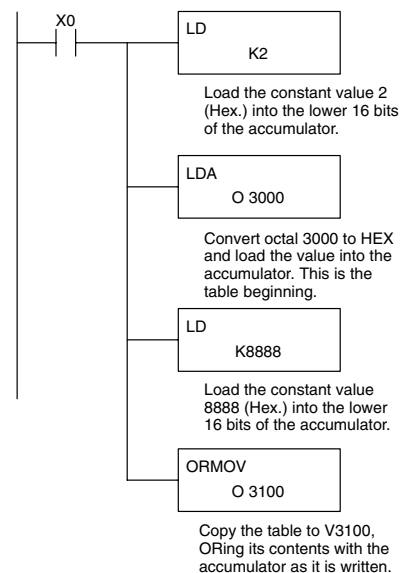


The program to the right performs the ORMOV example above. It assumes that the data in the table at V3000 – V3001 already exists. First we load the table length (two words) into the accumulator. Next we load the starting address of the source table, using the LDA instruction. Then we load the data into the accumulator to be ORed with the table. In the ORMOV command, we specify the table destination, V3100.

Handheld Programmer Keystrokes



DirectSOFT32



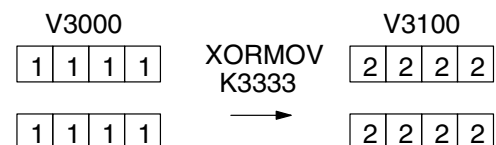
Load the constant value 2 (Hex.) into the lower 16 bits of the accumulator.

Convert octal 3000 to HEX and load the value into the accumulator. This is the table beginning.

Load the constant value 8888 (Hex.) into the lower 16 bits of the accumulator.

Copy the table to V3100, ORing its contents with the accumulator as it is written.

The example to the right shows a table of two words at V3000 and logicall XORs it with K3333. The copy of the table at V3100 shows the result of the XOR operation for each word.

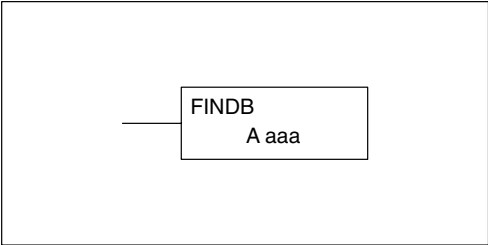


The ladder program example for the XORMOV is similar to the one above for the ORMOV. Just use the XORMOV instruction. On the handheld programmer, you must use the SHFT key and spell "XORMOV" explicitly.

Find Block
(FINDB)

430 440 450

The Find Block instruction searches for an occurrence of a specified block of values in a V-memory table. The function parameters are loaded into the first and second levels of the accumulator stack and the accumulator by three additional instructions. If the block is found, its starting address will be stored in the accumulator. If the block is not found, flag SP53 will be set.



Operand Data Type		DL450 Range
		aaa
Vmemory	V	All (See p. 3-42)
Vmemory	P	All (See p. 3-42)

Discrete Bit Flags	Description
SP53	on when the Find Block instruction was executed but did not find the block of data in table specified

- The steps listed below are the steps necessary to program the Find Block function.
- Step 1: — Load the number of bytes in the block to be located. This parameter must be a HEX value, 0 to FF.
- Step 2: — Load the length of a table (number of words) to be searched. The Find Block will search multiple tables that are adjacent in V-memory. This parameter must be a HEX value, 0 to FF.
- Step 3: — Load the ending location for all the tables into the accumulator. This parameter must be a HEX value. You can use the LDA instruction to convert an octal address to hex.
- Step 4: — Load the table starting location for all the tables into the accumulator. This parameter must be a HEX value. You can use the LDA instruction to convert an octal address to hex.
- Step 5: — Insert the Find Block instruction. This specifies the starting location of the block of data you are trying to locate.

Start Addr.

Table 1
Table 2
Table 3
Table n

Number of words

Start Addr.



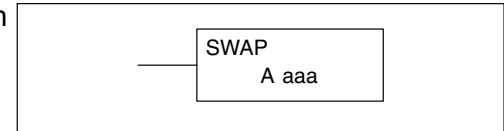
Number of bytes

End Addr.

Swap (SWAP)

✕ ✕ ✓
430 440 450

The Swap instruction exchanges the data in two tables of equal length.



The following description applies to both the Set Bit and Reset Bit table instructions.

Step 1: — Load the length of the tables (number of V-memory locations) into the first level of the accumulator stack. This parameter must be a HEX value, 0 to FF. Remember that the tables must be of equal length.

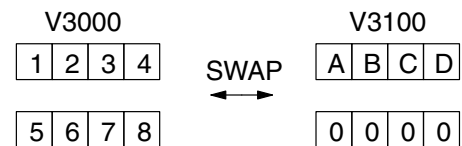
Step 2: — Load the starting V-memory location for the first table into the accumulator. This parameter must be a HEX value. You can use the LDA instruction to convert an octal address to hex.

Step 3: — Insert the Swap instruction. This specifies the starting address of the second table. This parameter must be a HEX value. You can use the LDA instruction to convert an octal address to hex.

Helpful hint: — The data swap occurs within a single scan. If the instruction executes on multiple consecutive scans, it will be difficult to know the actual contents of either table at any particular time. So, remember to swap just on a single scan.

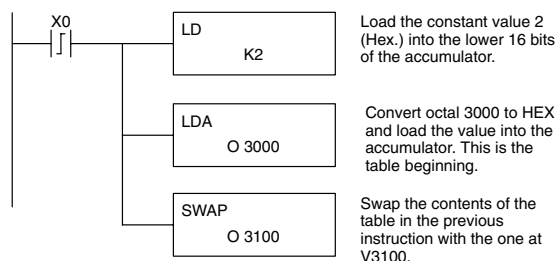
Operand Data Type	DL450 Range
	aaa
Vmemory V	All (See p. 3-42)

The example to the right shows a table of two words at V3000. We will swap its contents with another table of two words at 3100 by using the Swap instruction. The required ladder program is given below.

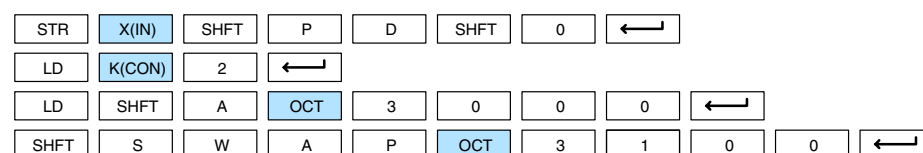


The example program below uses a PD contact (triggers for one scan for off-to-on transition). First, we load the length of the tables (two words) into the accumulator. Then we load the address of the first table (V3000) into the accumulator using the LDA instruction, converting the octal address to hex. Note that it does not matter which table we declare “first”, because the swap results will be the same.

DirectSOFT32



Handheld Programmer Keystrokes

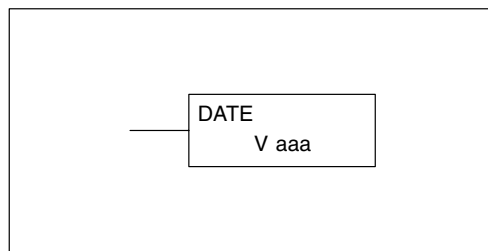


Clock / Calendar Instructions

Date (DATE)

☒ 430
 ☒ 440
 ☒ 450

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).



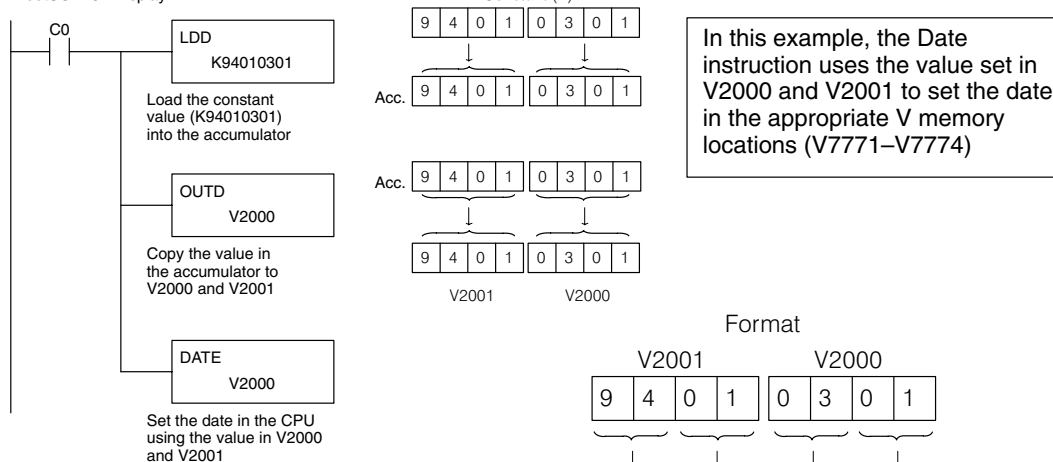
Date	Range	V Memory Location (BCD) (READ Only)
Year	0–99	V7774
Month	1–12	V7773
Day	1–31	V7772
Day of Week	0–06	V7771

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

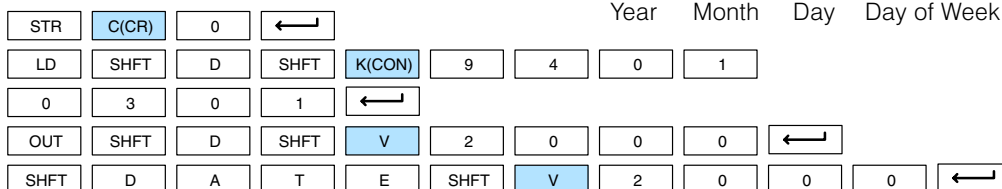
Operand Data Type	DL440 Range	DL450 Range
A	aaa	aaa
Vmemory V	All (See p. 3–41)	All (See p. 3–42)

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

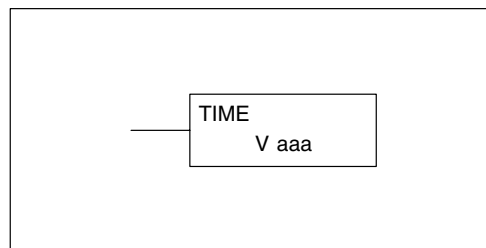
DirectSOFT32 Display



Handheld Programmer Keystrokes

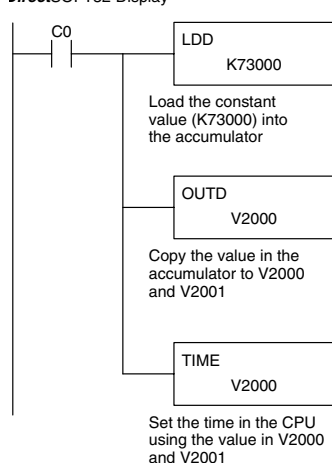


✗	✓	✓
430	440	450

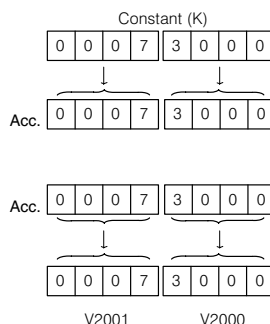


Date	Range	V Memory Location (BCD) (READ Only)
1/100 seconds (10ms)	0–99	V7747
Seconds	0–59	V7766
Minutes	0–59	V7767
Hour	0–23	V7770

Operand Data Type	DL440 Range	DL440 Range
A	aaa	aaa
Vmemory V	All (See p. 3–41)	All (See p. 3–41)



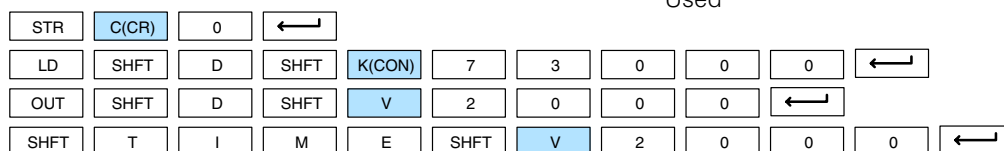
The Time instruction uses the value set in V2000 and V2001 to set the time in the appropriate V memory locations (V7766–V7770).



Format

V2001				V2000			
0	0	0	7	3	0	0	0
↓		↓		↓		↓	
Not Used		Hour		Minutes		Seconds	

Handheld Programmer Keystrokes



CPU Control Instructions

No Operation (NOP)

✓ 430 ✓ 440 ✓ 450

The No Operation is an empty (not programmed) memory location. These instructions are just place-holders in the program. So, you will not need to program them, because they automatically appear after the end of the program.

—(NOP)

DirectSOFT32 Display



Handheld Programmer Keystrokes

SHIFT N O P ←

End (END)

✓ 430 ✓ 440 ✓ 450

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

—(END)

DirectSOFT32 Display



Handheld Programmer Keystrokes

END ←

Stop (STOP)

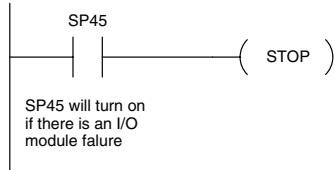
✓ 430 ✓ 440 ✓ 450

The Stop instruction changes the operational mode of the CPU from Run to Program (Stop) mode. This instruction is typically used to stop PLC operation in a shutdown condition such as a I/O module failure.

—(STOP)

In the following example, when SP45 comes on indicating a I/O module failure, the CPU will stop operation and switch to the program mode.

DirectSOFT32 Display



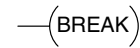
Handheld Programmer Keystrokes

STR SPCL 4 5 ←
SHIFT S T O P ←

**Break
(BREAK)**

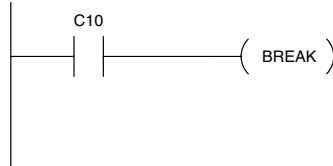
✕	✓	✓
430	440	450

The Break instruction changes the operational mode of the CPU from Run to the Test Program mode. This instruction is typically used to aid in debugging an application program. The Break instruction allows V memory and image register data to be retained where it would be normally cleared with the Stop instruction or a normal Run to Program transition.

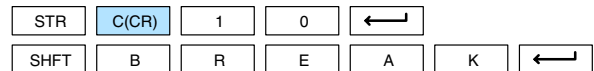


In the following example when C10 turns on, the CPU will stop operation and switch to the Test program mode.

DirectSOFT32 Display



Handheld Programmer Keystrokes

**Reset Watch Dog
Timer
(RSTWT)**

✓	✓	✓
430	440	450

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.

DirectSOFT32 Display



Handheld Programmer Keystrokes



Program Control Instructions

Goto / Label (GOTO / LBL)

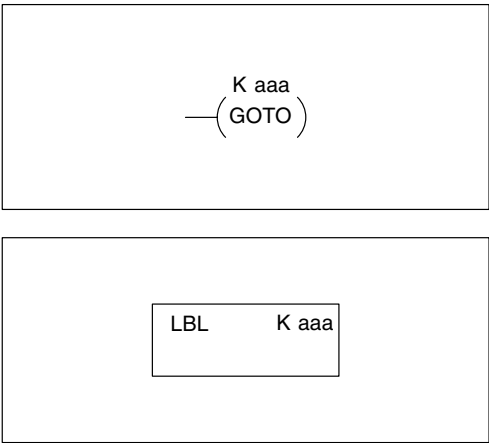
✕

✓

✓

430440450

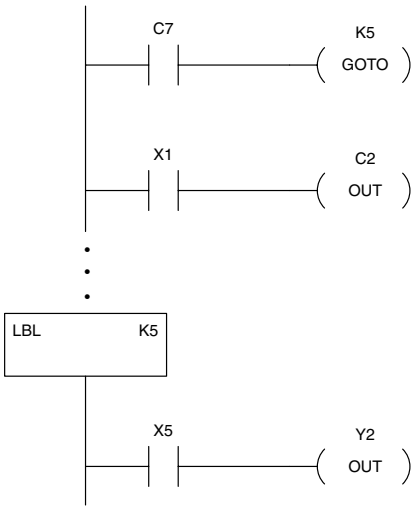
The Goto / Label skips all instructions between the Goto and the corresponding LBL instruction. The operand value for the Goto and the corresponding LBL instruction are the same. The logic between Goto and LBL instruction is not executed when the Goto instruction is enabled. Up to 128 Goto instructions and 64 LBL instructions can be used in the program.



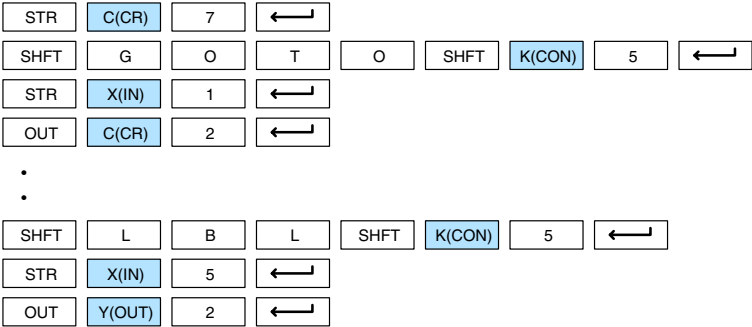
Operand Data Type		DL440 Range	DL450 Range
		aaa	aaa
Constant	K	1-FFFF	1-FFFF

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

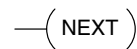
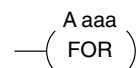
DirectSOFT32 Display



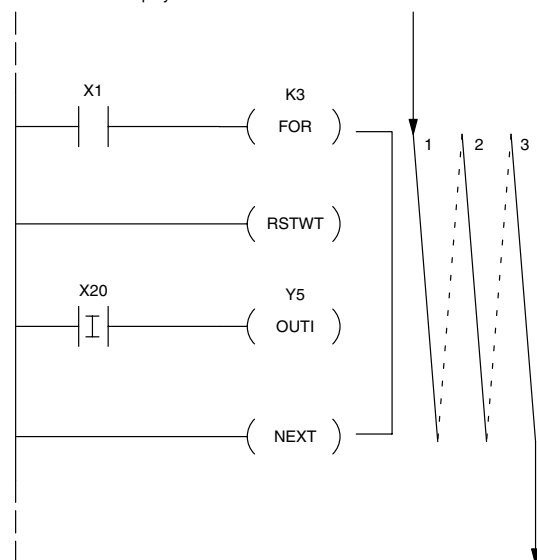
Handheld Programmer Keystrokes



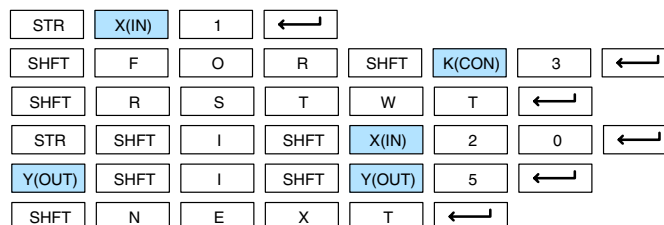
✗	✓	✓
430	440	450



Operand Data Type	DL440 Range	DL450 Range
A	aaa	aaa
Vmemory	V	All (See p. 3–41)
Constant	K	1–9999



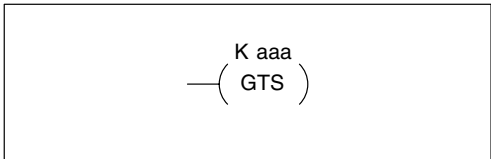
Handheld Programmer Keystrokes



Goto Subroutine (GTS)

430 440 450

The Goto Subroutine instruction allows a section of ladder logic to be placed outside the main body of the program execute only when needed. There can be a maximum of 192 (DL440) and an unlimited amount for DL450 GTS instructions.



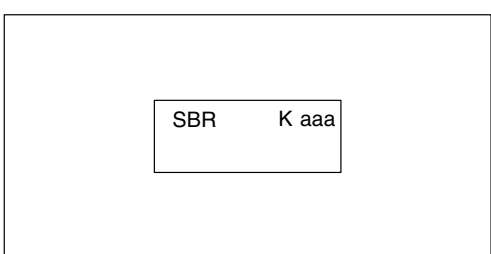
Upon completion of executing the subroutine, program execution returns to the main program immediately after the GTS instruction. GTS instructions can be nested up to 8 levels. An error E412 will occur if the maximum limits are exceeded. Typically this will be used in an application where a block of program logic may be slow to execute and is not required to execute every scan.

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

Subroutine (SBR)

430 440 450

The subroutine label and all associated logic is placed after the End statement in the program. There can be a maximum of 64 (DL440) and 256 (DL450) SBR instructions used in a program. When the subroutine is called from the main program, the CPU will execute the subroutine (SBR) with the same constant number (K) as the GTS instruction which called the subroutine.



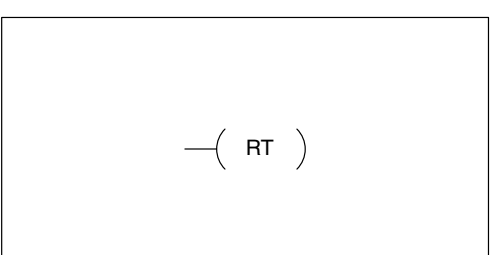
By placing code in a subroutine it is only scanned and executed when needed since it resides after the End instruction. Code which is not scanned does not impact the overall scan time of the program.

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

Subroutine Return (RT)

430 440 450

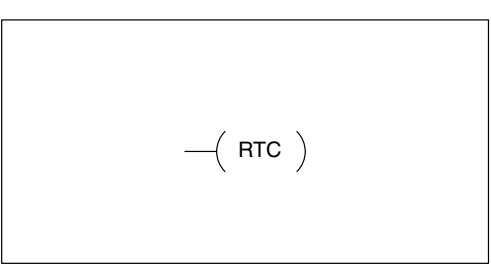
When a Subroutine Return is executed in the subroutine the CPU will return to the point in the main body of the program from which it was called. The Subroutine Return is used as termination of the subroutine which must be the last instruction in the subroutine and is a stand alone instruction (no input contact on the rung).



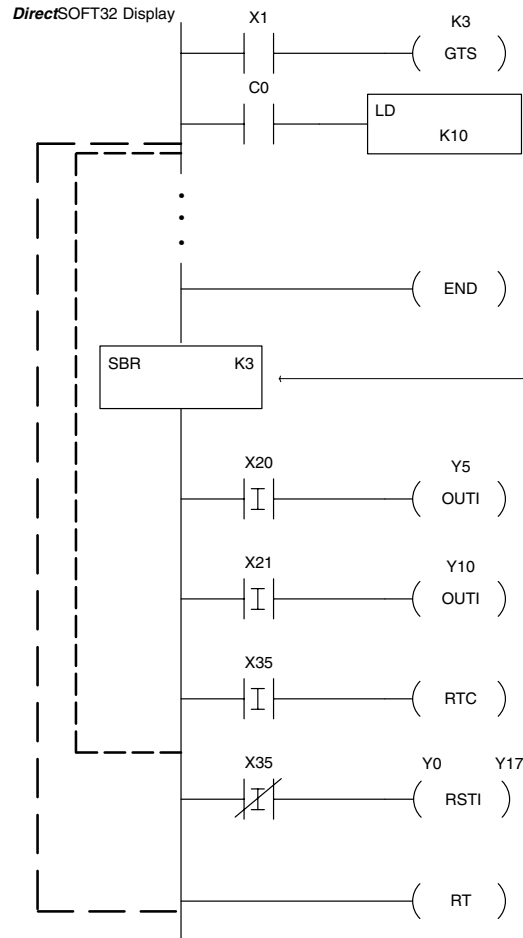
Subroutine Return Conditional (RTC)

430 440 450

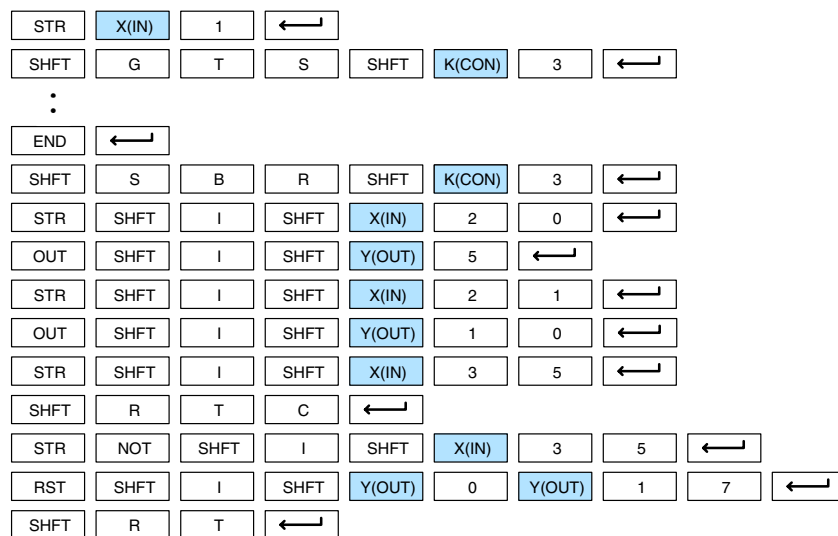
The Subroutine Return Conditional instruction is a optional instruction used with a input contact to implement a conditional return from the subroutine. The Subroutine Return (RT) is still required for termination of the Subroutine.



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

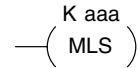


Master Line Set (MLS)



430 440 450

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.



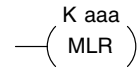
Operand Data Type	DL430 Range	DL440 Range	DL450 Range
	aaa	aaa	aaa
Constant K	1-7	1-7	1-7

Master Line Reset (MLR)



430 440 450

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	DL430 Range	DL440 Range	DL450 Range
	aaa	aaa	aaa
Constant K	0-6	0-6	1-6

Understanding Master Control Relays



430 440 450

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

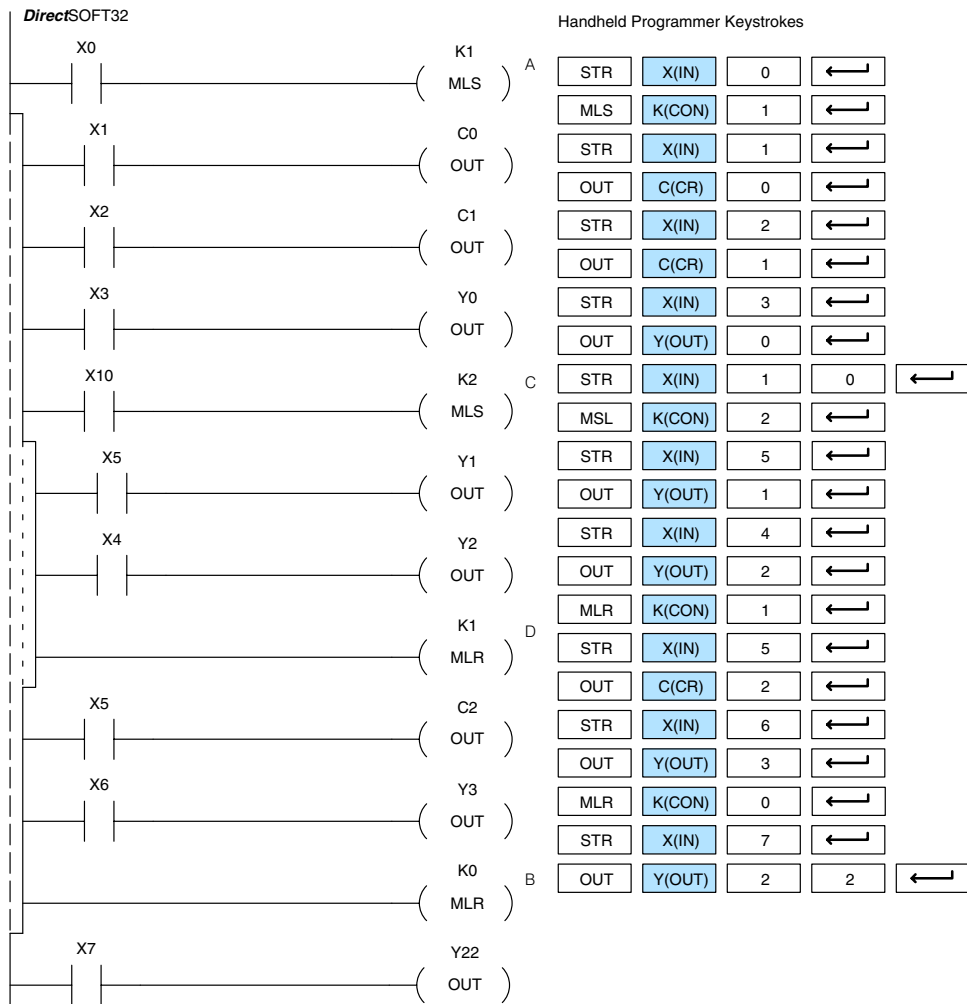
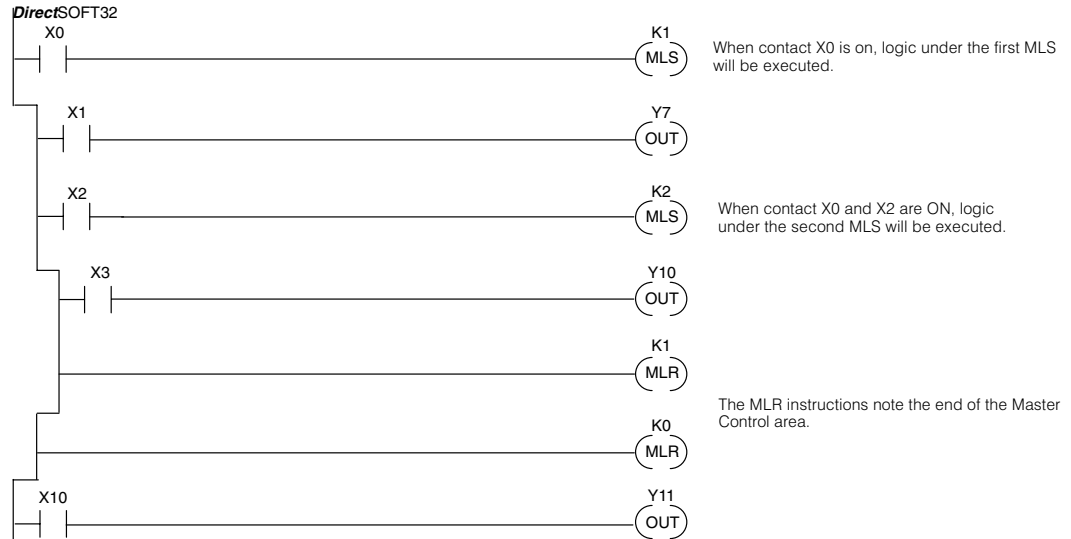
MLS/MLR Example

In the following MLS/MLR example, logic between the first MLS K1 (A) and MLR K0 (B) will only have power flow present at the power rail if input X0 is on. (Note, if X0 is off the logic will still be scanned, but there is no power flow.) The logic between the MLS K2 (C) and MLR K1 (D) will only have power flow present at the power rail if input X10 and X0 are on. The last rung is not controlled by either of the MLS coils and always has power flow present at the beginning of the rung.

Remember, the MLS / MLR instructions control the power flow between the power rails. It does not control the execution of the instructions. The instructions are still executed, but since there is no power flow, the logic cannot turn on the output coils. Consider the following case for our example.

1. X0 is off, which means that there is no power flow at the second rung.
2. You use a programming device to turn on C0.

Since there is no power flow at the second rung (STR X1, OUT C0), then you would expect that C0 would remain on, since you turned it on with the programming device. However, the MLS instruction does not mean that the instructions within the zone of control are not executed. They *are* in fact executed, but with no power flow. So in our case, C0 would be turned off when the rung was executed. This is because the CPU sees that there is no power flow present at the rung. When it executes this rung, it will turn off C0.



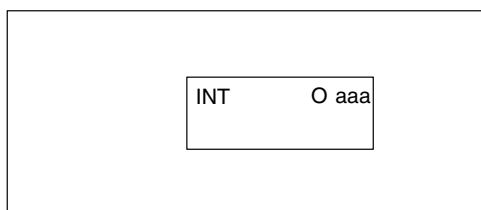
Interrupt Instructions

Interrupt (INT)



430 440 450

The Interrupt instruction allows a section of ladder logic to be placed outside the main body of the program and executed when needed. Interrupts can be called from the program or an interrupt module can be installed in slot 0 to provide 8 interrupt inputs.



One interrupt module can be installed in a DL430 system (X0–X7) and two interrupt modules can be installed in a DL440 or DL450 System (X0–X7, and X20–X27). Remember, the interrupt modules consume 16 points.

The two software interrupts use interrupt #16 and #17 which means the hardware interrupts #16 and #17 and the software interrupt cannot be used together.

Typically, interrupts will be used in an application where a fast response to an input is needed or a program section needs to execute faster than the normal CPU scan. The interrupt label and all associated logic must be placed after the End statement in the program. When the interrupt routine is called from the interrupt module or software interrupt, the CPU will complete execution of the instruction it is currently processing in ladder logic then execute the designated interrupt routine. There are two software interrupts and INT 16 and INT 17. The program execution will continue from where it was before the interrupt occurred once the interrupt is serviced.

The software interrupts are setup by programming the interrupt times in V736 and V737. The valid range is 3–999ms. The value must be a BCD value. The interrupt will not execute if the value is out of range.

See the example program of a software interrupt.

Operand Data Type		DL430 Range	DL440 Range	DL450 Range
		aaa	aaa	aaa
Constant	O	0–7	0–17	0–17

Software		DL430		DL440/450	
Interrupt Input	Interrupt Routine	Interrupt Input	Interrupt Routine	Interrupt Input	Interrupt Routine
—	—	X0	INT 0	X0	INT 0
—	—	X1	INT 1	X1	INT 1
—	—	X2	INT 2	X2	INT 2
—	—	X3	INT 3	X3	INT 3
—	—	X4	INT 4	X4	INT 4
—	—	X5	INT 5	X5	INT 5
—	—	X6	INT 6	X6	INT 6
—	—	X7	INT 7	X7	INT 7
—	—	—	—	X20	INT 10
—	—	—	—	X21	INT 11
—	—	—	—	X22	INT 12
—	—	—	—	X23	INT 13
—	—	—	—	X24	INT 14
—	—	—	—	X25	INT 15
V736 sets interrupt time	INT 16	—	—	X26 (cannot be used along with s/w interrupt)	INT 16
V737 sets interrupt time	INT 17	—	—	X27 (cannot be used along with s/w interrupt)	INT 17

2nd Module

**Interrupt Return
(IRT)**

430 440 450

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)**

430 440 450

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

**Enable Interrupts
(ENI)**

430 440 450

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)**

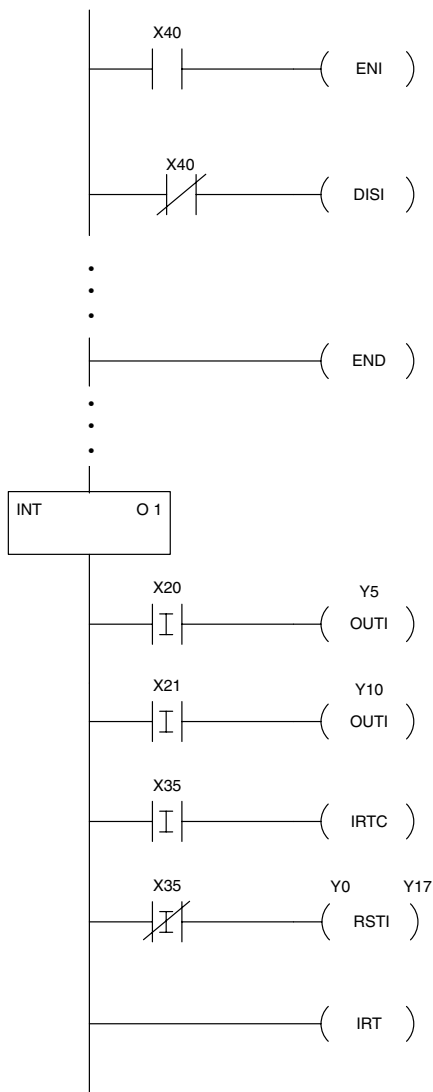
430 440 450

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 Interrupt
Module**

In the following example, when X40 is on, the interrupts will be enabled. When X40 is off the interrupts will be disabled. When an interrupt signal X1 is received the CPU will jump to the interrupt label INT O 1. The application ladder logic in the interrupt routine will be performed. If X35 is on the CPU will return to the main program with the IRTC instruction. If X35 is not on Y0–Y17 will be reset to off and then the CPU will return to the main body of the program.

DirectSOFT32 Display

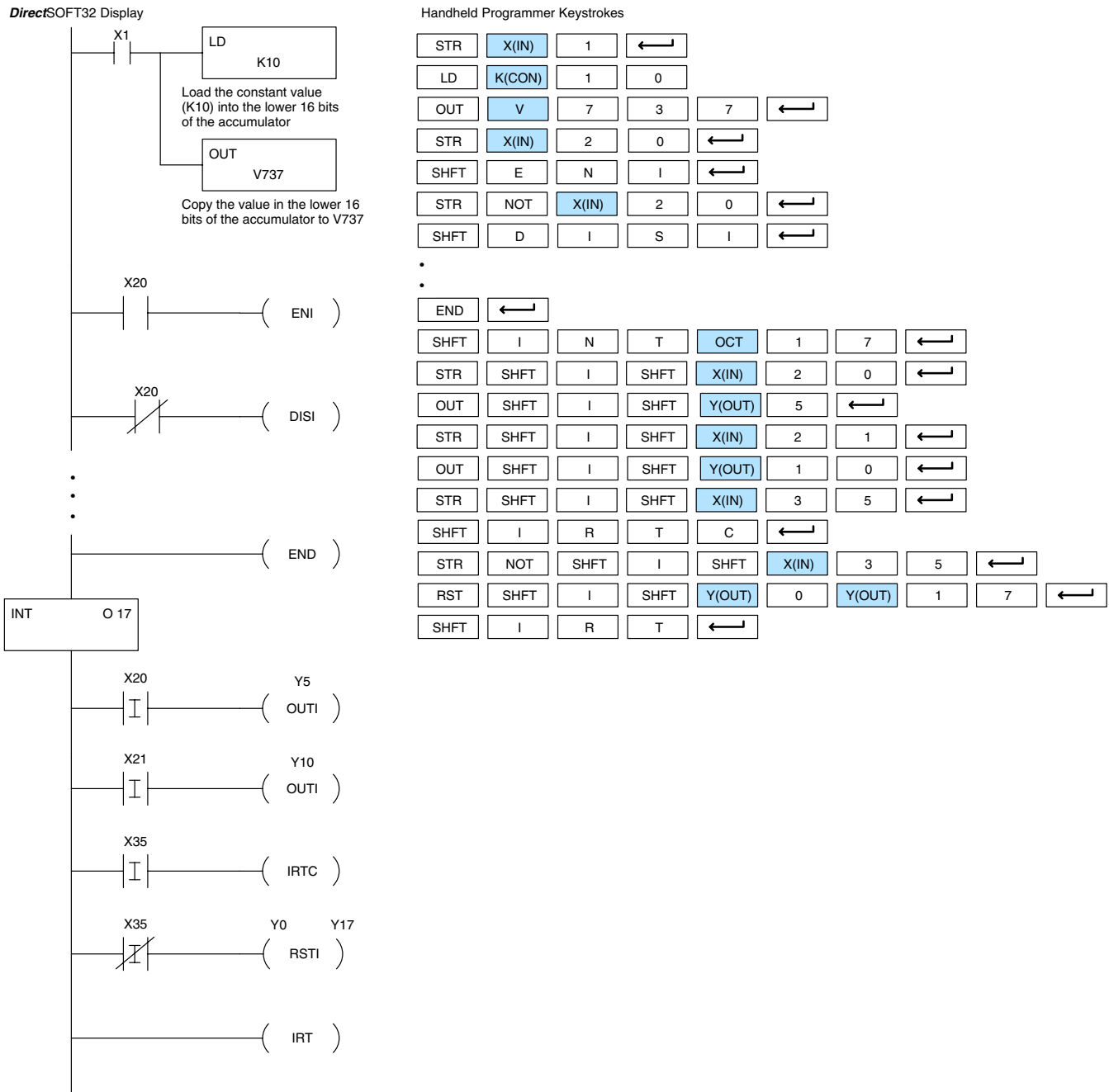


Handheld Programmer Keystrokes

STR	X(IN)	4	0	←					
SHFT	E	N	I	←					
STR	NOT	X(IN)	1	7	←				
SHFT	D	I	S	I	←				
.									
.									
END	←								
SHFT	I	N	T	OCT	1	←			
STR	SHFT	I	SHFT	X(IN)	2	0	←		
OUT	SHFT	I	SHFT	Y(OUT)	5	←			
STR	SHFT	I	SHFT	X(IN)	2	1	←		
OUT	SHFT	I	SHFT	Y(OUT)	1	0	←		
STR	SHFT	I	SHFT	X(IN)	3	5	←		
SHFT	I	R	T	C	←				
STR	NOT	SHFT	I	SHFT	X(IN)	3	5	←	
RST	SHFT	I	SHFT	Y(OUT)	0	Y(OUT)	1	7	←
SHFT	I	R	T	←					

Interrupt Example for Software Interrupt

In the following example, when X1 is on, the value 10 is copied to V737. This value sets the software interrupt to 10 ms. When X20 turns on, the interrupt will be enabled. When X20 turns off, the interrupts will be disabled. Every 10 ms the CPU will jump to the interrupt label INT O 17. The application ladder logic in the interrupt routine will be performed. If X35 is on the CPU will return to the main program with the IRTC instruction. If X35 is not on Y0–Y17 will be reset to off and then the CPU will return to the main body of the program when the IRT instruction is executed. The software interrupt is limited to the range 3 – 999 milliseconds. Entering a 0, 1, or 2 will yield an interrupt time of 3 milliseconds.



Intelligent I/O Instructions

Read from Intelligent Module (RD)



430 440 450

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.

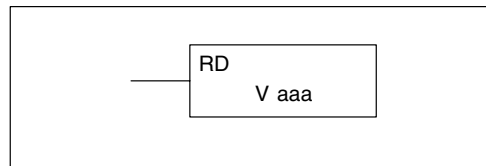
Step 1: — Load the base number (0–3) into the first byte and the slot number (0–7) into the second byte of the second level of the accumulator stack.

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.



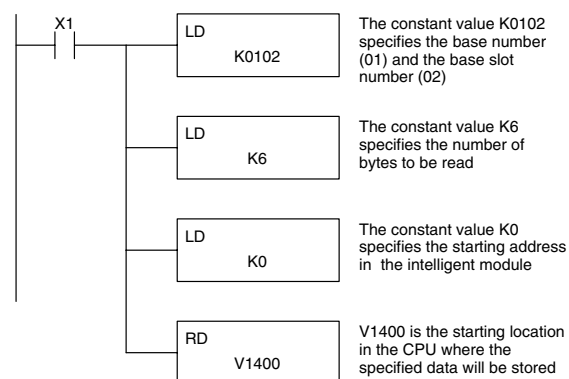
Operand Data Type	DL430 Range	DL440 Range	DL450 Range
	aaa	aaa	aaa
Vmemory V	All (See p. 3–40)	All (See p. 3–41)	All (See p. 3–42)

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

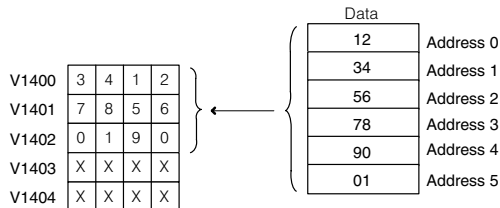
NOTE: Status flags are valid only until another instruction uses the same flag.

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

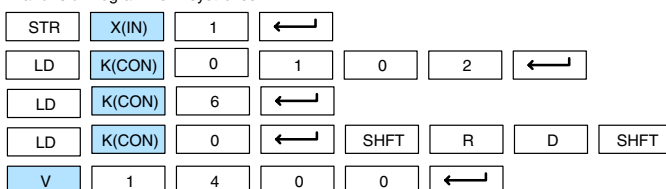
DirectSOFT32 Display



CPU Intelligent Module



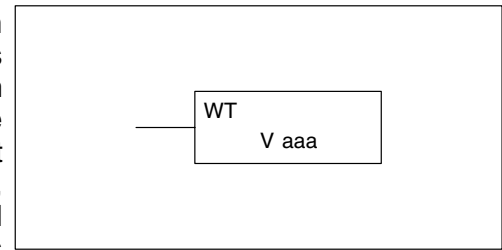
Handheld Programmer Keystrokes



Write to Intelligent Module (WT)


430 440 450

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.

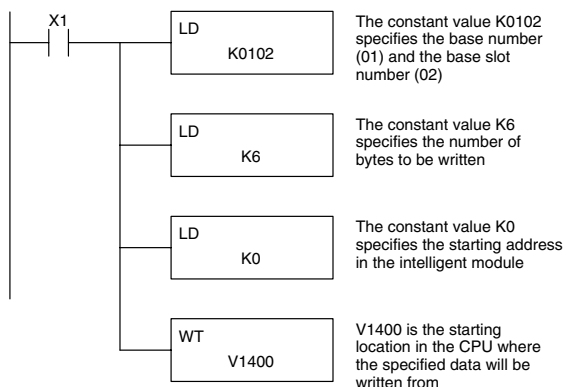
Operand Data Type	DL430 Range	DL440 Range	DL450 Range
	aaa	aaa	aaa
Vmemory V	All (See p. 3–40)	All (See p. 3–41)	All (See p. 3–42)

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

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

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

DirectSOFT32 Display



CPU

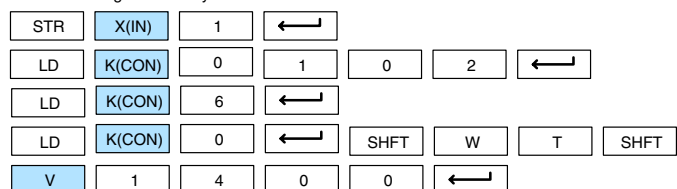
Intelligent Module

V1377	X	X	X	X
V1400	3	4	1	2
V1401	7	8	5	6
V1402	0	1	9	0
V1403	X	X	X	X
V1404	X	X	X	X

Data

12	Address 0
34	Address 1
56	Address 2
78	Address 3
90	Address 4
01	Address 5

Handheld Programmer Keystrokes



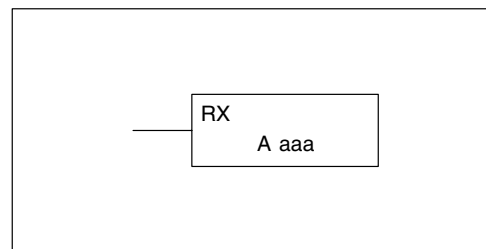
Network Instructions

Read from Network (RX)



430 440 450

The Read from Network instruction is used by the master device (CPU with a DCM) on a network to read a block of data from another CPU. The function parameters are loaded into the first and second level of the accumulator stack and the accumulator by three additional instructions. Listed below are the steps necessary to program the Read from Intelligent module function.



Step 1: — Load the slave address (1–90 BCD) into the first byte and the slot number of the master DCM (0–7) into the second byte of the second level of the accumulator stack. (Note: address 0 is only valid for peer stations.)

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

Step 3: — Load the address where you want to store the data in the master station. The address must be specified in HEX.

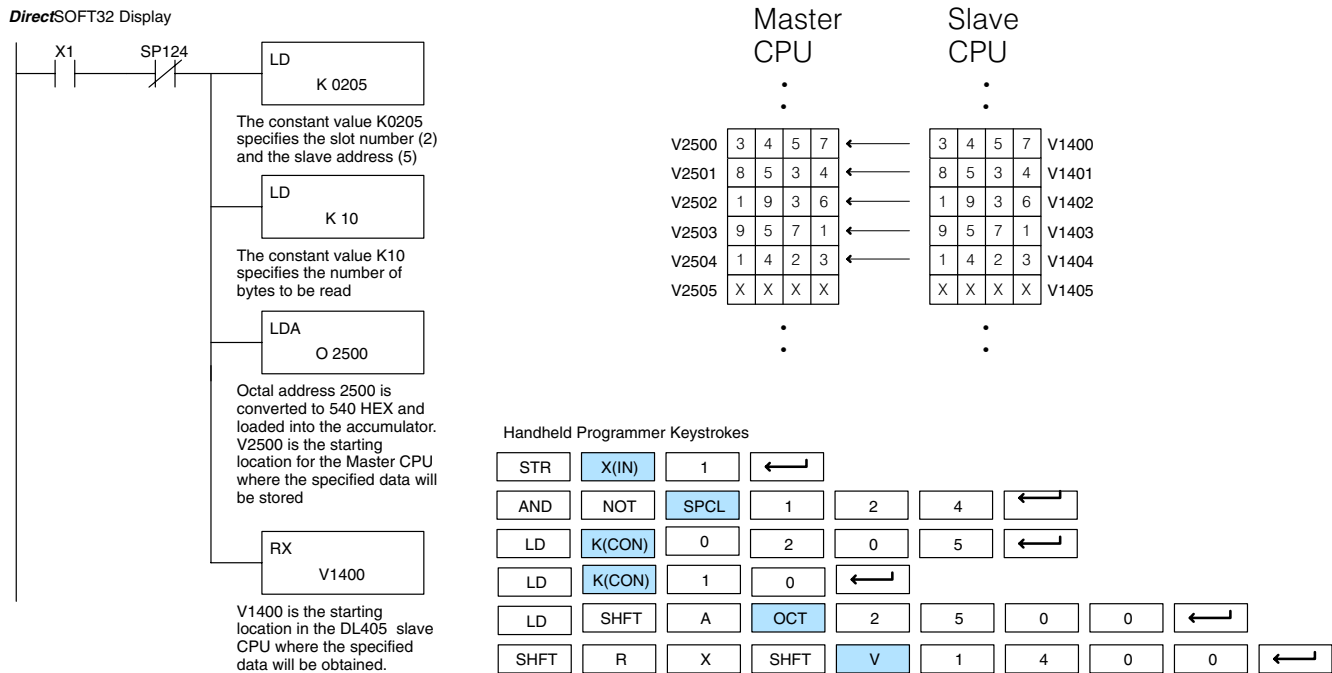
Step 4: — Insert the RX instruction and specify the starting V memory location (Aaaa) in the slave CPU where the data will be obtained.

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

Operand Data Type		DL430 Range	DL440 Range	DL450 Range
A		aaa	aaa	aaa
Vmemory	V	All (See p. 3–40)	All (See p. 3–41)	All (See p. 3–42)
Inputs	X	0–477	0–477	0–1777
Outputs	Y	0–477	0–477	0–1777
Control Relays	C	0–737	0–1777	0–3777
Stage	S	0–577	0–1777	0–1777
Timer	T	0–177	0–377	0–377
Counter	CT	0–177	0–177	0–377
Special Relay	SP	0–137, 320–617	0–137 320–717	0–137 320–717
Global I/O	GX	0–777	0–1777	0–2777
Program Memory	LS	0–3583	0–7679 (7.5K program mem.) 0–15871 (15.5K program mem.)	0–7679 (7.5K program mem.) 0–15871 (15.5K program mem.)
Scratchpad	Z	0–FFFF	0–FFFF	0–FFFF

NOTE: If you are using CoProcessor™, Share Data Network, or DCM modules, please refer to their manuals for information on how to transfer data over the **DirectNET** network.

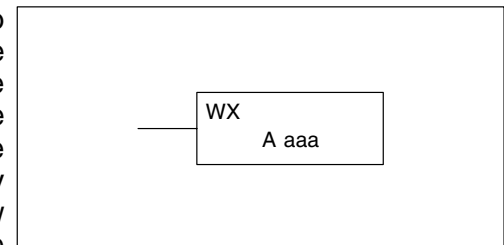
In the following example, when X1 is on and the module busy relay SP124 (see special relays) is not on, the RX instruction will access a DCM operating as a master in slot 2. Ten consecutive bytes of data (V1400 – V1404) will be read from a CPU at station address 5 and copied into V-memory locations V2500–V2504 in the CPU with the master DCM.



Write to Network (WX)

430 440 450

The Write to Network instruction is used to write a block of data from the master device (CPU with a DCM) to a slave device on the same network. The function parameters are loaded into the first and second level of the accumulator stack and the accumulator by three additional instructions. Listed below are the steps necessary to program the Write to Network function.



Step 1: — Load the slave address (1–90 BCD) into the first byte and the slot number of the master DCM (0–7) into the second byte of the second level of the accumulator stack. (Note: address 0 is only valid for peer stations.)

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

Step 3: — Load the address where you want to obtain the data in the master station. The address must be specified in HEX.

Step 4: — Insert the WX instruction and specify the starting V memory location (Aaaa) in the slave CPU where the data will be stored.

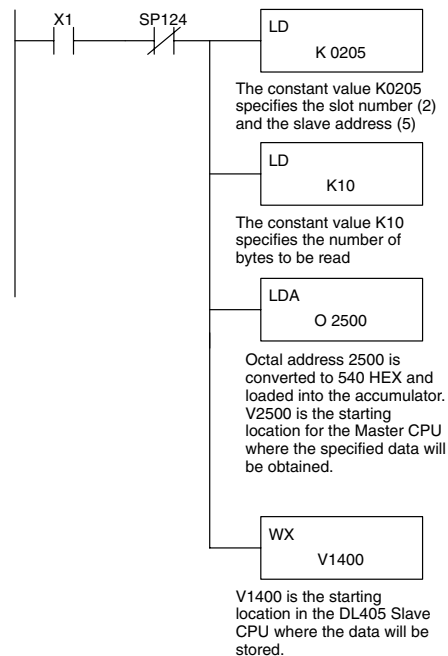
Helpful Hint: — For parameters that require HEX values, the LDA instruction can be used to convert an octal address to the HEX equivalent and load the value into the accumulator.

Operand Data Type		DL430 Range	DL440 Range	DL450 Range
A		aaa	aaa	aaa
Vmemory	V	All (See p. 3-40)	All (See p. 3-41)	All (See p. 3-42)
Inputs	X	0-477	0-477	0-1777
Outputs	Y	0-477	0-477	0-1777
Control Relays	C	0-737	0-1777	0-3777
Stage	S	0-577	0-1777	0-1777
Timer	T	0-177	0-377	0-377
Counter	CT	0-177	0-177	0-377
Special Relay	SP	0-137, 320-617	0-137 320-717	0-137 320-717
Global I/O	GX	0-777	0-1777	0-2777
Program Memory	\$	0-3585	0-7679 (7.5K program mem.) 0-15873 (15.5K program mem.)	0-7679 (7.5K program mem.) 0-15873 (15.5K program mem.)
Scratchpad	Z	0-FFFF	0-FFFF	0-FFFF

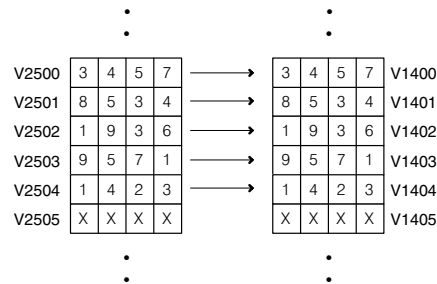
NOTE: If you are using CoProcessor™, Share Data Network, or DCM modules, please refer to their manuals for information on how to transfer data over the DirectNET network.

In the following example when X1 is on and the module busy relay SP124 (see special relays) is not on, the WX instruction will access a DCM operating as a master in slot 2. 10 consecutive bytes of data is read from the CPU at station address 5 and copied to Vmemory locations V1400-V1404 in the slave CPU.

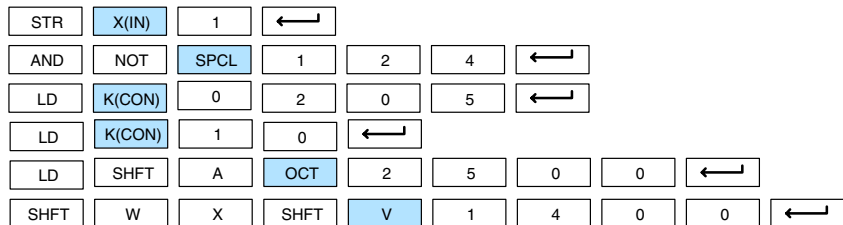
DirectSOFT32 Display



Master CPU Slave CPU



Handheld Programmer Keystrokes



Message Instructions

System Errors and Fault Messages

The DL405 CPUs provide error logging capabilities. There are certain predefined system error messages and codes, but you can also use the Fault instruction to create your own specific messages. The CPU logs the error, the date, and the time the error occurred. There are two separate tables that store this information.

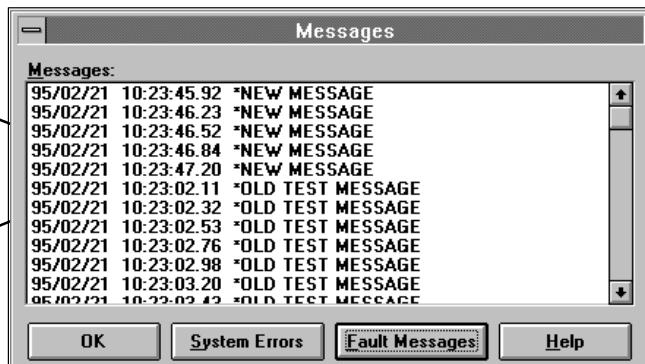
- **System Error Table** – both the DL430 and DL440 have several predefined system error codes. The DL430 can hold one error at a time. The DL440 can show up to 32 errors in an error table. When an error occurs, the error is loaded into the first available location. *Therefore, the most recent error may not appear in the top row of the table.* If the table is full when an error occurs, the oldest error is pushed (erased) from the table and the new error is inserted in the row.
- **Fault Message Table** – the DL430 and DL440 also allow you to build your own error codes and messages. These are called Fault Messages. With the DL430, you can only build numeric error codes. With the DL440, you can build error codes, or up to 16 messages that can contain up to 23-character alphanumeric characters. In either case, you can have up to 16 messages or codes shown in the table. When a message is triggered, it is put in the first available table location. *Therefore, the most recent error may not appear in the top row of the table.* If the table is full when an error occurs, the oldest error is pushed (erased) from the table and the new error is inserted in the row.

The following diagram shows an example of a the Fault Message table as shown in **DirectSOFT32**. You cannot view the entire table at one time with the handheld programmer. Instead, the messages automatically appear on the handheld programmer display as they occur. The message will remain on the display as long as the Fault instruction is being executed. You can also use an Auxiliary function (5C) to view the messages one at a time. (More on the handheld programmer display later.)

DL440 Error Msg. Example

Most recent message appears here, not at the top of the table.

Next message will show up in this row, which is now the oldest message.



There are several instructions that can be used in combination to create these error codes and messages.

- FAULT – Fault
- DLBL – Data Label
- ACON – ASCII Constant
- NCON – Numeric Constant

The next few pages provide details on these instructions. Also, at the end of this section, there are two examples that show how the instructions are used together.

Fault
(FAULT)

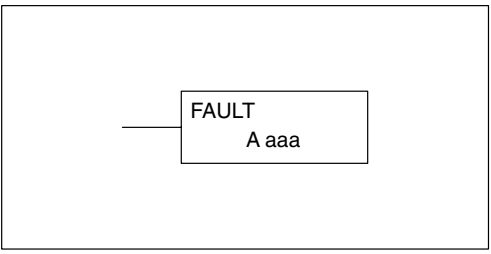
✗

✓

✓

430 440 450

In a DL440 or DL450, the Fault instruction is used to display a message or numeric error code on the handheld programmer, **DirectSOFT32** screen, or DV-1000 Operator Interface display. The message has a maximum of 23 characters and can be either Vmemory data, numerical constant data or ASCII text.



To display a value in a V memory location, specify the V memory location in the instruction.

To display ASCII or numeric data, you must use the DLBL (Data Label) instruction, and ACON (ASCII constant) or NCON (Numeric constant) instructions, in conjunction with the Fault instruction. In this case, you should specify the constant (K) value in the Fault instruction for the corresponding data label area that contains the ACON and/or NCON instructions.

Operand Data Type		DL440 Range	DL450 Range
A	A	aaa	aaa
Vmemory	V	All (See p. 3-41)	All (See p. 3-42)
Constant	K	1-FFFF	1-FFFF

Fault
(FAULT)

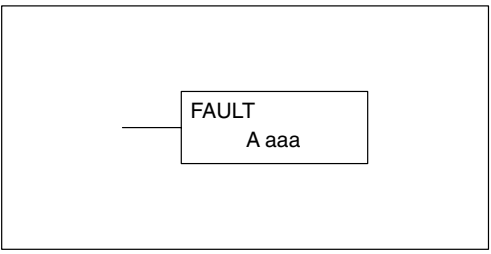
✓

✗

✗

430 440 450

In a DL430, the Fault instruction is used to display a numeric error code on the handheld programmer, **DirectSOFT32** screen, or DV-1000 Operator Interface display.



The error code may be obtained from a V memory location, or may be constructed by specifying a constant in the Fault instruction.

To display the value in a V memory location, specify the V memory location in the instruction. To display a numeric constant, specify the constant (K) value in the instruction.

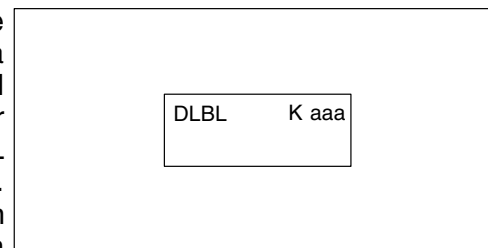
Operand Data Type		DL430 Range
A	A	aaa
Vmemory	V	All (See p. 3-40)
Constant	K	1-FFFF

NOTE: The DL430 *does not* support the necessary instructions to build alphanumeric error messages. You can only build error codes by obtaining the code from a V memory location, or by specifying a constant in the Fault instruction.

Data Label (DLBL)

✕ ✓ ✓
430 440 450

The Data Label instruction marks the beginning of an ASCII / numeric data area and is typically used with ACON and NCON instructions. DLBLs are programmed after the End statement. A maximum of 64 DLBL instructions can be used in a program. Multiple NCONs and ACONs can be used in a DLBL area. Examples are shown later in this section.



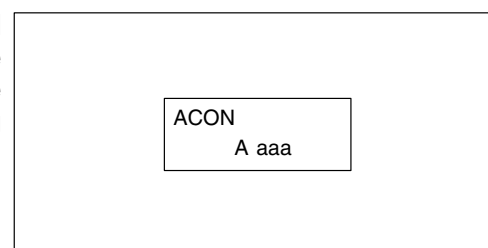
Operand Data Type	DL430 Range	DL440 Range	DL450 Range
	aaa	aaa	aaa
Constant K	—	1-FFFF	1-FFFF

ASCII Constant (ACON)

✕ ✓ ✓
430 440 450

The ASCII Constant instruction is used with the DLBL instruction to store ASCII text for use with other instructions. The instruction is utilized differently between the handheld programmer and our **DirectSOFT32** programming software.

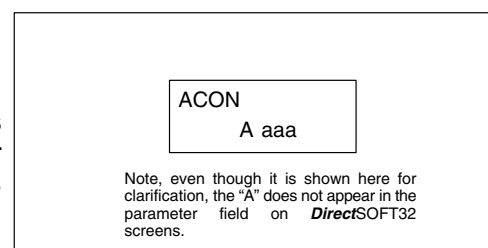
Handheld Programmer: With a handheld programmer, two ASCII characters can be stored in an ACON instruction. If only one character is stored in an ACON, a leading space will be printed in the Fault message.



Operand Data Type	DL430 Range	DL440 Range	DL450 Range
	aaa	aaa	aaa
ASCII A	—	0-9 A-Z	0-9 A-Z

DirectSOFT32 Programming Software:

If you're using **DirectSOFT32**, you can store up to 40 characters in an ACON. Also, you have a much wider range of characters that are supported. (See your **DirectSOFT32** manual for a complete listing of ASCII characters available.)



Operand Data Type	DL430 Range	DL440 Range	DL450 Range
	aaa	aaa	aaa
ASCII A	—	(See DirectSOFT32 Manual)	(See DirectSOFT32 Manual)

At first glance you may wonder why the instructions work differently in the two different programming tools. In reality, the instructions *do not* work differently. In **DirectSOFT32**, the 40-characters are actually broken down into multiple ACONs that contain 2 characters each when it is downloaded to the CPU. So, if you create the program with the software, and you examine the program mnemonics with a handheld (or even with **DirectSOFT32**), you would see multiple ACONs that contain 2 characters each. Examples are shown on the following pages.

Numeric Constant (NCON)

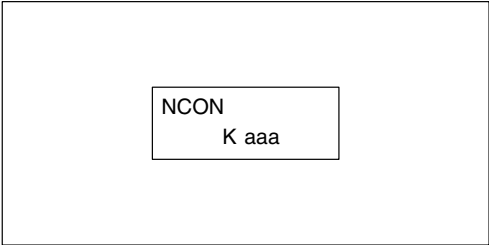
✗

✓

✓

430 440 450

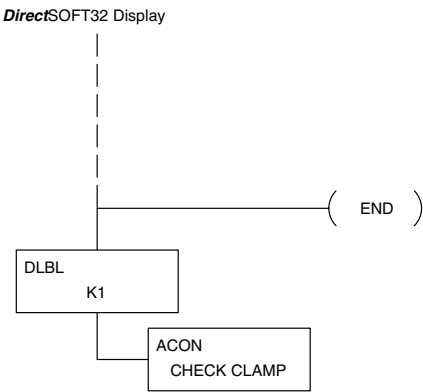
The Numeric Constant instruction is used with the DLBL instruction to store the HEX ASCII equivalent of numeric data for use with other instructions. Two digits can be stored in an NCON instruction.



Operand Data Type		DL440 Range	DL450 Range
		aaa	aaa
Constant	K	0-FFFF	0-FFFF

Using the Instructions to Build the Messages

The Fault instruction actually copies the messages into the appropriate Error table. However, it is important to understand how the DLBL, ACON, and NCON instructions are combined to *build* the message. The DLBL is placed after the END instruction, which signifies the end of the main program. The ACON and NCON instructions are placed within the DLBL area. The diagram shows an example.



History (HISTRY)

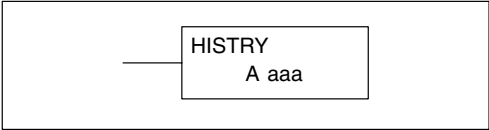
✗

✗

✓

430 440 450

The History instruction stores event history information in memory of the PLC.



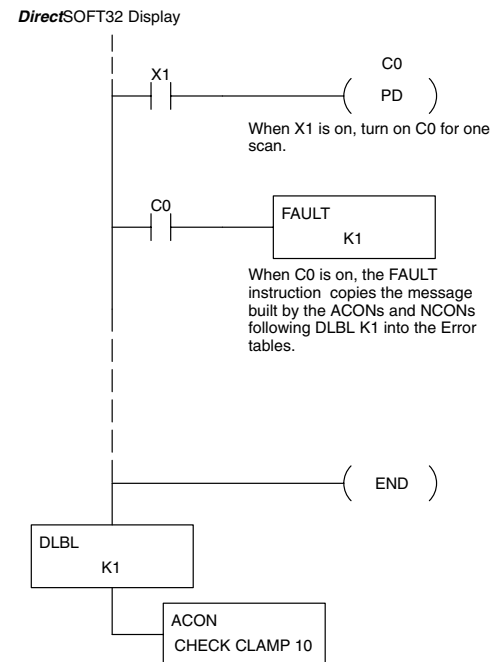
Important Information about FAULT Execution

It is important to understand how the Error Message and Error Code tables work when the Fault instruction is executed. Each time the instruction is executed, the code or message is inserted into the table. Since the CPU scan is extremely fast, then it is easily possible to completely fill up a table with a single message. In many cases this is not desirable, since it's nice to maintain a history of the errors. (That's why there are 32 positions in the Error Code table and 16 positions in the Error Message table.)

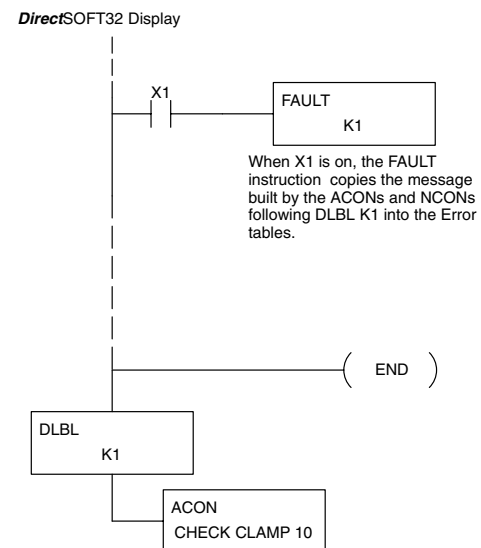
For example, let's say you are examining a limit switch (X1). When the switch is closed, you want an error message (CHECK CLAMP 10) to be logged into the Error Message table. In the real world, the limit switch may be closed for several seconds (or minutes, or hours...). During this time, the CPU will execute the Fault instruction a number of times, so the entire Error Message table will be full of a single message.

How do you solve this problem? Simple. Instead of using the limit switch to trigger the Fault instruction, use the limit switch to trigger a control relay used as a one-shot (PD coil instruction). Then, use the control relay as the input to the Fault instruction. Since the Fault is now triggered by the PD, which is only on for one scan, the message will only be copied into the table one time. This keeps the history of older messages intact.

NOTE: This method *does not* work correctly with a handheld programmer. You *must* use the input contact (X1 in this example) to trigger the Fault instruction directly.



NOTE: This method *does not* work correctly with a handheld programmer. The message will never appear on the display. You *must* use the input contact (X1 in this example) to trigger the Fault instruction directly. The diagram shown here provides an example.

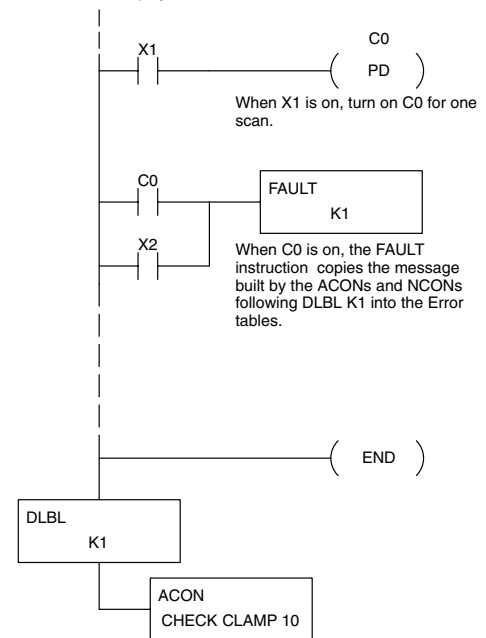


Obviously, you could combine other instructions with the PD instruction usage to develop logic that would work for both the handheld programmer *and* **DirectSOFT32**. For example, if it's possible in your application, you can have *two* input contacts that can trigger the FAULT instruction.

Consider the example shown to the right. If X1 comes on, then the message will only be copied into the table one time. You could see the message with **DirectSOFT32**, but you would not automatically see it on the handheld programmer display.

If X2 comes on, then the message would probably fill the entire table. Also, it would automatically appear on the handheld programmer.

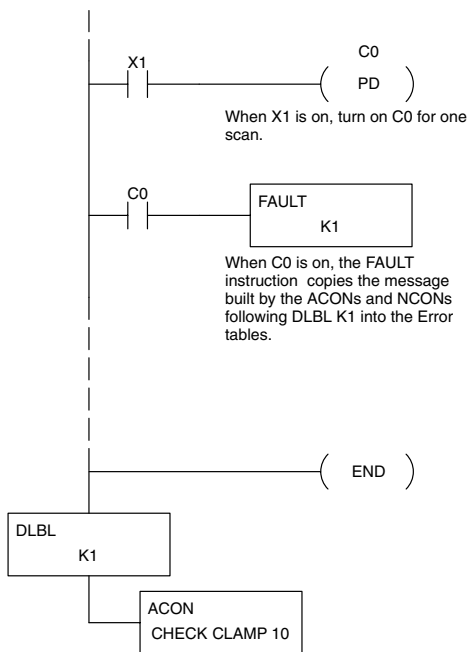
DirectSOFT32 Display



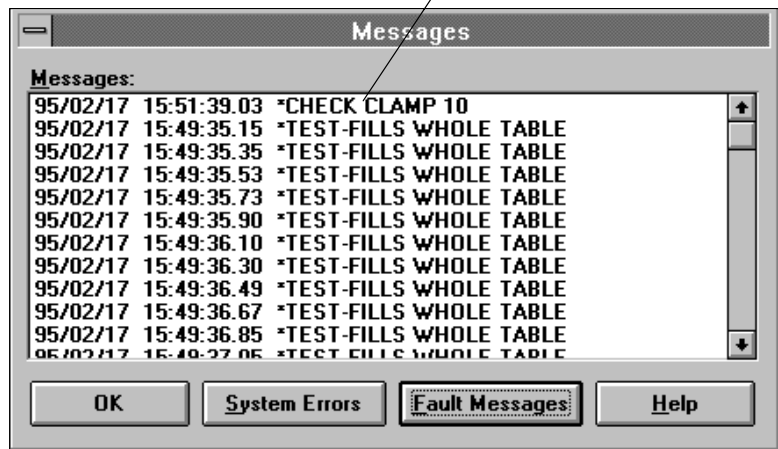
DirectSOFT32 Example

Since you can enter more characters in an ACON instruction in **DirectSOFT32**, it is very easy to build the entire message in one ACON instruction.

DirectSOFT32 Display

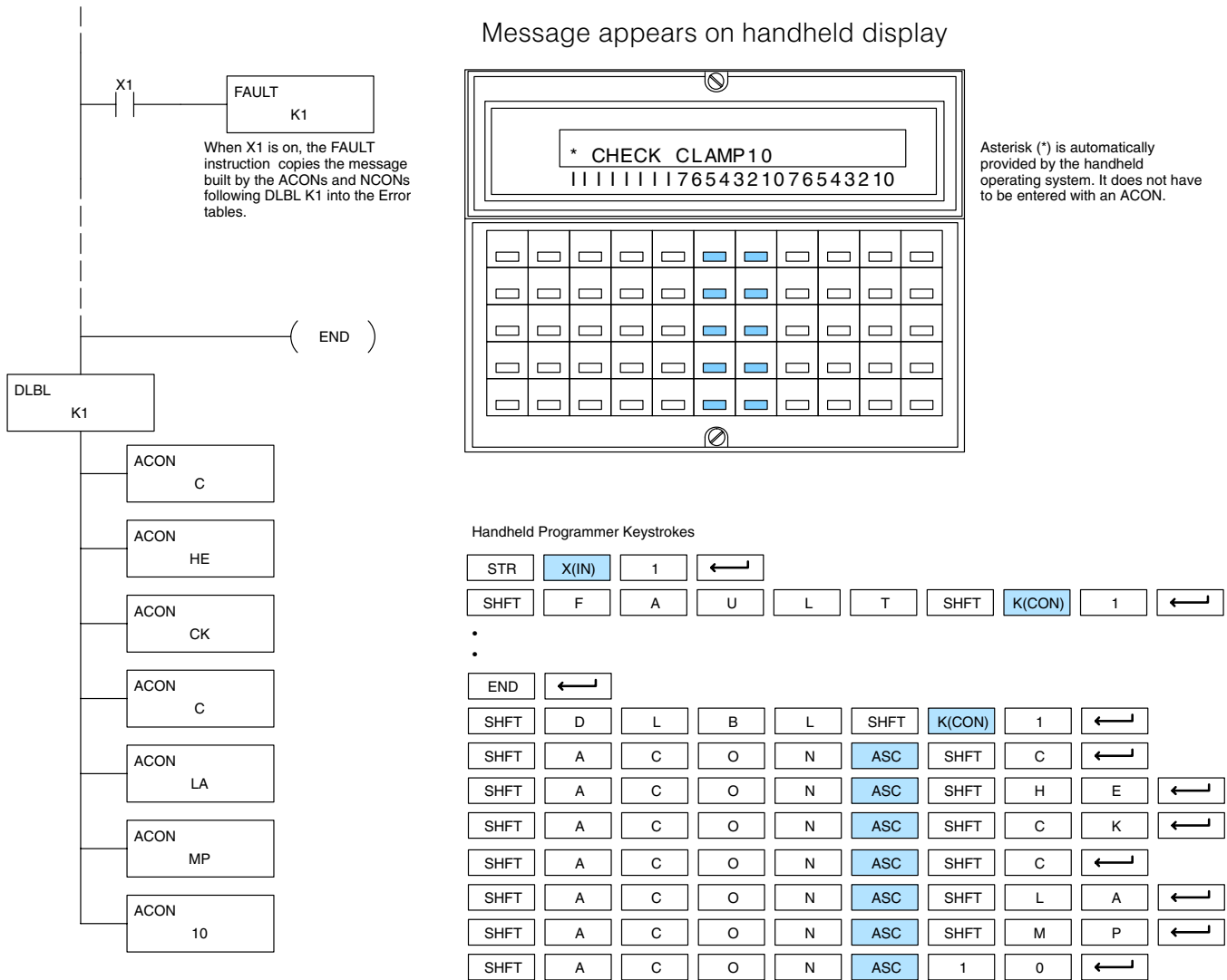


PD C0 triggers one entry into the table



**Handheld
Programmer
Example**

Since you can only enter two characters per ACON with the handheld programmer, the program appears to be longer to get the same results. You'll also notice that we've organized the ACON contents slightly differently. For example, we've only included the character "C" in the first ACON. We have to do it this way in order to get the message to appear correctly. (Remember, a single character entered in an ACON with a handheld will have a blank space preceding it. So if your messages do not contain an even number of characters, you may have to play around with them to get the spacing just right.) Also, we have to use X1 to directly trigger the FAULT instruction. Otherwise, the message will never automatically appear on the handheld programmer display.

**Clearing the
Messages**

You use different methods to clear the System Errors vs. the Fault Messages.

- System Errors — initialize the CPU's scratchpad memory
- Fault Message Table — clear the CPU program memory

WARNING: If you initialize the scratchpad, you will also remove any retentive memory ranges that you may have changed.

**Print Message
(PRINT)**

430 440 450

The Print Message instruction prints the embedded text or text/data variable message to the specified communications port (1, 2, or 3 on the DL450 CPU), which must have the communications port configured.

Data Type	DL450 Range
A	aaa
Constant K	1, 2, or 3

You may recall from the CPU specifications in Chapter 3 that the DL450's ports are capable of several protocols. To configure a port using the Handheld Programmer, use AUX 56 and follow the prompts, making the same choices as indicated below on this page. To configure a port in **DirectSOFT32**, choose the PLC menu, then Setup, then Setup Secondary Comm Port.

- **Port:** From the port number list box at the top, choose "Port 3".
- **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 **DirectSOFT32** to use to store the port setup information. You will need to reserve 9 words in V-memory for this purpose. Select "Always use for printing" if it applies.
- **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.
- **RTS:** Choose the appropriate Request-to-Send option for your printer.



Then click the button indicated to send the Port 3 configuration to the CPU, and click Close. Then see Chapter 3 for port wiring information, in order to connect your printer to the DL450.

Ports 1 and 2 on the DL450 has standard RS232 levels, and should work with most printer serial input connections. Port 3 has RS422 levels, and will not work with most printers. You could use port 3 for long cables, converting the signals to RS232 at the other end to drive the printer.

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

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

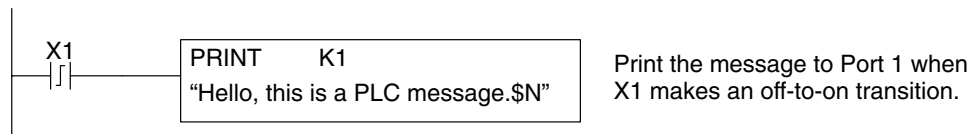
The following examples show various syntax conventions and the length of the output to the printer.

Example:

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

In printing an ordinary line of text, you will need to include double quotation marks before and after the text string. Note that **DirectSOFT32** does not give error indications for syntax errors in the PRINT instruction. Therefore, it is important to test your PRINT instruction data during the application development.

The following example prints the message to port 1. We use a PD contact, which causes the message instruction to be active for just one scan. Note the \$N at the end of the message, which produces a carriage return / line feed on the printer. This prepares the printer to print the next line, starting from the left margin.



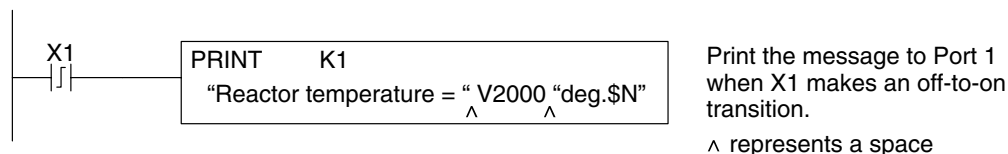
V-memory element – this is used for printing V-memory contents in the integer format or real format. Use V-memory number or V-memory number with “:” and data type. The data types are shown in the table below. The Character code must be in capital letters.

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

Example:

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

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



V-memory text element – this is used for printing text stored in V-memory. Use the % followed by the number of characters after V-memory number for representing the text. If you assign “0” as the number of characters, the print function will read the character count from the first location. Then it will start at the next V-memory location and read that number of ASCII codes for the text from memory.

Example:

V2000 % 16	16 characters in V2000 to V2007 are printed.
V2000 % 0	The characters in V2001 to Vxxxx (determined by the number in V2000) will be printed.

Bit element – this is used for printing the state of the designated bit in V-memory or a relay bit. The bit element can be assigned by the designating point (.) and bit number preceded by the V-memory number or relay number. The output type is described as shown in the table below.

#	Data format	Description
1	none	Print 1 for an ON state, and 0 for an OFF state
2	: BOOL	Print “TRUE” for an ON state, and “FALSE” for an OFF state
3	: ONOFF	Print “ON” for an ON state, and “OFF” for an OFF state

Example:

V2000 . 15	Prints the status of bit 15 in V2000, in 1/0 format
C100	Prints the status of C100 in 1/0 format
C100 : BOOL	Prints the status of C100 in TRUE/FALSE format
C100 : ON:OFF	Prints the status of C00 in ON/OFF format
V2000.15 : BOOL	Prints the status of bit 15 in V2000 in TRUE/FALSE format

The maximum numbers of characters you can print is 128. The number of characters for each element is listed in the table below:

Element type	Maximum Characters
Text, 1 character	1
16 bit binary	6
32 bit binary	11
4 digit BCD	4
8 digit BCD	8
Floating point (real number)	12
Floating point (real with exponent)	12
V-memory/text	2
Bit (1/0 format)	1
Bit (TRUE/FALSE format)	5
Bit (ON/OFF format)	3

The handheld programmer’s mnemonic is “PRINT”, followed by the DEF field.

Special relay flags SP112 through SP117 indicate the status of the DL450 CPU ports (busy, or communications error). See the appendix on special relays for a description.

NOTE: You must use the appropriate special relay in conjunction with the PRINT command to ensure the ladder program does not try to PRINT to a port that is still busy from a previous PRINT or WX or RX instruction.