Standard RLL Instructions

In This Chapter...

Introduction

The DL105 Micro PLCs offer a wide variety of instructions to perform many different types of operations. This chapter shows you how to use each standard Relay Ladder Logic (RLL) instruction. In addition to these instructions, you may also need to refer to the Drum instruction in Chapter 6, or the Stage programming instructions in Chapter 7.

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

- If you know the instruction category (Boolean, Comparative Boolean, etc), use the title at the top of the manual page that refers to the instructions in that category.
- OR if you know the individual instruction name, use the following table to find the page reference for the instruction.

Instruction		Page
ACON	ASCII Constant	5-87
ADD	Add BCD	5-63
ADDD	Add Double BCD	5-64
AND	And for contacts or boxes	5-11, 5-23, 5-55
ANDSTR	And Store	5-12
ANDD	And Double	5-56
ANDE	And if Equal	5-20
ANDI	And Immediate	5-26
ANDN	And Not	5-23
ANDNE	And if Not Equal	5-20
ANDNI	And Not Immediate	5-26
BCD	Binary Coded Decimal	5-75
BIN	Binary	5-74
CMP	Compare	5-61
CMPD	Compare Double	5-62
CNT	Counter	5-35
DECB	Decrement Binary	5-69
DECO	Decode	5-73
DISI	Disable Interrupts	5-83
DIV	Divide	5-68
DLBL	Data Label	5-87
EDRUM	Event Drum	6-2, 6-12
ENCO	Encode	5-72
END	End	5-4, 5-80
ENI	Enable Interrupts	5-83
FAULT	Fault	5-85
INCB	Increment Binary	5-69
INT	Interrupt	5-83
INV	Invert	5-76
IRT	Interrupt Return	5-83
IRTC	Interrupt Return Conditional	5-83
ISG	Initial Stage	7-20
JMP	Jump	7-20

Instruction		Page
LD	Load	5-47
LDA	Load Address	5-50
LDD	Load Double	5-48
LDF	Load Formatted	5-49
LDLBL	Load Label	5-78
MLR	Master Line Reset	5-81
MLS	Master Line Set	5-81
MOV	Move	5-77
MOVMC	Move Memory Cartridge	5-78
MUL	Multiply	5-67
NCON	Numeric Constant	5-87
NOP	No Operation	5-80
OR	Or	5-10, 5-22, 5-57
OROUT	Or Out	5-14
OROUTI	Or Out Immediate	5-27
ORSTR	Or Store	5-12
ORD	Or Double	5-58
ORE	Or if Equal	5-19
ORI	Or Immediate	5-25
ORN	Or Not	5-10, 5-22
ORNE	Or if Not Equal	5-19
ORNI	Or Not Immediate	5-25
OUT	Out	5-14, 5-51
OUTD	Out Double	5-52
OUTF	Out Formatted	5-51
OUTI	Out Immediate	5-27
PAUSE	Pause	5-17
PD	Positive Differential	5-15
POP	Pop	5-54
RST	Reset	5-16
RSTI	Reset Immediate	5-28
SET	Set	5-16
SETI	Set Immediate	5-28
SG	Stage	7-19
SGCNT	Stage Counter	5-37

Continued on next page.

Using Boolean Instructions

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

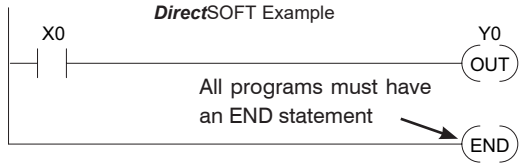
Many of the instructions in this chapter are not program instructions used in *Direct*SOFT, but are implied. In other words, they are not actually keyboard commands but they can be seen in a Mnemonic View of the program once the *Direct*SOFT program has been developed and accepted (compiled). Each instruction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

DS	Implied
HPP	Used

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

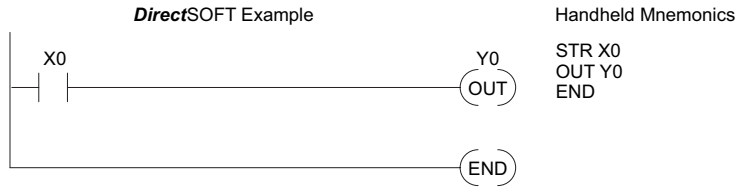
END Statement

All DL105 programs require an END statement as the last instruction. This tells the CPU that this is the end of the program. Normally, any instructions placed after the END statement will not be executed. There are exceptions to this such as interrupt routines, etc. This chapter discusses the instruction set in detail.



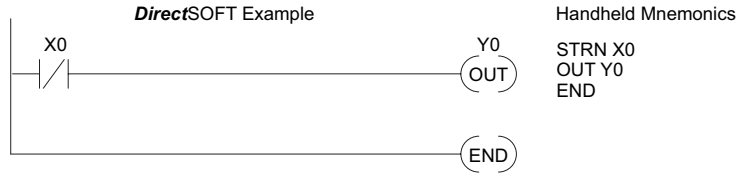
Simple Rungs

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.



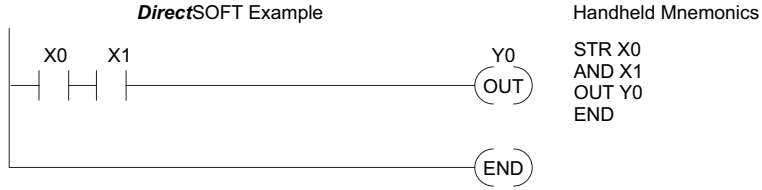
Normally Closed Contact

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.



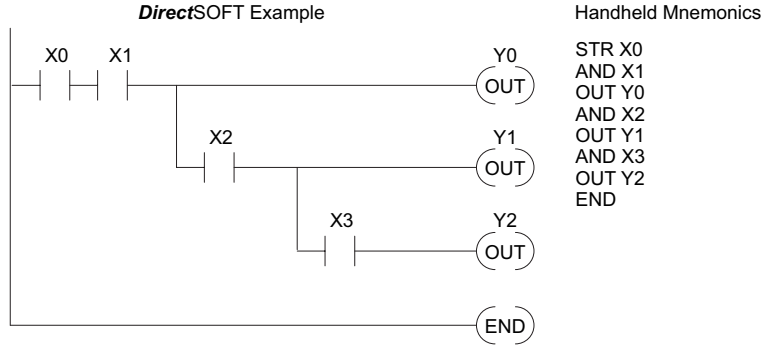
Contacts in Series

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.



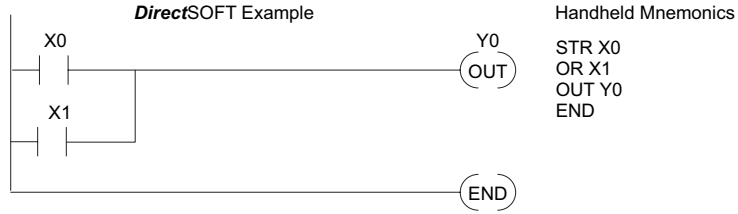
Mid-line Outputs

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



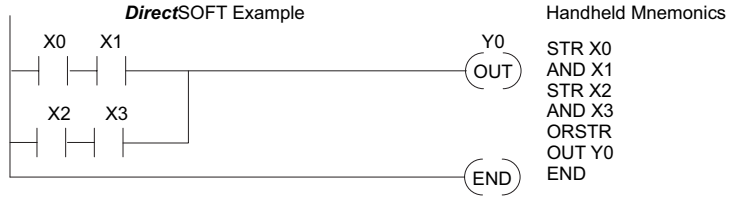
Parallel Elements

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



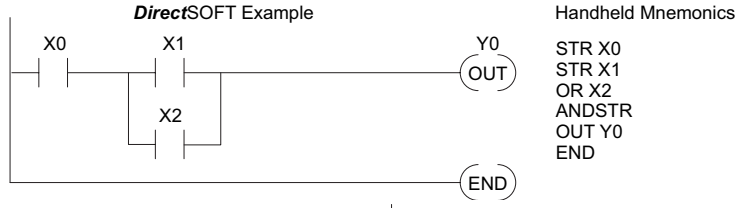
Joining Series Branches in Parallel

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



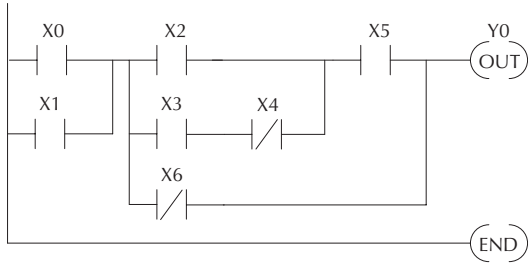
Joining Parallel Branches in Series

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



Combination Networks

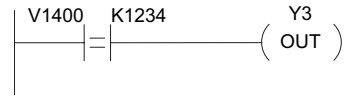
You can combine the various types of series and parallel branches to solve most any application problem. The following example shows a simple combination network.



Comparative Boolean

The DL105 Micro PLCs provide Comparative Boolean instructions that allow you to quickly and easily compare two numbers. The Comparative Boolean provides evaluation of two 4-digit values using boolean contacts. The valid evaluations are: equal to, not equal to, equal to or greater than, and less than.

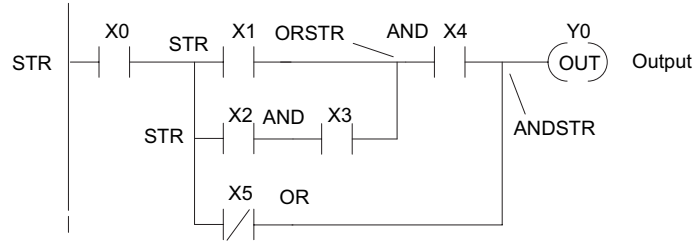
In the example ,when the BCD value in V-memory location V1400 is equal to the constant value 1234, Y3 will energize.



Boolean Stack

There are limits to how many elements you can include in a rung. This is because the DL105 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.

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



STR X0

1	STR X0
2	
3	
4	

STR X1

1	STR X1
2	STR X0
3	
4	

STR X2

1	STR X2
2	STR X1
3	STR X0
4	

AND X3

1	STR X2
2	STR X1
3	STR X0
4	

ORSTR

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

AND X4

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

ORNOT X5

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

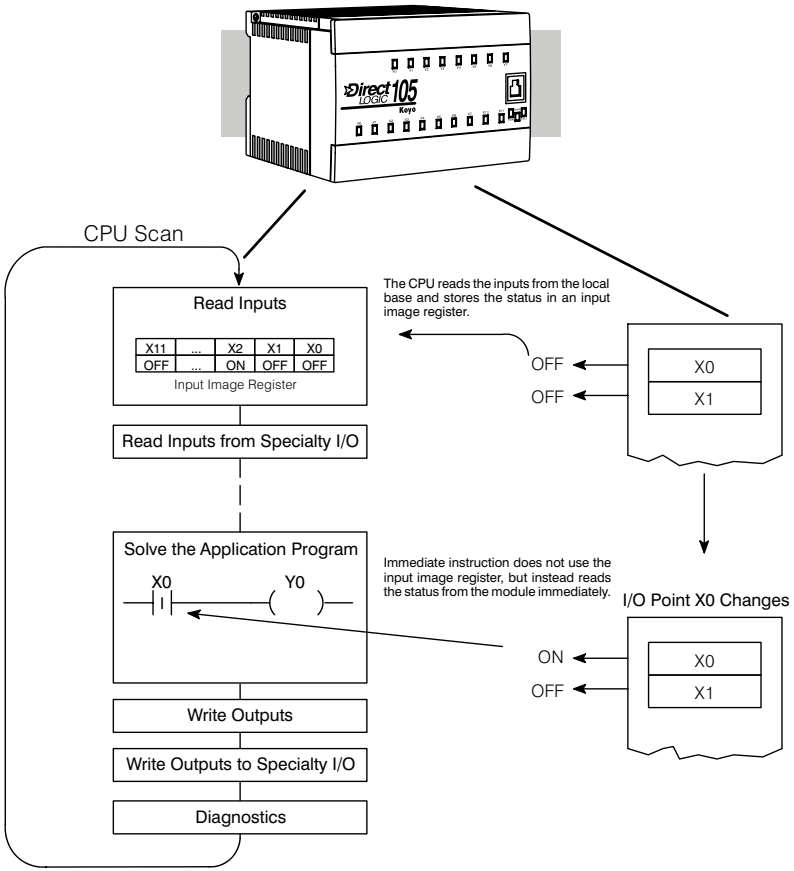
ANDSTR

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

Immediate Boolean

The DL105 Micro PLCs can usually complete an operation cycle in a matter of milliseconds. However, in some applications you may not be able to wait a few milliseconds until the next I/O update occurs. The DL105 PLCs offer immediate input and outputs which are special boolean instructions that allow reading directly from inputs and writing directly to outputs during the program execution portion of the CPU cycle. You may recall that this is normally done during the input or output update portion of the CPU cycle. The immediate instructions take longer to execute because the program execution is interrupted while the CPU reads or writes the I/O point. This function is not normally done until the read inputs or the write outputs portion of the CPU cycle.

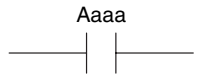
NOTE: *Even though the immediate input instruction reads the most current status from the input point, it only uses the results to solve that one instruction. It does not use the new status to update the image register. Therefore, any regular instructions that follow will still use the image register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



Boolean Instructions

Store (STR)

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		Range
		DL-130
	A	aaa
Inputs	X	0–11
Outputs	Y	0–7
Control Relays	C	0–377
Stage	S	0–377
Timer	T	0–77
Counter	CT	0–77
Special Relay	SP	0–117, 540–577

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

DS	Used
HPP	Used

*Direct*SOFT

Handheld Programmer Keystrokes

$ STR	→	B 1	ENT
GX OUT	→	C 2	ENT

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

DS	Used
HPP	Used

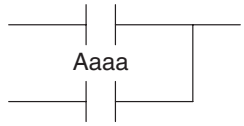
*Direct*SOFT

Handheld Programmer Keystrokes

SP STRN	→	B 1	ENT
GX OUT	→	C 2	ENT

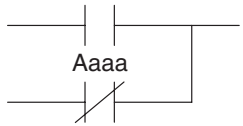
Or (OR)

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



Or Not (ORN)

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



Operand Data Type		Range DL-130
	A	aaa
Inputs	X	0-11
Outputs	Y	0-7
Control Relays	C	0–377
Stage	S	0–377
Timer	T	0–77
Counter	CT	0–77
Special Relay	SP	0-117, 540-577

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

DS	Implied
HPP	Used

*Direct*SOFT

Handheld Programmer Keystrokes

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

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

DS	Implied
HPP	Used

*Direct*SOFT

Handheld Programmer Keystrokes

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

And (AND)

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

And Not (ANDN)

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

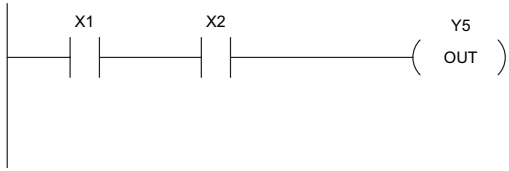
In the following And example, when input X1 and X2

Operand Data Type		DL-130 Range
	A	aaa
Inputs	X	0–11
Outputs	Y	0–7
Control Relays	C	0–377
Stage	S	0–377
Timer	T	0–77
Counter	CT	0–77
Special Relay	SP	0–117, 540–577

DS	Implied
HPP	Used

are on output Y5 will energize.

*Direct*SOFT



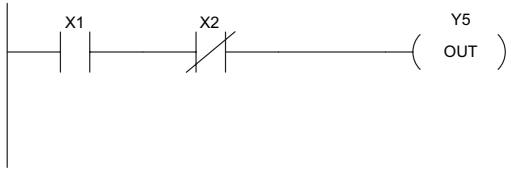
Handheld Programmer Keystrokes

$ STR	→	B 1	ENT
V AND	→	C 2	ENT
GX OUT	→	F 5	ENT

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

DS	Implied
HPP	Used

*Direct*SOFT

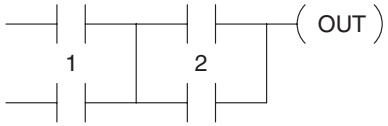


Handheld Programmer Keystrokes

$ STR	→	B 1	ENT
W ANDN	→	C 2	ENT
GX OUT	→	F 5	ENT

And Store (ANDSTR)

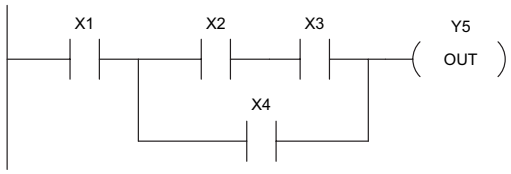
The And Store instruction logically ands two branches of a rung in series. Both branches must begin with the Store instruction.



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.

DS	Implied
HPP	Used

DirectSOFT

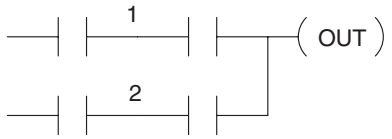


Handheld Programmer Keystrokes

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

Or Store (ORSTR)

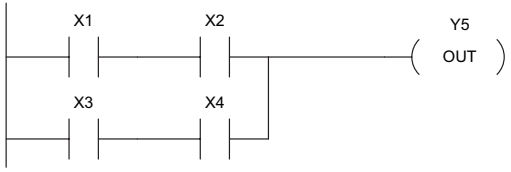
The Or Store instruction logically ors two branches of a rung in parallel. Both branches must begin with the Store instruction.



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

DS	Implied
HPP	Used

DirectSOFT

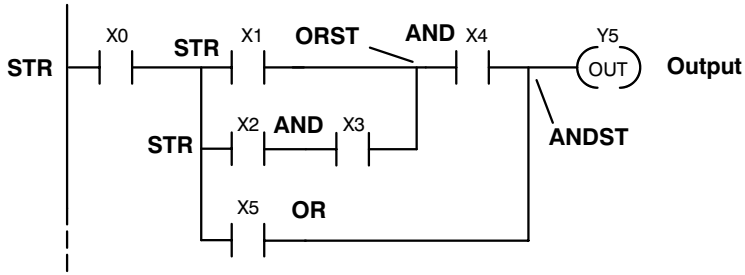


Handheld Programmer Keystrokes

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

There are limits to what you can enter with boolean instructions. This is because the DL105 internal CPU uses an 8-level stack to evaluate the various logic elements. The stack is a temporary storage area that helps solve the logic for the rung. Each time you enter a STR instruction, the instruction is placed on the top of the stack. Any other instructions on the stack are pushed down a level. The And Store and Or Store instructions combine levels of the stack when they are encountered. Since the stack is only eight levels, an error will occur if the CPU encounters a rung that uses more than the eight levels of the stack.

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



STR X0

1	STR X0
2	
3	
4	
5	
6	
7	
8	

STR X1

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

STR X2

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

AND X3

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

ORST

1	X1 OR (X2 AND X3)
2	STR X0
3	
⋮	
8	

AND X4

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

OR X5

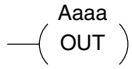
1	X5 OR [X4 AND [X1 OR (X2 AND X3)]]
2	STR X0
3	
⋮	
8	

ANDST

1	X0 AND [(X5 OR [X4) AND [X1 OR (X2 AND X3)]]]
2	
3	
⋮	
8	

Out (OUT)

The Out instruction reflects the status of the rung (on/off) and outputs the discrete (on/off) state to the specified image register point or memory location. Multiple Out instructions referencing the same discrete location should not be used since only the last Out instruction in the program will control the physical output point. Instead, use the next instruction, the Or Out.



Operand Data Type		Range DL-130
	A	aaa
Inputs	X	0–11
Outputs	Y	0–7
Control Relays	C	0–377

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

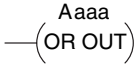
DS	Used
HPP	Used

Handheld Programmer Keystrokes

$ STR	→	B 1	ENT
GX OUT	→	C 2	ENT
GX OUT	→	F 5	ENT

Or Out (OROUT)

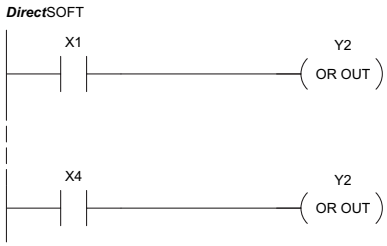
The Or Out instruction allows more than one rung of discrete logic to control a single output. Multiple Or Out instructions referencing the same output coil may be used, since *all* contacts controlling the output are logically OR'd together. If the status of *any* rung is on, the output will also be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



Operand Data Type		Range DL-130
	A	aaa
Inputs	X	0–177
Outputs	Y	0-177
Control Relays	C	0–377

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

DS	Used
HPP	Used

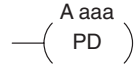


Handheld Programmer Keystrokes

$ STR	→	B 1	ENT				
O INST#	D 3	F 5	ENT	ENT	→	C 2	ENT
$ STR	→	E 4	ENT				
O INST#	D 3	F 5	ENT	ENT	→	C 2	ENT

Positive Differential (PD)

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



Operand Data Type		Range
		DL-130
	A	aaa
Inputs	X	0–11
Outputs	Y	0–7
Control Relays	C	0–377

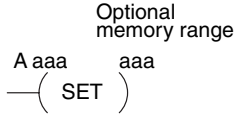
In the following example, every time X1 makes an off to on transition, C0 will energize for one scan.

DS	Used
HPP	Used

Handheld Programmer Keystrokes

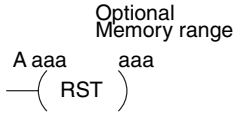
Set (SET)

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.



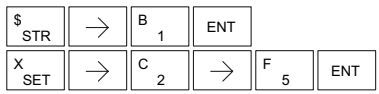
Operand Data Type		Range DL-130
	A	aaa
Inputs	X	0–11
Outputs	Y	0-7
Control Relays	C	0–377
Stage	S	0-377
Timer	T	0-77
Counter	CT	0-77

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

DS	Used
HPP	Used

*Direct*SOFT

Handheld Programmer Keystrokes

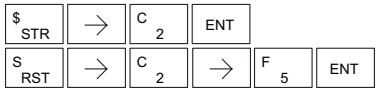


In the following example, when X2 is on, Y2 through Y5 will be reset or de–energized.

DS	Used
HPP	Used

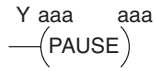
*Direct*SOFT

Handheld Programmer Keystrokes



Pause (PAUSE)

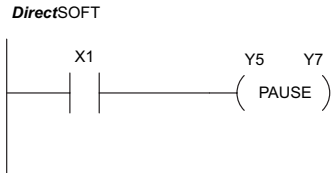
The Pause instruction disables the output update on a range of outputs. The ladder program will continue to run and update the image register; however, the outputs in the range specified in the Pause instruction will be turned off at the output points.



Operand Data Type		Range
		DL-130
		aaa
Outputs	Y	0-7

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

DS	Used
HPP	Used



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

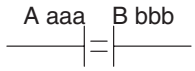
Handheld Programmer Keystrokes

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

Comparative Boolean

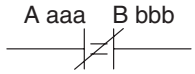
Store If Equal (STRE)

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



Store If Not Equal (STRNE)

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



Operand Data Type		Range DL-130	
	B	aaa	bbb
V-memory	V	All. (See page 4-29)	All. (See page 4-29)
Constant	K	–	0-FFFF

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

Handheld Programmer Keystrokes

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

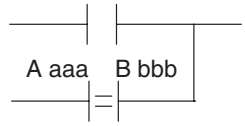
In the following example, when the value in V-memory location V2000 ≠ 5060, Y3 will energize.

Handheld Programmer Keystrokes

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

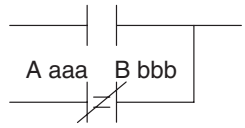
Or If Equal (ORE)

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



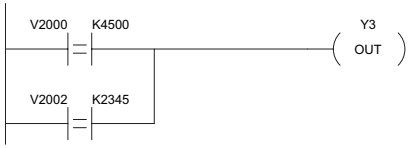
Or If Not Equal (ORNE)

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



Operand Data Type		Range	
		DL-130	
	B	aaa	bbb
V-memory	V	All. (See page 4-29)	All. (See page 4-29)
Constant	K	–	0-FFFF

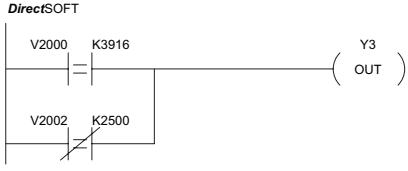
In the following example, when the value in V-memory location V2000 = 4500 or V2202 = 2345, Y3 will energize.



Handheld Programmer Keystrokes

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

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

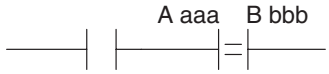


Handheld Programmer Keystrokes

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

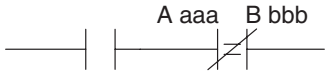
And If Equal (ANDE)

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



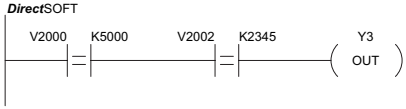
And If Not Equal (ANDNE)

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



Operand Data Type		Range DL-130	
	A/B	aaa	bbb
V-memory	V	All (See page 4-29)	All (See page 4-29)
Constant	K	–	0-FFFF

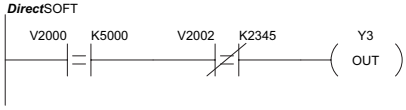
In the following example, when the value in V-memory location V2000 = 5000 and V2002 = 2345, Y3 will energize.



Handheld Programmer Keystrokes

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

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

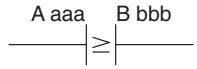


Handheld Programmer Keystrokes

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

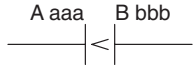
Store (STR)

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



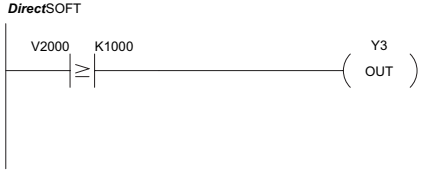
Store Not (STRN)

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



Operand Data Type		Range DL-130	
	A/B	aaa	bbb
V-memory	V	All (See page 4-29)	All (See page 4-29)
Constant	K	–	0-FFFF
Timer	T	0–77	
Counter	CT	0–77	

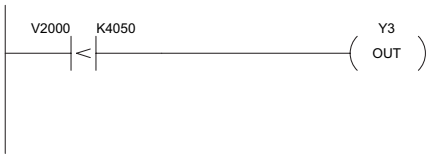
In the following example, when the value in V-memory location V2000 ≥ 1000, Y3 will energize.



Handheld Programmer Keystrokes

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

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

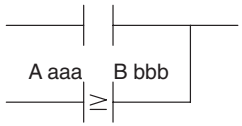


Handheld Programmer Keystrokes

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

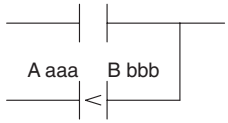
Or (OR)

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



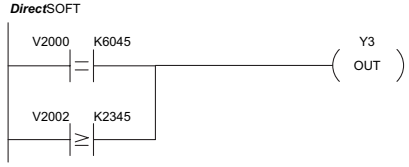
Or Not (ORN)

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



Operand Data Type		Range DL-130	
	A/B	aaa	bbb
V-memory	V	All (See page 4-29)	All (See page 4-29)
Constant	K	–	0-FFFF
Timer	T	0–77	
Counter	CT	0–77	

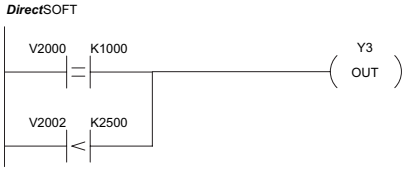
In the following example, when the value in V-memory location V2000 = 6045 or V2002 ≥ 2345, Y3 will energize.



Handheld Programmer Keystrokes

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

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

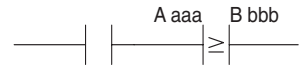


Handheld Programmer Keystrokes

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

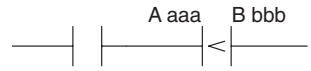
And (AND)

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



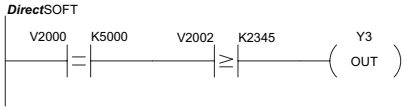
And Not (ANDN)

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		Range	
		DL-130	
	A/B	aaa	bbb
V-memory	V	All (See page 4-29)	All (See page 4-29)
Constant	K	–	0-FFFF
Timer	T	0–77	
Counter	CT	0–77	

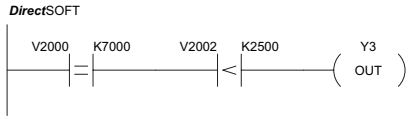
In the following example, when the value in V-memory location V2000 = 5000, and V2002 ≥ 2345, Y3 will energize.



Handheld Programmer Keystrokes

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

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



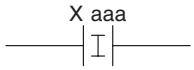
Handheld Programmer Keystrokes

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

Immediate Instructions

Store Immediate (STRI)

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



Store Not Immediate (STRNI)

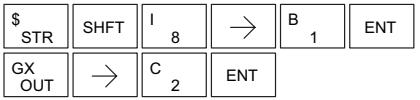
The Store Not Immediate instruction begins a new rung or additional branch in a rung. The status of the contact will be opposite the status of the associated input point *at the time the instruction is executed*. The image register is not updated.

Operand Data Type		Range
		DL-130
		aaa
Inputs	X	0–11

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

DirectSOFT

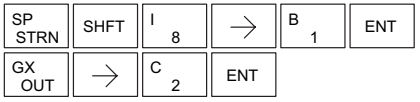
Handheld Programmer Keystrokes



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

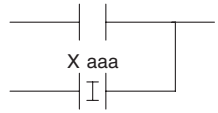
DirectSOFT

Handheld Programmer Keystrokes



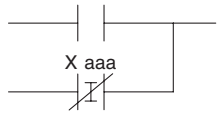
Or Immediate (ORI)

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



Or Not Immediate (ORNI)

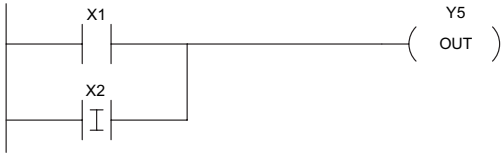
The Or Not Immediate connects two contacts in parallel. The status of the contact will be opposite the status of the associated input point *at the time the instruction is executed*. The image register is not updated.



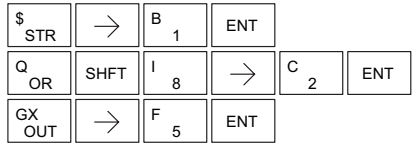
Operand Data Type		Range
		DL-130
		aaa
Inputs	X	0–177

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

DirectSOFT

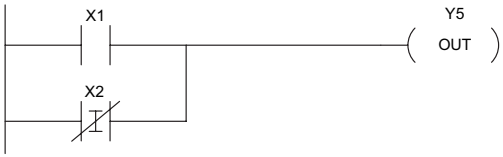


Handheld Programmer Keystrokes

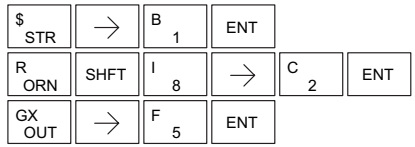


In the following example, when X1 is on or X2 is off, Y5 will energize.

DirectSOFT

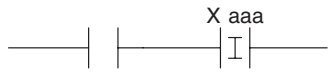


Handheld Programmer Keystrokes



And Immediate (ANDI)

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



And Not Immediate (ANDNI)

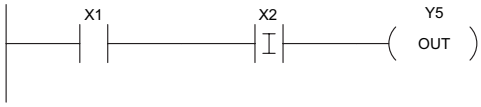
The And Not Immediate connects two contacts in series.
opposite the status of the associated input point *at the tim*
the instruction is executed. The image register is not updatec

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

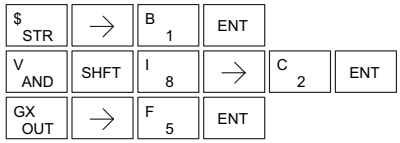
Operand Data Type	Range
	DL-130
	aaa
Inputs X	0–11

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

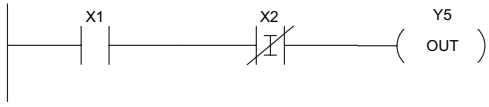
DirectSOFT



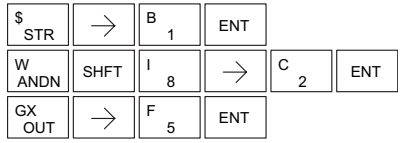
Handheld Programmer Keystrokes



DirectSOFT

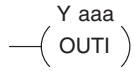


Handheld Programmer Keystrokes



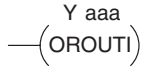
Out Immediate (OUTI)

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



Or Out Immediate (OROUTI)

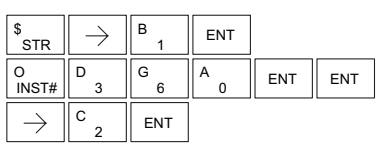
The Or Out Immediate instruction has been designed to use more than one rung of discrete logic to control a single output. Multiple Or Out Immediate instructions referencing the same output coil may be used, since all contacts controlling the outp
status of any rung is on *at the time the instruction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

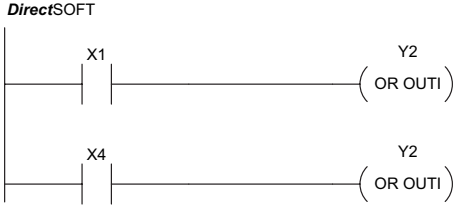


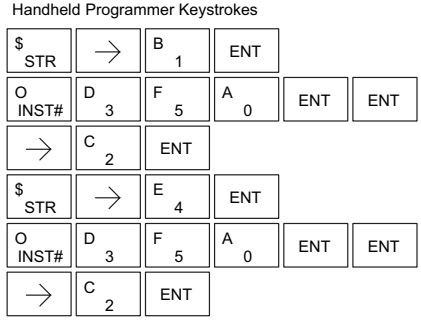
In the following example, when X1 is on, output point Y2 on the c
will turn on. For instruction entry on the Handheld Prograı
the instruction number (#350) as shown, or type each letter

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

Operand Data Type		Range DL-130
		aaa
Outputs	Y	0-177

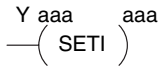






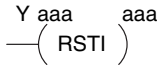
Set Immediate (SETI)

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



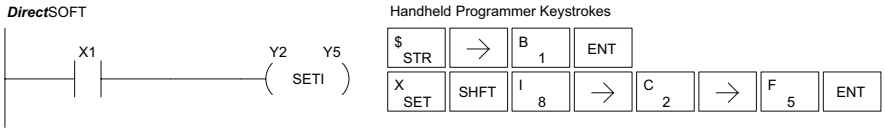
Reset Immediate (RSTI)

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

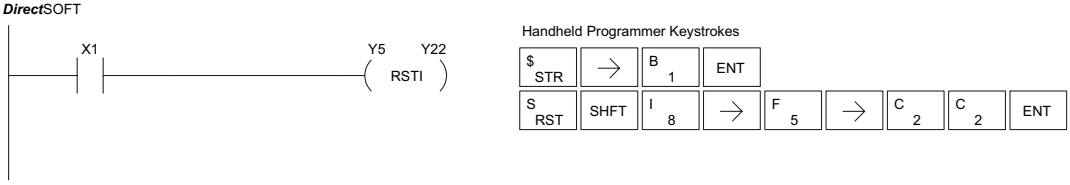


Operand Data Type		Range DL-130
		aaa
Outputs	Y	0-177

In the following example, when X1 is on, Y2 through Y5 will be set on in the image register and on the corresponding output points.



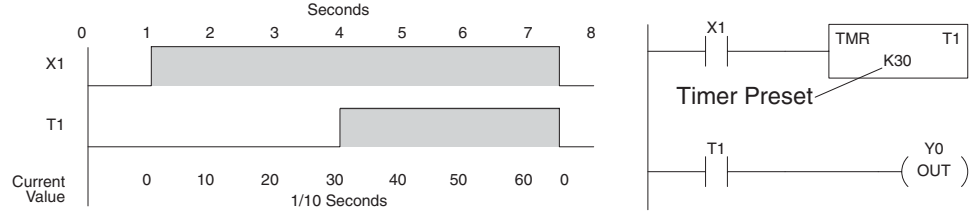
In the following example, when X1 is on, Y5 through Y22 will be reset (off) in the image register and on the corresponding output module(s).



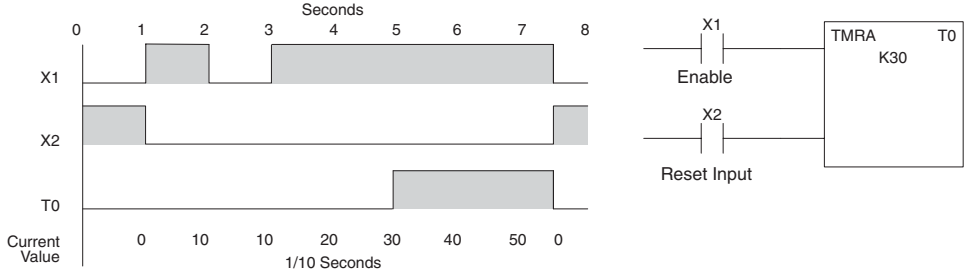
Timer, Counter and Shift Register Instructions

Using Timers

Timers are used to time an event for a desired length of time. 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. A discrete bit is associated with each timer to indicate that the current value is equal to or greater than the preset value. The timing diagram below shows the relationship between the timer input, associated discrete bit, current value, and timer preset.

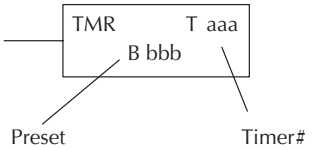


Some applications that need an accumulating timer, meaning it has the ability to time, stop, and then resume from where it previously stopped. The accumulating timer works similarly to the regular timer, but two inputs are required. The enable input starts and stops the timer. When the timer stops, the elapsed time is maintained. When the timer starts again, the timing continues from the elapsed time. When the reset input is turned on, the elapsed time is cleared and the timer will start at 0 when it is restarted. A tenth of a second and a hundredth of a second timers are available with a maximum time of 9999999.9 and 999999.99 seconds respectively. The timing diagram below shows the relationship between the timer input, timer reset, associated discrete bit, current value, and timer preset.



Timer (TMR) and Timer Fast (TMRF)

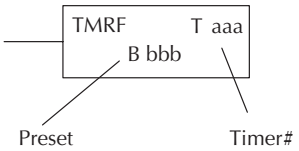
The Timer instruction is a 0.1 second single-input timer that times to a maximum of 999.9 seconds. The Timer Fast instruction is a 0.01 second single input timer that times up to a maximum of 99.99 seconds. These timers will be enabled if the input logic is true (on) and will be reset to 0 if the input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



Instruction Specifications

Timer Reference (Taaa): Specifies the timer number.

Preset Value (Bbbb): Constant value (K) or a V-memory location. (Pointer (P) for D2-240, D2-250–1, D2-260 and D2-262).



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

Discrete Status Bit: The discrete status bit is referenced by 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.

NOTE: *Timer preset constants (K(may be changed by using a handheld programmer, even when the CPU is in Run mode. A V-memory preset is required only if the ladder program or an Operator Interface unit must change the preset.*

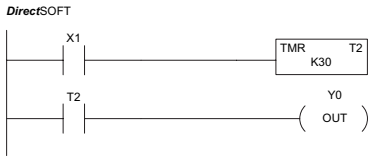
Operand Data Type		DL-130 Range	
	A/B	aaa	bbb
Timers	T	0–77	
V-memory for preset values	V	–	2000–2377 4000–4177
Pointers (presets only)	P		
Constants (presets only)	K	–	0–9999
Timer discrete status bits	T/V	0–77 or V41100–41103	
Timer current values	V/T*	0-77	

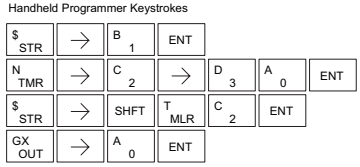
NOTE: **Both the Timer discrete status bits and the current value are accessed with the same data reference with the HPP.* ***Direct****SOFT uses separate references, such as "T2" for discrete status bit for Timer T2, and "TA2" for the current value of Timer T2.*

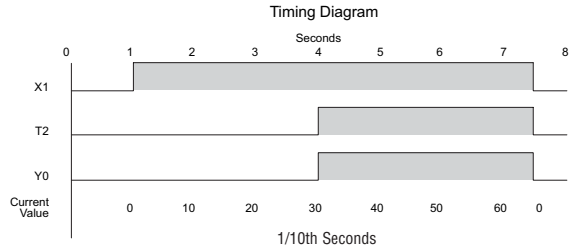
You can perform functions when the timer reaches the specified preset using the discrete status bit. Or, use the comparative contacts to perform functions at different time intervals based on one timer. The examples on the following page show these methods of programming timers.

Timer Example Using Discrete Status Bits

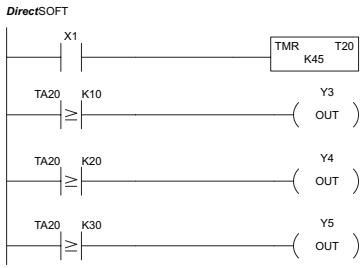
In the following example, a single-input timer is used with a preset of 3 seconds. The timer discrete status bit (T2) will turn on when the timer has timed for 3 seconds. The timer is reset when X1 turns off, turning the discrete status bit off and resetting the timer current value to 0.

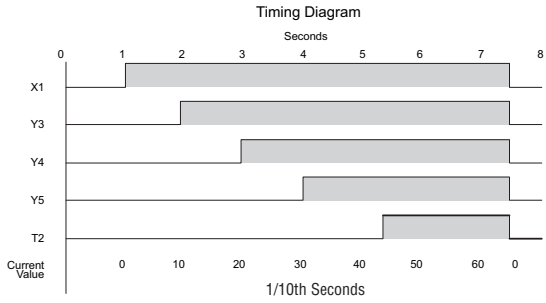


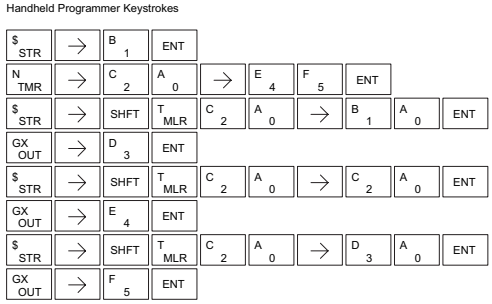




In the following example, a single-input timer is used with a preset of 4.5 seconds. Comparative contacts are used to energize Y3, Y4, and Y5 at one-second intervals respectively. When X1 is turned off, the timer will be reset to 0 and the comparative contacts will turn off Y3, Y4, and Y5.

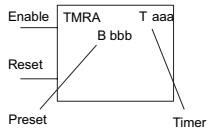






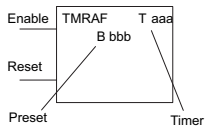
Accumulating Timer (TMRA)

The Accumulating Timer is a 0.1 second, two-input timer that will time to a maximum of 9999999.9. *The TMRA uses two timer registers in V-memory.*



Accumulating Fast Timer (TMRAF)

The Accumulating Fast Timer is a 0.01 second two-input timer that will time to a maximum of 999999.99. *The TMRAF uses two timer registers in V-memory.*



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 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 two consecutive V-memory locations. (Pointer (P) for D2-240, D2-250–1, D2-260 and D2-262).

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

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

NOTE: *The accumulating timer uses two consecutive V-memory locations for the 8-digit value; therefore, two consecutive timer locations. For example, if TMR 1 is used, the next available timer number is TMR 3.*

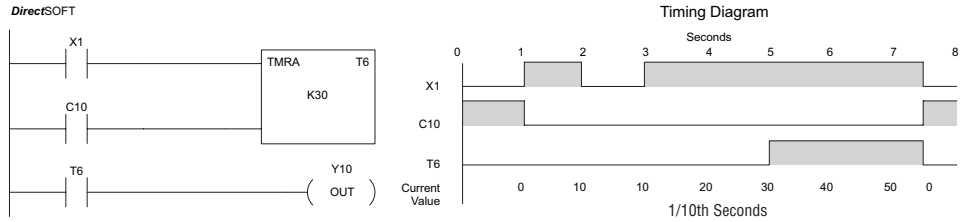
Operand Data Type		DL-130 Range	
	A/B	aaa	bbb
Timers	T	0–77	
V-memory for preset values	V	–	2000–2376 4000–4176
Pointers (presets only)	P		
Constants (presets only)	K	–	0–99999999
Timer discrete status bits	T/V	0–77 or V41100–41103	
Timer current values	V/T*	0-77	

NOTE: ** Both the Timer discrete status bits and the current value are accessed with the same data reference with the HPP.* ***Direct****SOFT uses separate references, such as "T2" for discrete status bit for Timer T2, and "TA2" for the current value of Timer T2.*

The following examples show two methods of programming timers. One performs functions when the timer reaches the preset value using the discrete status bit, or use comparative contacts to perform functions at different time intervals.

Accumulating Timer Example using Discrete Status Bits

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



Handheld Programmer Keystrokes

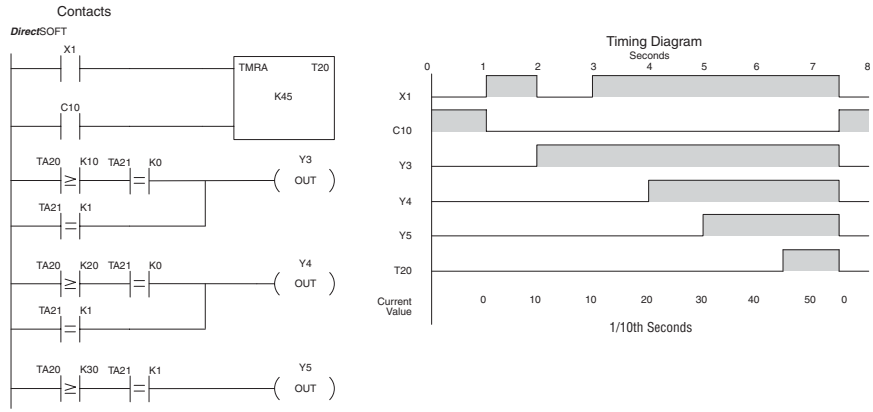
$ STR	→	B 1	ENT			
$ STR	→	SHFT	C 2	B 1	A 0	ENT
N TMR	SHFT	A 0	→	G 6	→	

Handheld Programmer Keystrokes (cont'd)

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

Accumulator Timer Example Using Comparative Contacts

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



Handheld Programmer Keystrokes

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

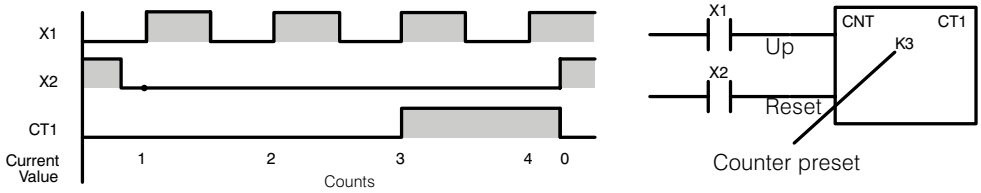
Handheld Programmer Keystrokes (cont'd)

$ STR	→	SHFT	T MLR	C 2	A 0	→	C 2	A 0	ENT	
V AND	SHFT	E 4	→	SHFT	T MLR	C 2	B 1	→	A 0	ENT
Q OR	SHFT	E 4	→	SHFT	T MLR	C 2	B 1	→	B 1	ENT
GX OUT	→	E 4	ENT							
$ STR	→	SHFT	T MLR	C 2	A 0	→	D 3	A 0	ENT	
V AND	SHFT	E 4	→	SHFT	T MLR	C 2	B 1	→	B 1	ENT
GX OUT	→	F 5	ENT							

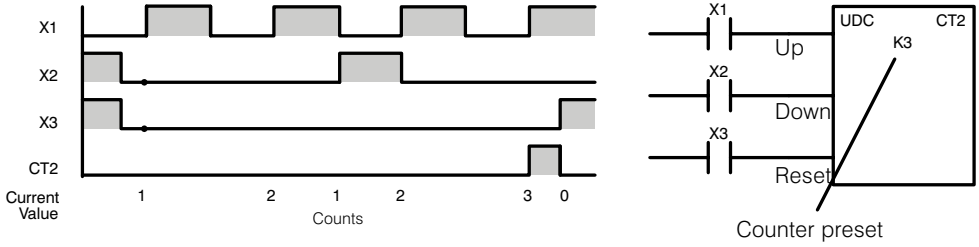
Using Counters

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

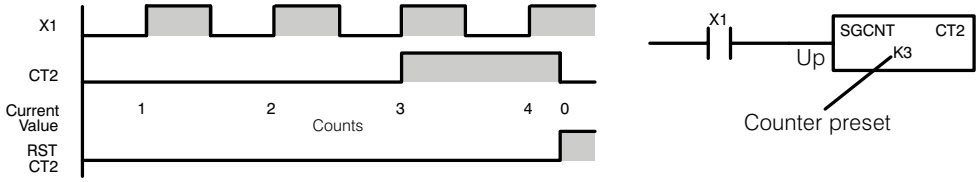
The up counter has two inputs, a count input and a reset input. The maximum count value is 9999. The timing diagram below shows the relationship between the counter input, counter reset, associated discrete bit, current value, and counter preset.



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.

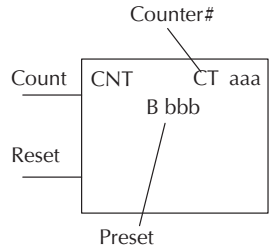


The stage counter has a count input and is reset by the RST instruction. This instruction is useful when programming using the RLL*PLUS* structured programming. The maximum count value is 9999. The timing diagram below shows the relationship between the counter input, associated discrete bit, current value, counter preset and reset instruction.



Counter (CNT)

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 zero. 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. (Pointer (P) for D2-240, D2-250–1, D2-260 and D2-262.)

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

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

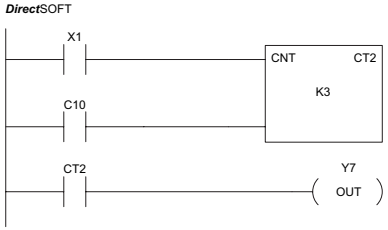
NOTE: *Counter preset constants (K) may be changed by using a programming device, even when the CPU is in Run Mode. A V-memory preset is required only if the ladder program or an OIT must used to change the preset*

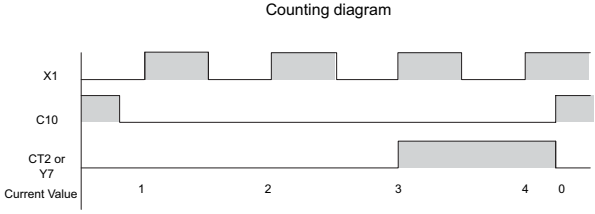
Operand Data Type		DL-130 Range	
	B	aaa	bbb
Counters	CT	0–77	
V-memory for preset values	V	–	2000–2377
Pointers (presets only)	P		
Constants (presets only)	K	–	0–9999
Counter discrete status bits	CT/V	0–77 or V41140–41143	
Counter current values	V/CT*	1000–1077	

NOTE: ** Both the Counter discrete status bits and the current value are accessed with the same data reference with the HPP.* ***Direct****SOFT uses separate references, such as "CT2" for discrete status bit for Counter CT2, and "CTA2" for the current value of Counter CT2.*

Counter Example Using Discrete Status Bits

In the following example, when X1 makes an off-to-on transition, counter CT2 will increment by one. When the current value reaches the preset value of 3, the counter status bit CT2 will turn on and energize Y7. When the reset C10 turns on, the counter status bit will turn off and the current value will be 0. The current value for counter CT2 will be held in V-memory location V1002.





Handheld Programmer Keystrokes

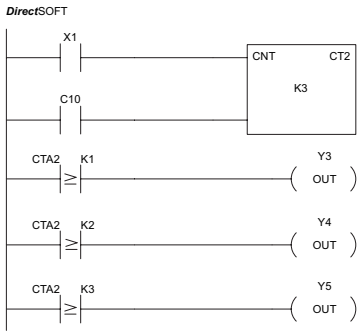
$ STR	→	B 1	ENT				
$ STR	→	SHFT	C 2	B 1	A 0	ENT	
GY CNT	→	C 2	→	D 3	ENT		

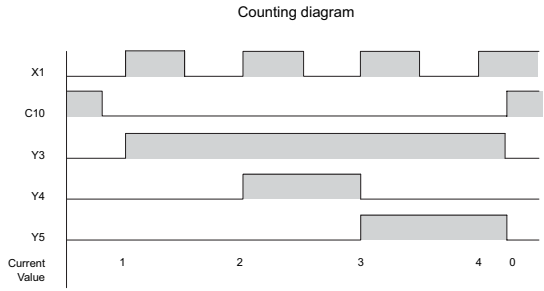
Handheld Programmer Keystrokes (cont)

$ STR	→	SHFT	C 2	SHFT	T MLR	C 2	ENT
GX OUT	→	H 7	ENT				

Counter Example Using Comparative Contacts

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





Handheld Programmer Keystrokes

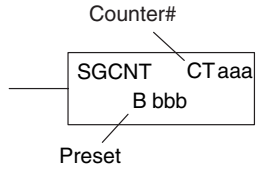
$ STR	→	B 1	ENT				
$ STR	→	SHFT	C 2	B 1	A 0	ENT	
GY CNT	→	C 2	→	D 3	ENT		
$ STR	→	SHFT	C 2	SHFT	T MLR	C 2	
→	B 1	ENT					
GX OUT	→	D 3	ENT				

Handheld Programmer Keystrokes (cont)

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

Stage Counter (SGCNT)

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



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

Instruction Specifications

Counter Reference (CTaaa): Specifies the counter number.

Preset Value (Bbbb): Constant value (K) or a V-memory location.(Pointer (P) for D2-240, D2-250–1, D2-260 and D2-262.)

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

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

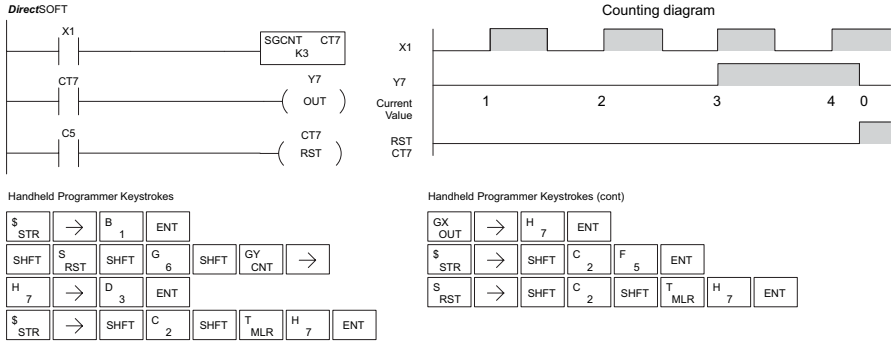
Operand Data Type		DL-130 Range	
	B	aaa	bbb
Counters	CT	0–77	
V-memory for preset values	V	–	2000–2377
Pointers (presets only)	P		
Constants (presets only)	K	–	0–9999
Counter discrete status bits	CT/V	0–77 or V41140–41143	
Counter current values	V/CT*	1000–1077	

***NOTE:** * Both the Counter discrete status bits and the current value are accessed with the same data reference with the HPP. **Direct**SOFT uses separate references, such as "CT2" for discrete status bit for Counter CT2, and "CTA2" for the current value of Counter CT2.*

Stage Counter Example Using Discrete Status Bits

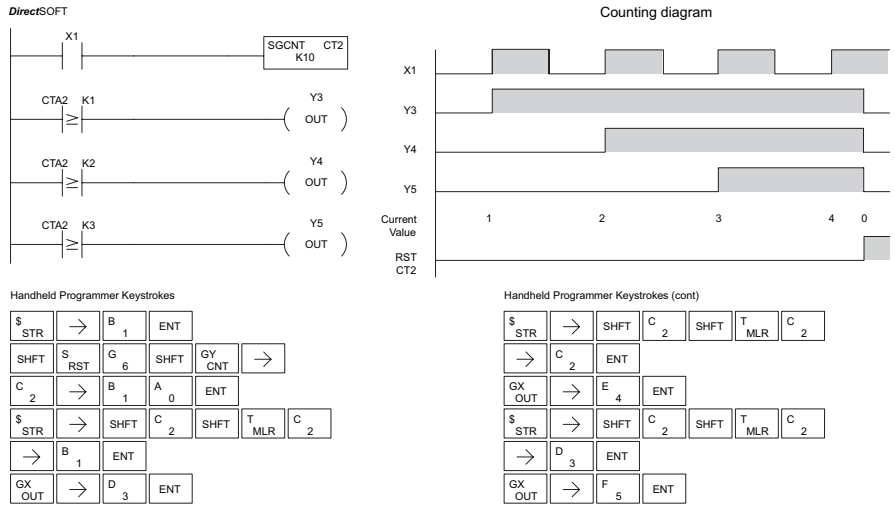
In the following example, when X1 makes an off-to-on transition, stage counter CT7 will increment by one. When the current value reaches 3, the counter status bit CT7 will turn on and energize Y7. The counter status bit CT7 will remain on until the counter is reset using the RST instruction. When the counter is reset, the counter status bit will turn off and the counter current value will be 0. The current value for counter CT7 will be held in V-memory location V1007.

St



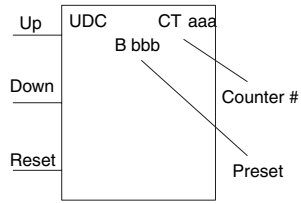
Counter Example Using Comparative Contacts

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



Up Down Counter (UDC)

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



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

Instruction Specification

Counter Reference (CTaaa): Specifies the counter number.

Preset Value (Bbbb): Constant value (K) or two consecutive V-memory locations. (Pointer (P) for D2-240, D2-250–1, D2-260 and D2-262).

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

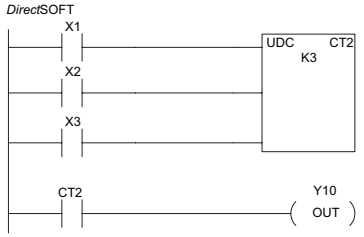
Discrete Status Bit: The discrete status bit is accessed by referencing the associated CT memory location. It will be on if the value is equal to or greater than the preset value. For example, the discrete status bit for Counter 2 would be CT2.

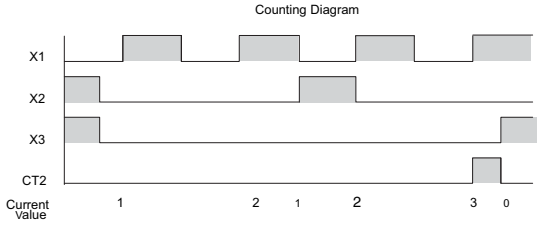
Operand Data Type		DL-130 Range	
	B	aaa	bbb
Counters	CT	0–77	
V-memory for preset values	V	–	2000–2377
Pointers (presets only)	P		
Constants (presets only)	K	–	0–9999
Counter discrete status bits	CT/V	0–77 or V41140–41143	
Counter current values	V/CT*	1000–1077	

*NOTE: * Both the Counter discrete status bits and the current value are accessed with the same data reference with the HPP. **Direct**SOFT uses separate references, such as "CT2" for discrete status bit for Counter CT2, and "CTA2" for the current value of Counter CT2.*

Up/Down Counter Example Using Discrete Status Bits

In the following example, if X2 and X3 are off, 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.





Handheld Programmer Keystrokes

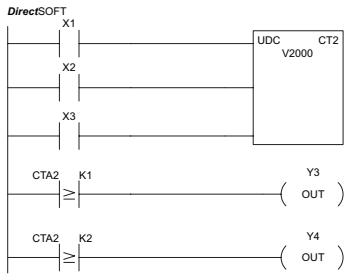
$ STR	→	B 1	ENT			
$ STR	→	C 2	ENT			
$ STR	→	D 3	ENT			
SHFT	U ISG	D 3	C 2	→	C 2	

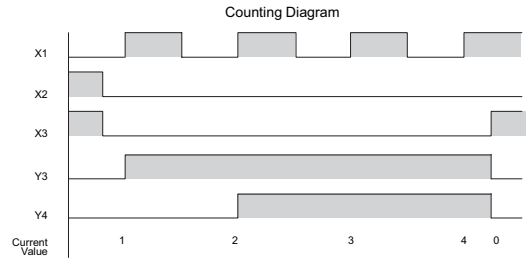
Handheld Programmer Keystrokes (cont)

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

Up/Down Counter Example Using Comparative Contacts

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





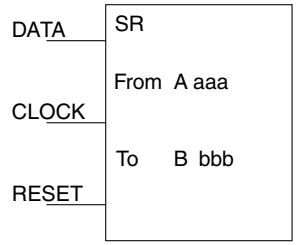
Handheld Programmer Keystrokes

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

Handheld Programmer Keystrokes (cont)

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

Shift Register (SR)



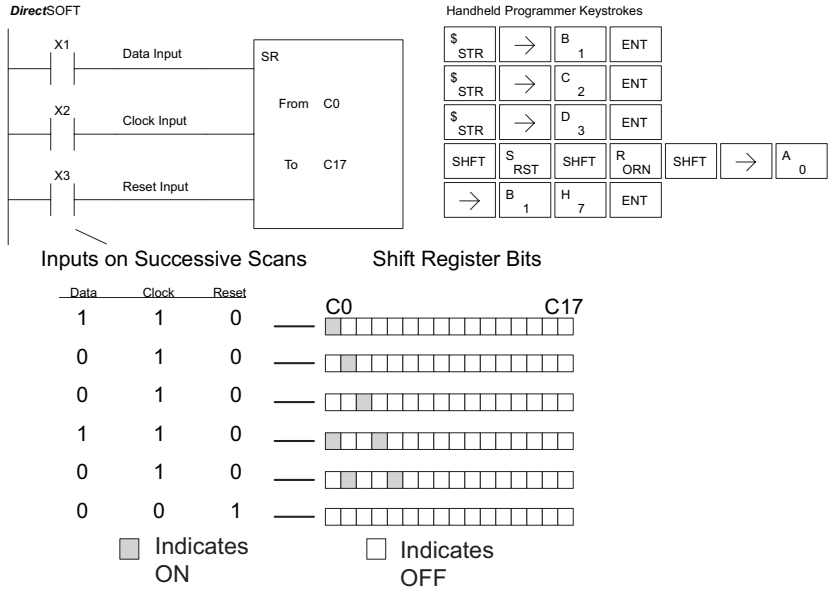
The Shift Register instruction shifts data through a predefined number of control relays. The control ranges in the shift register block must start at the beginning of an 8-bit boundary and use 8-bit blocks.

The Shift Register has three contacts.

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

With each off-to-on transition of the clock input, the bits which make up the shift register block are shifted by one bit position and the status of the data input is placed into the starting bit position in the shift register. The direction of the shift depends on the entry in the From and To fields. From C0 to C17 would define a block of 16 bits to be shifted from left to right. From C17 to C0 would define a block of 16 bits, to be shifted from right to left. The maximum size of the shift register block depends on the number of available control relays. The minimum block size is 8 control relays.

Operand Data Type		DL-130 Range	
	A/B	aaa	bbb
Control Relay	C	0-377	0-377



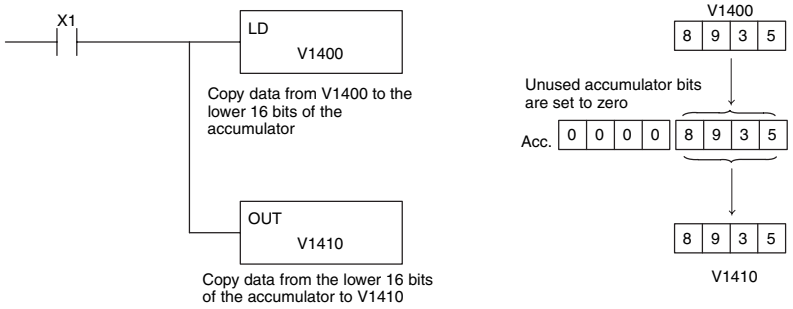
Accumulator/Stack Load and Output Data Instructions

Using the Accumulator

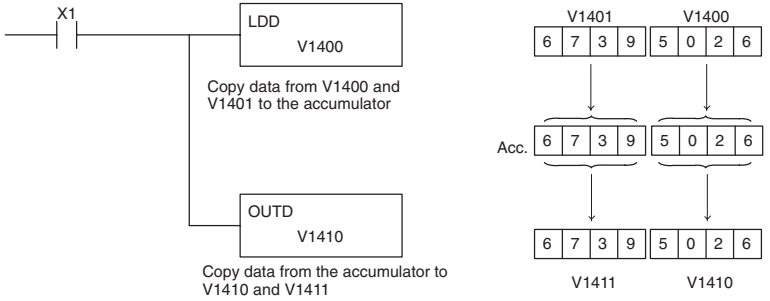
The accumulator in the DL105 series CPUs is a 32-bit register that is used as a temporary storage location for data that is being copied or manipulated in some manner. For example, you have to use the accumulator to perform math operations, such as, add, subtract, multiply, etc. Since there are 32 bits, you can use up to an 8-digit BCD number or a 32-bit 2's compliment number. The accumulator is reset to 0 at the end of every CPU scan.

Copying Data to the Accumulator

The Load and Out instructions and their variations are used to copy data from a V-memory location to the accumulator, or to copy data from the accumulator to V-memory. The following example copies data from V-memory location V1400 to V-memory location V1410.

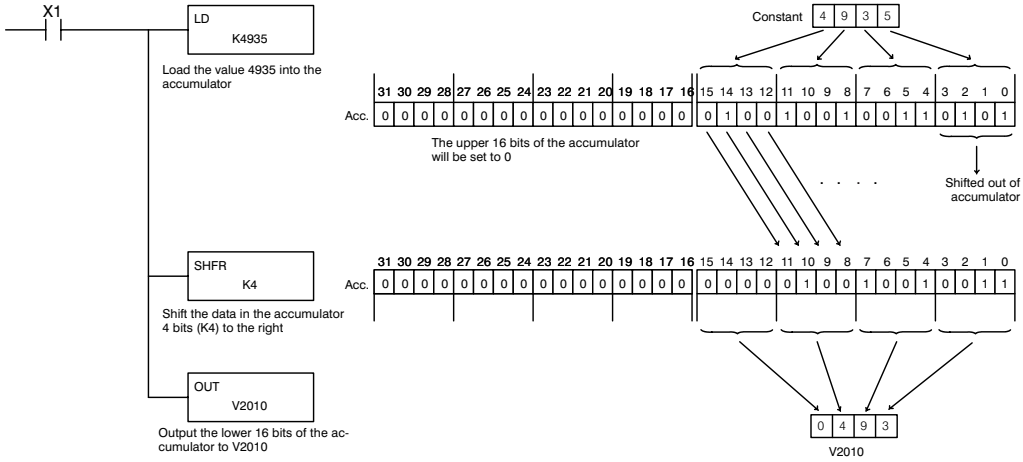


Since the accumulator is 32 bits and V-memory locations are 16 bits, the Load Double and Out Double (or variations thereof) use two consecutive V-memory locations or 8-digit BCD constants to copy data either to the accumulator from a V-memory address or from a V-memory address to the accumulator. For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



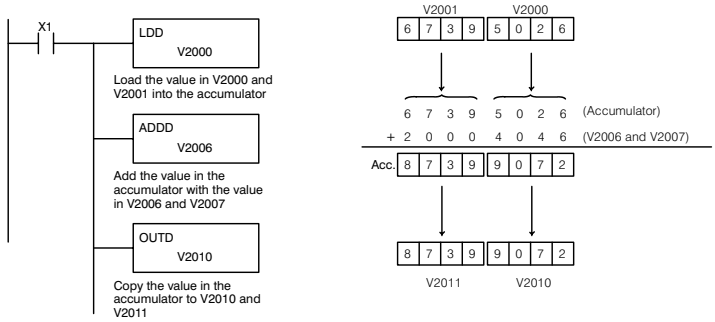
Changing the Accumulator Data

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 value 4935 into the accumulator, shifts the data right 4 bits, and outputs the result to V2010.



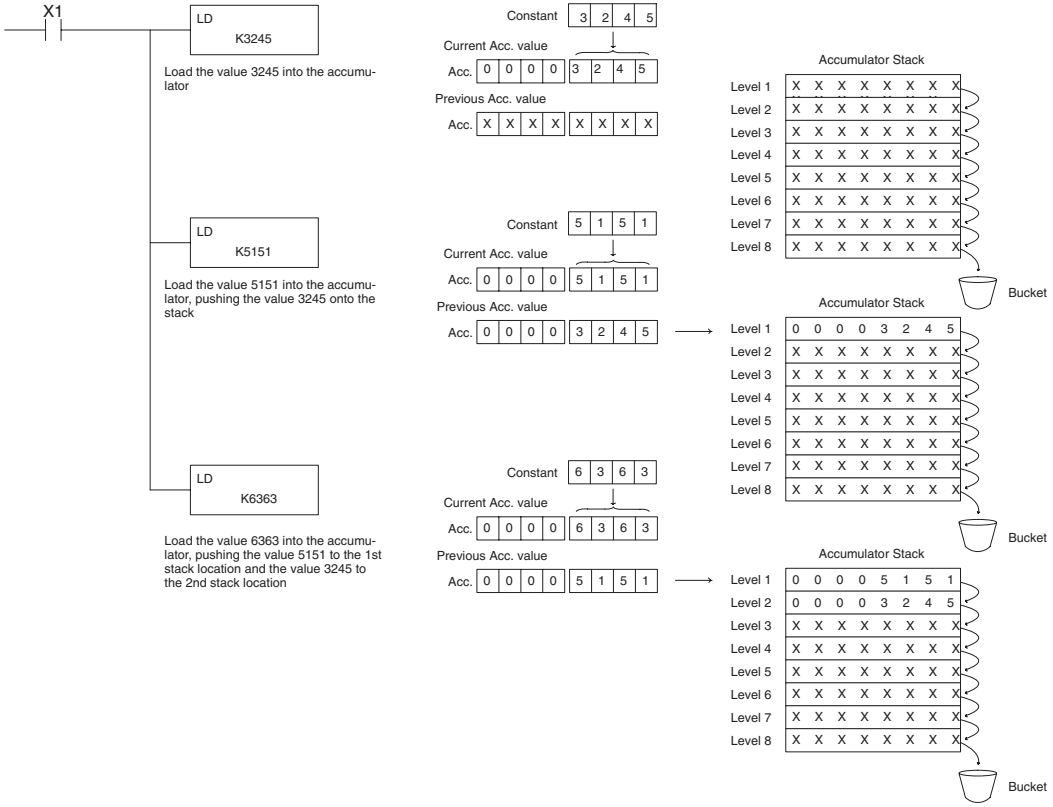
Some of the data manipulation instructions use 32 bits. They use two consecutive V memory locations or an 8 digit BCD constant to manipulate data in the accumulator.

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

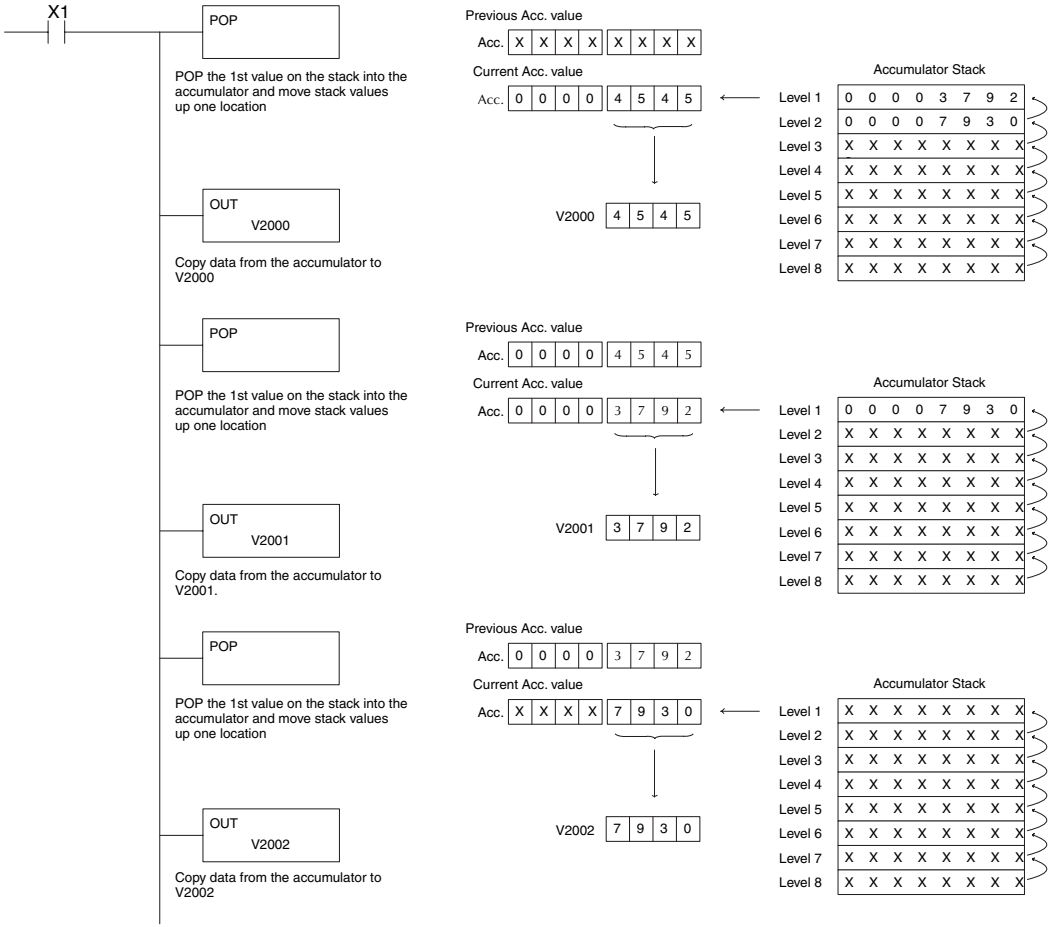


Using the Accumulator Stack

The accumulator stack is used for instructions that require more than one parameter to execute a function or for user-defined functionality. The accumulator stack is used when more than one Load instruction is executed without the use of an Out instruction. The first Load instruction in the scan places a value into the accumulator. Every Load instruction thereafter without the use of an Out instruction places a value into the accumulator and the value that was in the accumulator is placed onto the accumulator stack. The Out instruction nullifies the previous Load instruction and does not place the value that was in the accumulator onto the accumulator stack when the next Load instruction is executed. Every time a value is placed onto the accumulator stack, the other values in the stack are pushed down one location. The accumulator is eight levels deep (eight 32-bit registers). If there is a value in the eighth location when a new value is placed onto the stack, the value in the eighth location is pushed off the stack and cannot be recovered.



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

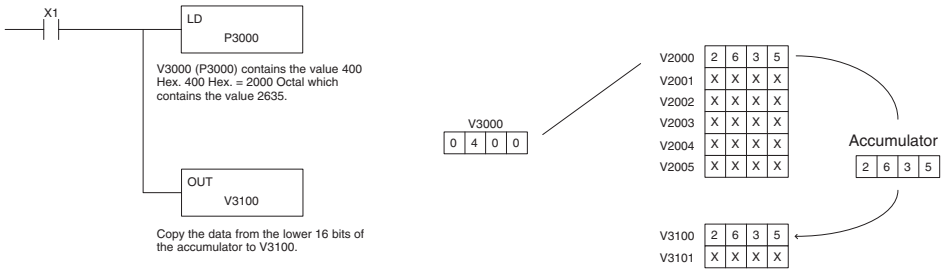

X1
POP
POP the 1st value on the stack into the accumulator and move stack values up one location
OUT
V2000
Copy data from the accumulator to V2000
POP
POP the 1st value on the stack into the accumulator and move stack values up one location
OUT
V2001
Copy data from the accumulator to V2001.
POP
POP the 1st value on the stack into the accumulator and move stack values up one location
OUT
V2002
Copy data from the accumulator to V2002
Previous Acc. value
Acc. X X X X X X X X
Current Acc. value
Acc. 0 0 0 0 4 5 4 5
V2000 4 5 4 5
Accumulator Stack
Level 1 0 0 0 0 3 7 9 2
Level 2 0 0 0 0 7 9 3 0
Level 3 X X X X X X X X
Level 4 X X X X X X X X
Level 5 X X X X X X X X
Level 6 X X X X X X X X
Level 7 X X X X X X X X
Level 8 X X X X X X X X
Previous Acc. value
Acc. 0 0 0 0 4 5 4 5
Current Acc. value
Acc. 0 0 0 0 3 7 9 2
V2001 3 7 9 2
Accumulator Stack
Level 1 0 0 0 0 7 9 3 0
Level 2 X X X X X X X X
Level 3 X X X X X X X X
Level 4 X X X X X X X X
Level 5 X X X X X X X X
Level 6 X X X X X X X X
Level 7 X X X X X X X X
Level 8 X X X X X X X X
Previous Acc. value
Acc. 0 0 0 0 3 7 9 2
Current Acc. value
Acc. X X X X 7 9 3 0
V2002 7 9 3 0
Accumulator Stack
Level 1 X X X X X X X X
Level 2 X X X X X X X X
Level 3 X X X X X X X X
Level 4 X X X X X X X X
Level 5 X X X X X X X X
Level 6 X X X X X X X X
Level 7 X X X X X X X X
Level 8 X X X X X X X X

Using Pointers

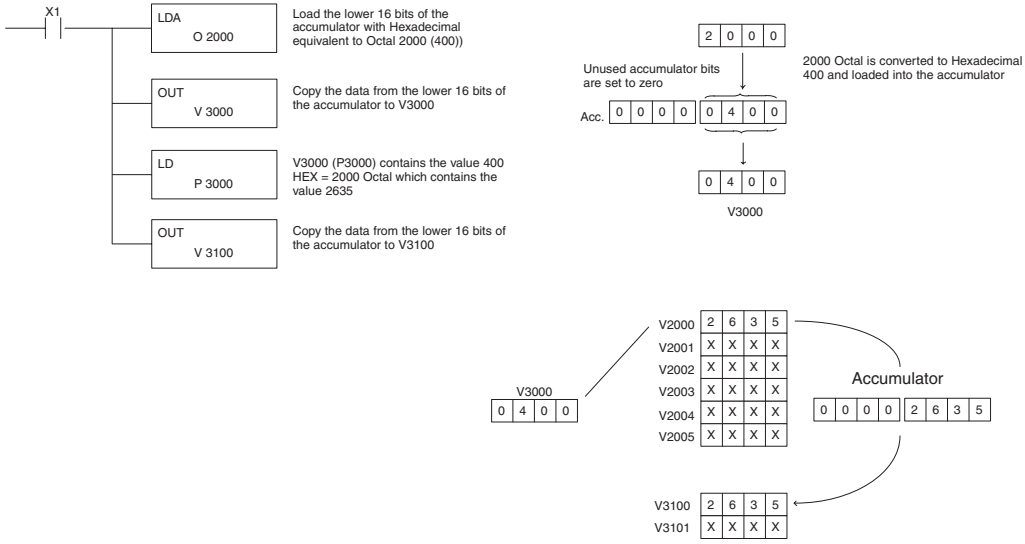
Many of the DL105 series instructions will allow V-memory pointers as an operand. Pointers can be useful in ladder logic programming, but can be difficult to understand or implement in your application if you do not have prior experience with pointers (commonly known as indirect addressing). Pointers allow instructions to obtain data from V-memory locations referenced by the pointer value.

NOTE: *In the DL105, 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 an address into the pointer location. This instruction performs the Octal to Hexadecimal conversion automatically.*

The following example uses a pointer operand in a Load instruction. V-memory location 3000 is the pointer location. V3000 contains the value 400, which is the HEX equivalent of the Octal address V-memory location V2000. The CPU copies the data from V2000 into the lower word of the accumulator.

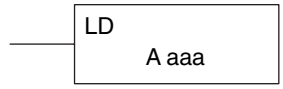


The following example is similar to the one above, except for the LDA (load address) instruction, which automatically converts the Octal address to the Hex equivalent.



Load (LD)

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

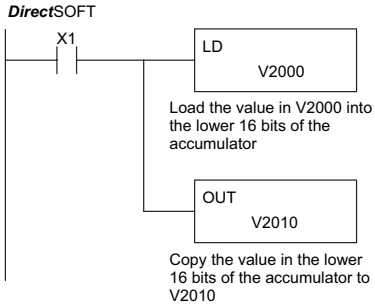


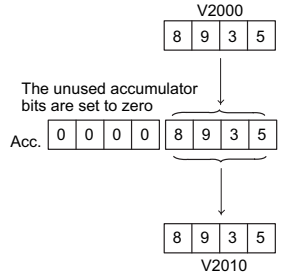
Operand Data Type		DL130 Range
	A	aaa
V-memory for preset values	V	All (See page 4-29)
Pointers (presets only)	P	All V mem (See page 4-29)
Constants (presets only)	K	0–FFFF

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

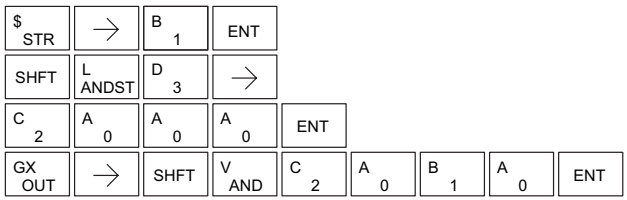
NOTE: *Two consecutive Load instructions will place the value of the first Load instruction onto the accumulator stack.*

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



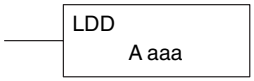


Handheld Programmer Keystrokes



Load Double (LDD)

The Load Double instruction is a 32-bit instruction that loads the value (Aaaa), which is either two consecutive V-memory locations or an 8-digit constant value, into the accumulator.

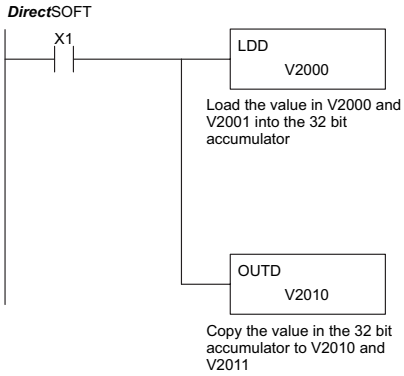


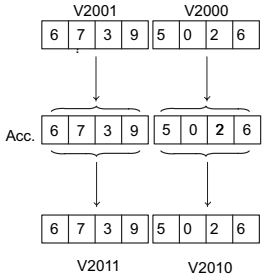
Operand Data Type		DL130 Range
	A	aaa
V-memory	V	All (See page 4-29)
Pointer	P	All V mem (See page 4-29)
Constant	K	0–FFFFFFFF

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

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

In the following example, when X1 is on, the 32-bit value in V2000 and V2001 will be loaded into the accumulator and output to V2010 and V2011.





Handheld Programmer Keystrokes

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

Load Formatted (LDF)

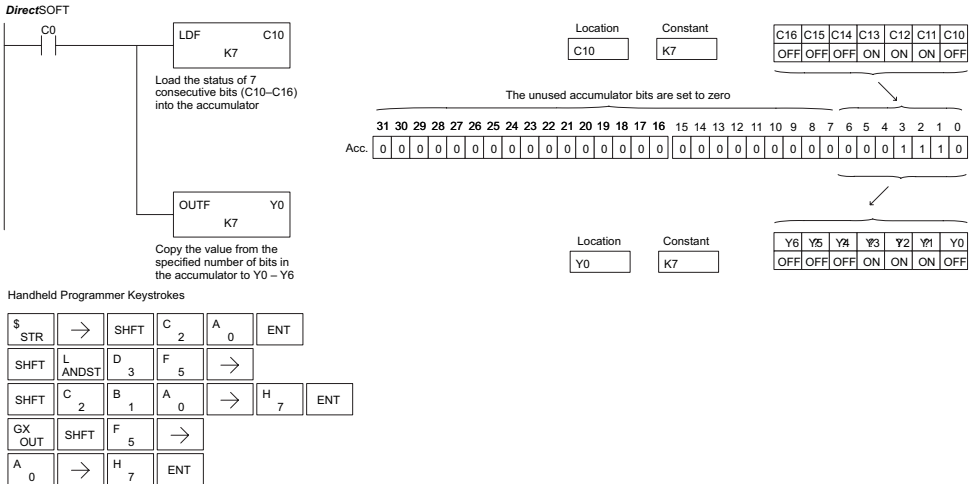
The Load Formatted instruction loads 1 to 32 consecutive bits from discrete memory locations into the accumulator. The instruction requires a starting location (Aaaa) and the number of bits (Kbbb) to be loaded. Unused accumulator bit locations are set to zero.

Operand Data Type		DL130 Range	
	A	aaa	bbb
Inputs	X	0–1	–
Outputs	Y	0–7	–
Control Relays	C	0–377	–
Stage bits	S	0–377	–
Timer bits	T	0–77	–
Counter bits	CT	0–77	–
Special Relays	SP	0–117, 540–577	–
Constant	K	–	1–32

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

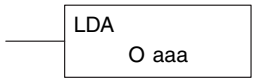
NOTE: *Two consecutive Load instructions will place the value of the first Load instruction onto the accumulator stack.*

In the following example, when C0 is on, the binary pattern of C10–C16 (7 bits) will be loaded into the accumulator using the Load Formatted instruction. The lower 7 bits of the accumulator are output to Y0–Y6 using the Out Formatted instruction.



Load Address (LDA)

The Load Address instruction is a 16-bit instruction. It converts any octal value or address to the HEX equivalent value and loads the HEX value into the accumulator. This instruction is useful when an address parameter is required since all addresses for the DL205 system are in octal.

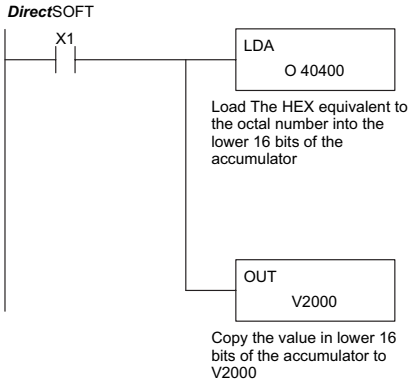


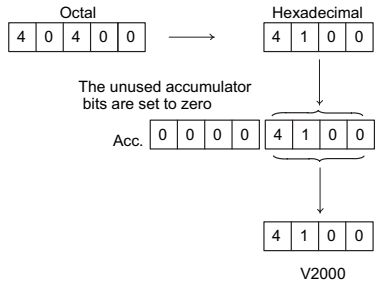
Operand Data Type		DL130 Range
		aaa
Octal Address	O	All V-memory (See page 4-29)

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

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

In the following example, when X1 is on, the octal number 40400 will be converted to a HEX 4100 and loaded into the accumulator using the Load Address instruction. The value in the lower 16 bits of the accumulator is copied to V2000 using the Out instruction.





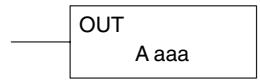
Handheld Programmer Keystrokes

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

Out (OUT)

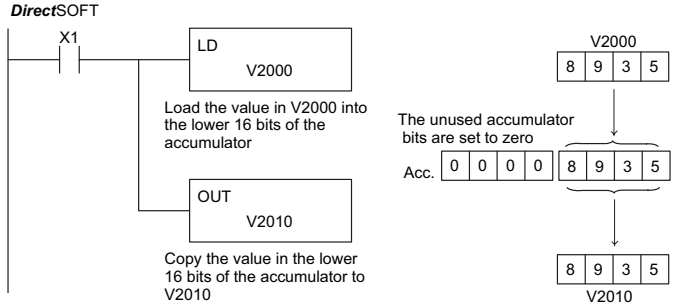
☑ 230
☑ 240
☑ 250-1
☑ 260
☑ 262

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).

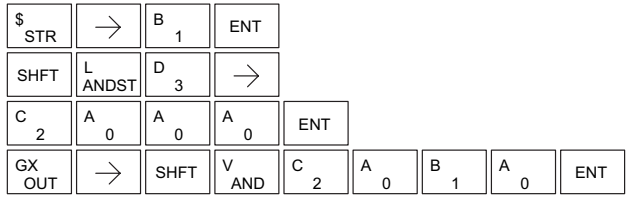


Operand Data Type		DL130 Range
	A	aaa
V-memory	V	All (See page 4-29)
Pointer	P	All V mem (See page 4-29)

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



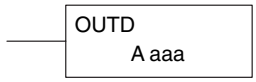
Handheld Programmer Keystrokes



Out Double (OUTD)

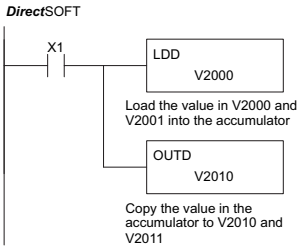
☑ 230
☑ 240
☑ 250-1
☑ 260
☑ 262

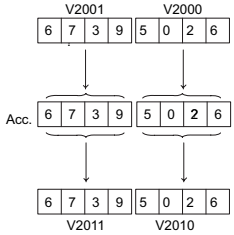
The Out Double instruction is a 32-bit instruction that copies the value in the accumulator to two consecutive V-memory locations at a specified starting location (Aaaa).



Operand Data Type		DL130 Range
	A	aaa
V-memory	V	All (See page 4-29)
Pointer	P	All V mem (See page 4-29)

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



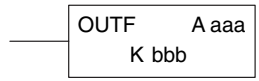


Handheld Programmer Keystrokes

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

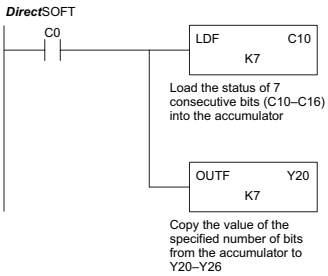
Out Formatted (OUTF)

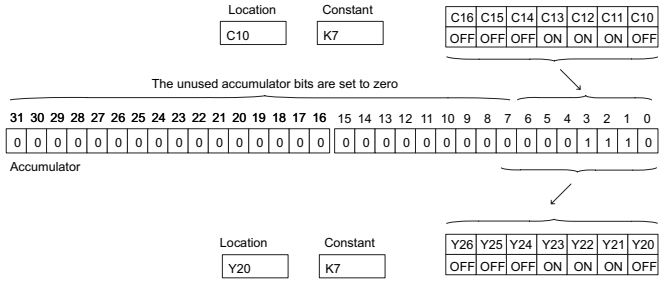
The Out Formatted instruction outputs 1 to 32 bits from the accumulator to the specified discrete memory locations. The instruction requires a starting location (Aaaa) for the destination and the number of bits (Kbbb) to be output.



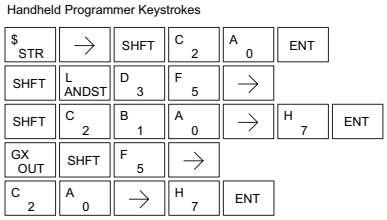
Operand Data Type	DL130 Range		
	A	aaa	bbb
Inputs	X	0–1	–
Outputs	Y	0–7	–
Control Relays	C	0–377	–
Constant	K	–	1–32

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





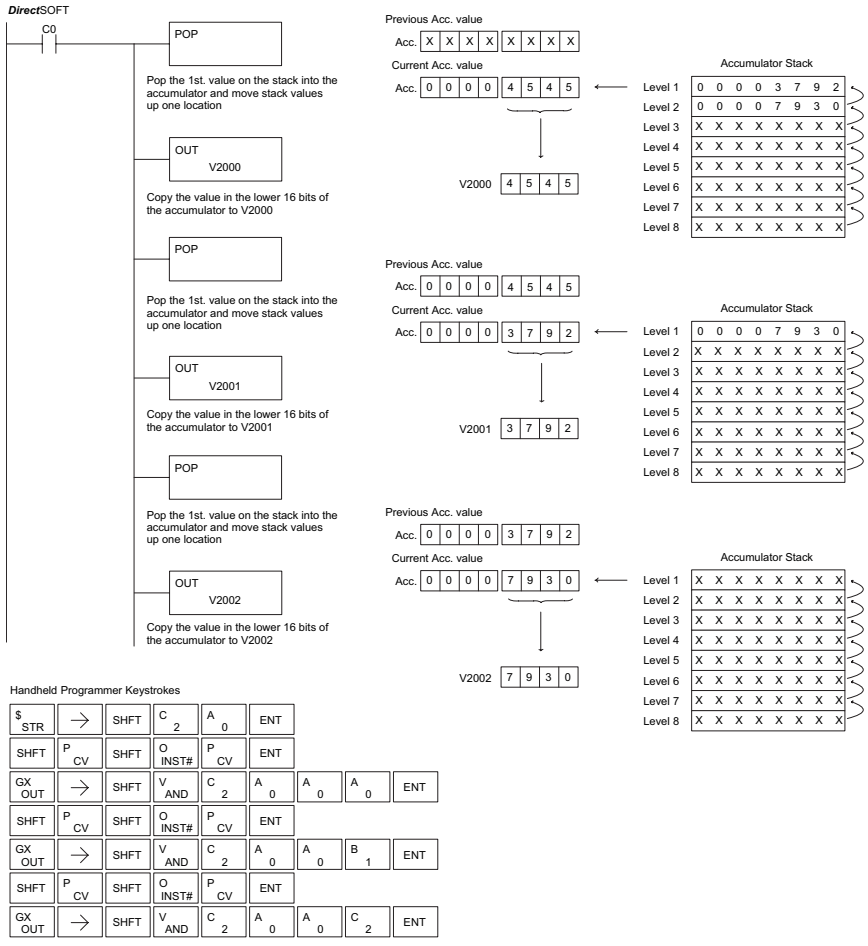
Handheld Programmer Keystrokes



Pop (POP)

The Pop instruction moves the value from the first level of the accumulator stack (32 bits) to the accumulator and shifts each value in the stack up one level. In the example below, when C0 is on, the value 4545 that was on top of the stack is moved into the accumulator using the Pop instruction. The value is output to V2000 using the Out instruction. The next Pop moves the value 3792 into the accumulator and outputs the value to V2001. The last Pop moves the value 7930 into the accumulator and outputs the value to V2002. Please note if the value in the stack were greater than 16 bits (4 digits) the Out Double instruction would be used and two V-memory locations for each Out Double must be allocated.

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



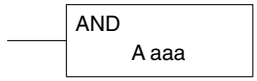
Handheld Programmer Keystrokes

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

Logical Instructions (Accumulator)

And (AND)

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

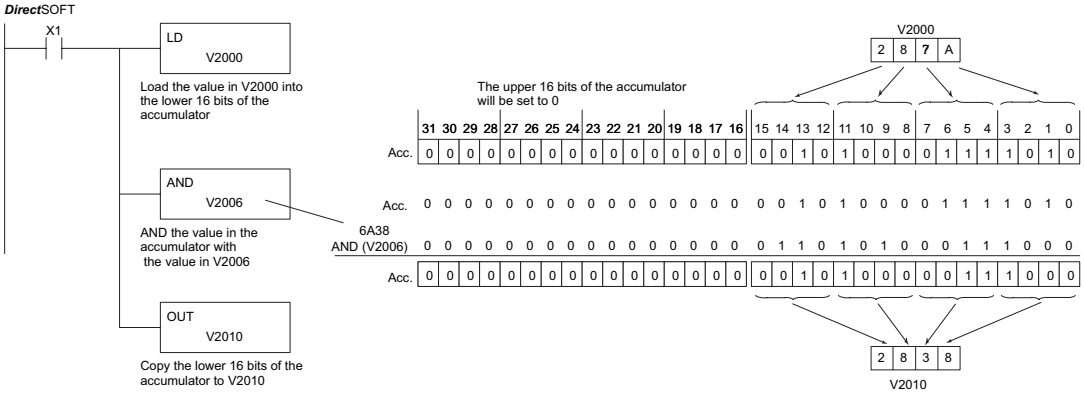


Operand Data Type		DL130 Range
	A	aaa
V-memory	V	All (See page 4-29)

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

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

In the following example, when X1 is on, the value in V2000 will be loaded into the accumulator using the Load instruction. The value in the accumulator is anded with the value in V2006 using the And instruction. The value in the lower 16 bits of the accumulator is output to V2010 using the Out instruction.

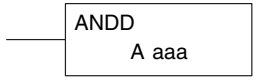


Handheld Programmer Keystrokes

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

And Double (ANDD)

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

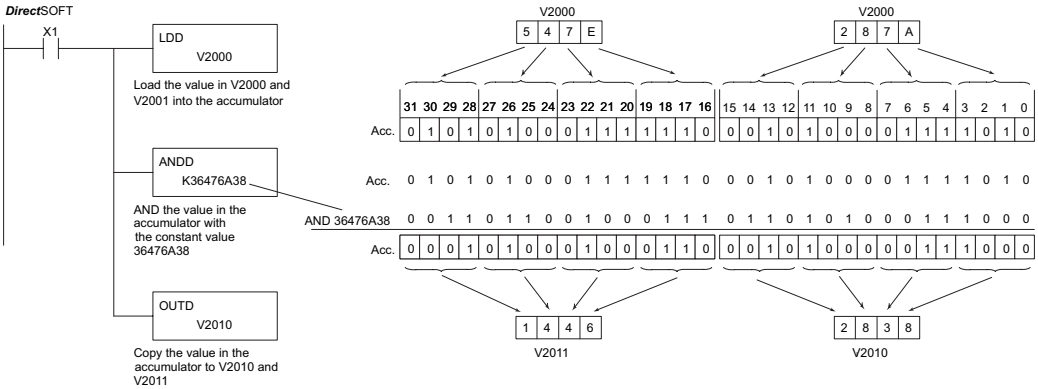


Operand Data Type		DL130 Range
	A	aaa
V-memory	V	All (See page 4-29)
Constant	K	0–FFFFFFFF

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

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

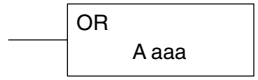


Handheld Programmer Keystrokes

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

Or (OR)

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

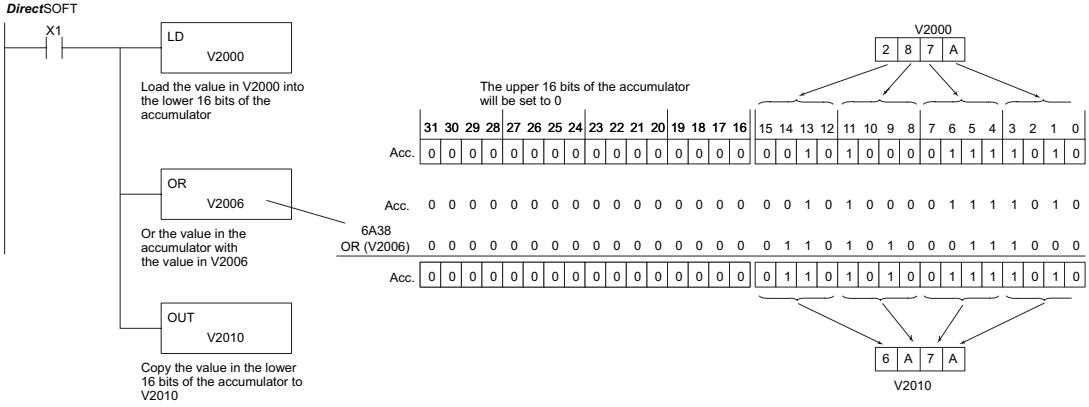


Operand Data Type		DL130 Range
	A	aaa
V-memory	V	All (See page 4-29)

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

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

In the following example, when X1 is on, the value in V2000 will be loaded into the accumulator using the Load instruction. The value in the accumulator is OR'd with V2006 using the OR instruction. The value in the lower 16 bits of the accumulator are output to V2010 using the Out instruction.

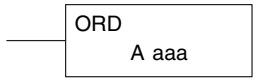


Handheld Programmer Keystrokes

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

Or Double (ORD)

The Or Double is a 32-bit instruction that ORs the value in the accumulator with the value (Aaaa) 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).

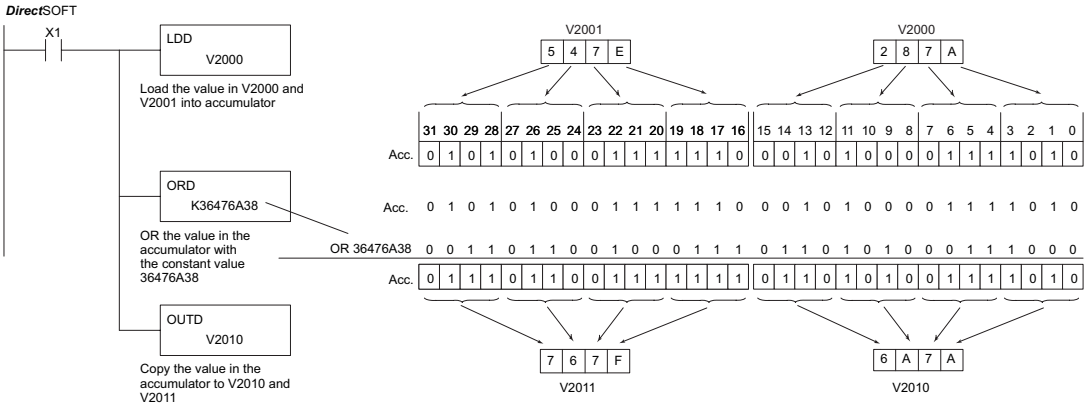


Operand Data Type		DL130 Range
	A	aaa
V-memory	V	All (See page 4-29)
Constant	K	0–FFFFFFFF

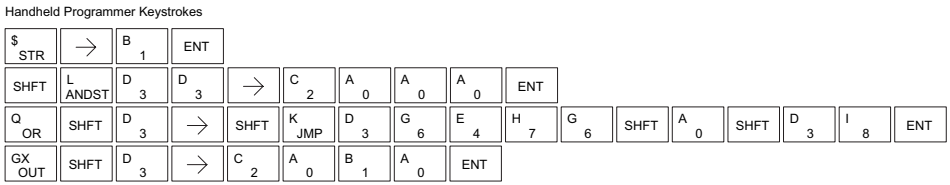
Discrete Bit Flags	Description
SP63	Will be ON if the result in the accumulator is zero
SP70	Will be ON if the result in the accumulator is negative

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

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

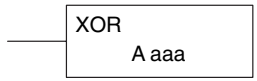


Handheld Programmer Keystrokes



Exclusive Or (XOR)

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

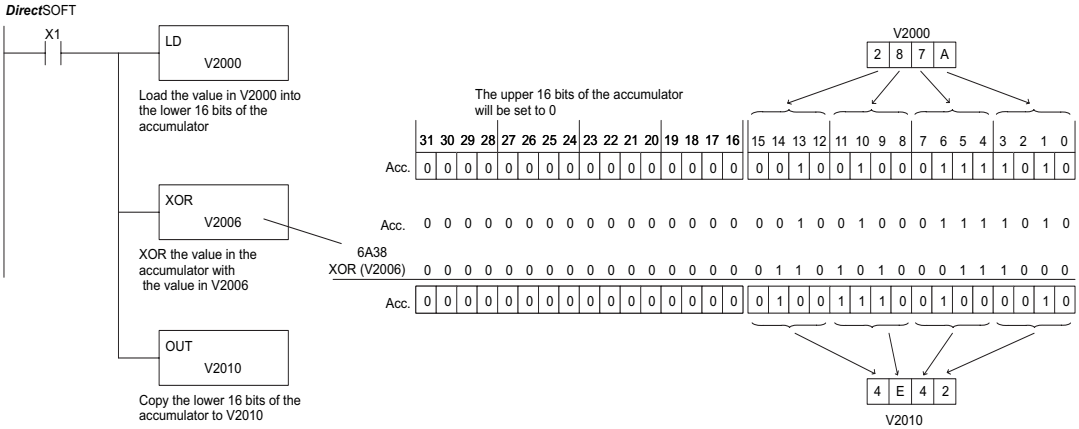


Operand Data Type		DL130 Range
	A	aaa
V-memory	V	All (See page 4-29)

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

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

In the following example, when X1 is on, the value in V2000 will be loaded into the accumulator using the Load instruction. The value in the accumulator is exclusive OR'd with V2006 using the Exclusive Or instruction. The value in the lower 16 bits of the accumulator are output to V2010 using the Out instruction.

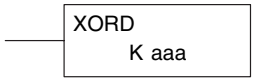


Handheld Programmer Keystrokes

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

Exclusive OR Double (XORD)

The Exclusive OR Double is a 32-bit instruction that performs an exclusive OR of the value in the accumulator and the value (Kaaa), which is 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).

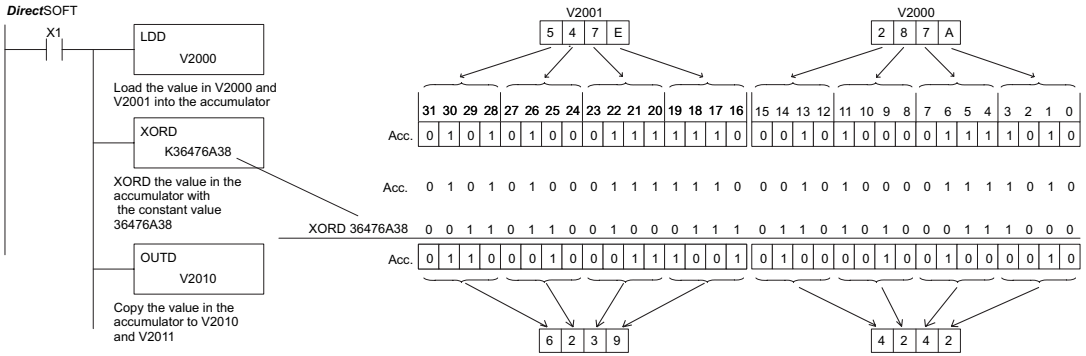


Operand Data Type		DL130 Range
	A	aaa
V-memory	V	All (See page 4-29)
Constant	K	0–FFFFFFFF

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

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

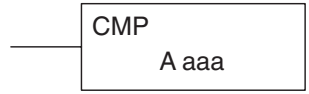


Handheld Programmer Keystrokes

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

Compare (CMP)

The compare instruction is a 16-bit instruction that compares the value in the lower 16 bits of the accumulator with the value in a specified V-memory location (Aaaa). The corresponding status flag will be turned on indicating the result of the comparison.

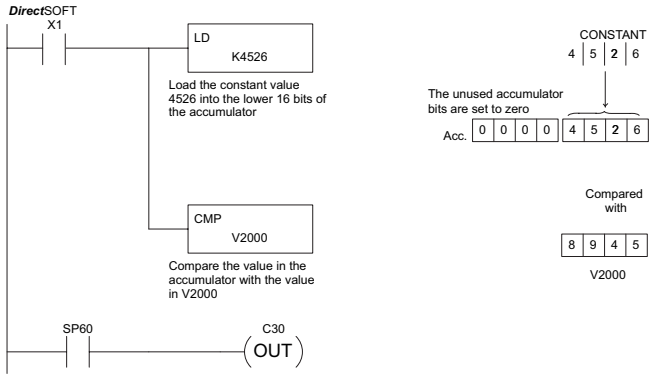


Operand Data Type		DL130 Range
	A	aaa
V-memory	V	All (See page 4-29)

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

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

In the following example, when X1 is on, the constant 4526 will be loaded into the lower 16 bits of the accumulator using the Load instruction. The value in the accumulator is compared with the value in V2000 using the Compare instruction. The corresponding discrete status flag will be turned on indicating the result of the comparison. In this example, if the value in the accumulator is less than the value specified in the Compare instruction, SP60 will turn on, energizing contact C30.

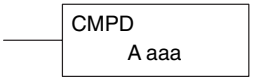


Handheld Programmer Keystrokes

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

Compare Double (CMPD)

The Compare Double instruction is a 32–bit instruction that compares the value in the accumulator with the value (Aaaa), which is either two consecutive V-memory locations or an 8–digit (max) constant. The corresponding status flag will be turned on indicating the result of the comparison.

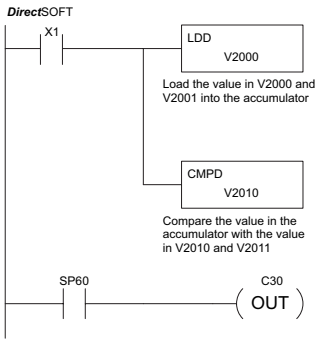


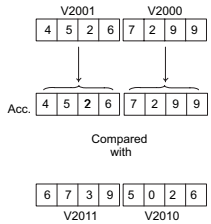
Operand Data Type		DL130 Range
	A	aaa
V-memory	V	All (See page 4-29)
Constant	K	0–FFFFFFFF

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

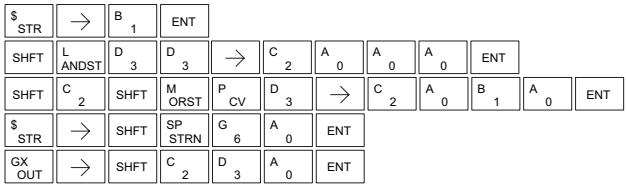
NOTE: *The status flags are only valid until another instruction that uses the same flags is executed.*

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





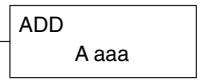
Handheld Programmer Keystrokes



Math Instructions

Add (ADD)

Add is a 16-bit instruction that adds a BCD value in the accumulator with a BCD value in a V-memory location (Aaaa). (You cannot use a constant "K" as the BCD value in the box.) The result resides in the accumulator.

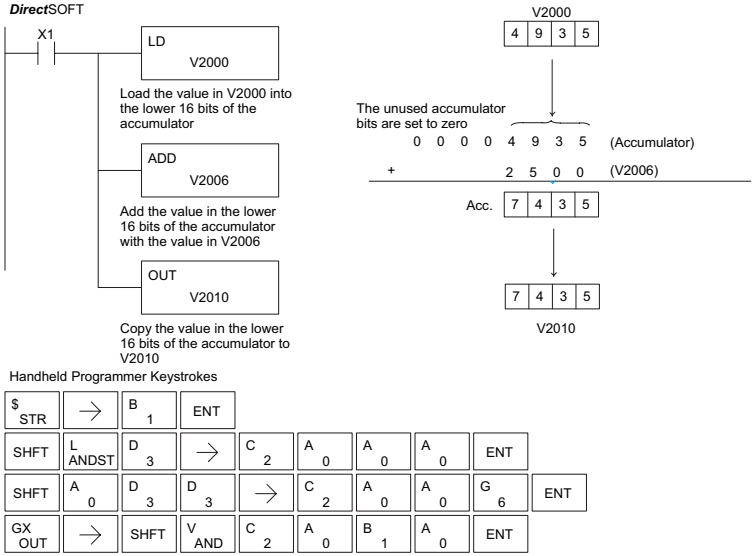


Operand Data Type		DL130 Range
	A	aaa
V-memory	V	All (See page 4-29)

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

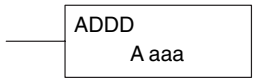
NOTE: *The status flags are only valid until another instruction that uses the same flags is executed.*

In the following example, when X1 is on, the value in V2000 will be loaded into the accumulator using the Load instruction. The value in the lower 16 bits of the accumulator are added to the value in V2006 using the Add instruction. The value in the accumulator is copied to V2010 using the Out instruction.



Add Double (ADDD)

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

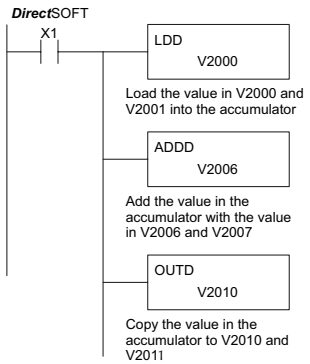


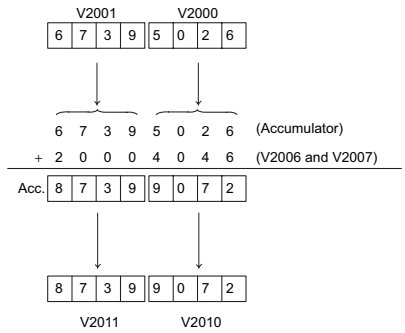
Operand Data Type		DL130 Range
	A	aaa
V-memory	V	All (See page 4-29)
Constant	K	0–99999999

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

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

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



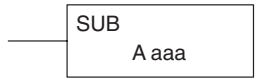


Handheld Programmer Keystrokes

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

Subtract (SUB)

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

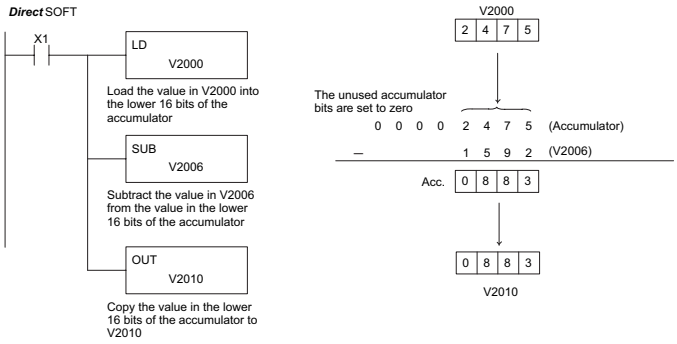


Operand Data Type		DL130 Range
	A	aaa
V-memory	V	All (See page 4-29)

Discrete Bit Flags	Description
SP63	ON when the result of the instruction causes the value in the accumulator to be zero
SP64	ON when the 16 bit addition instruction results in a carry
SP65	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 is encountered

NOTE: *A constant (K) cannot be used for the BCD value.*
Status flags are only valid until another instruction that uses the same flags is executed.

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

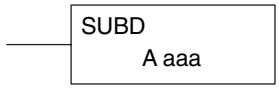


Handheld Programmer Keystrokes

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

Subtract Double (SUBD)

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

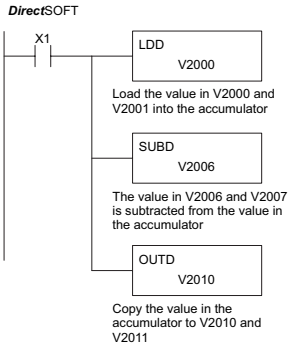


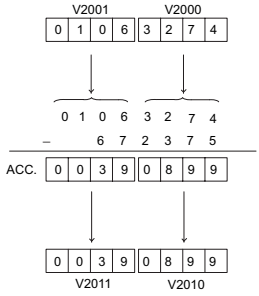
Operand Data Type		DL130 Range
	A	aaa
V-memory	V	All (See page 4-29)
Constant	K	0–99999999

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

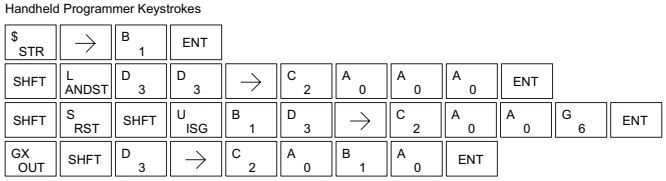
NOTE: *The status flags are only valid until another instruction that uses the same flags is executed.*

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



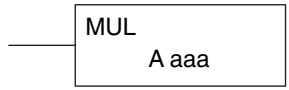


Handheld Programmer Keystrokes



Multiply (MUL)

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

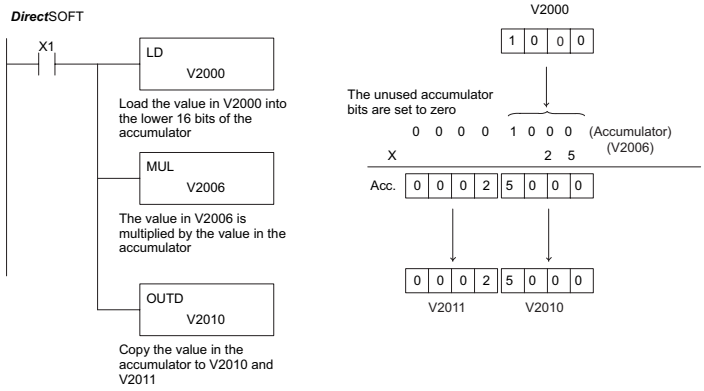


Operand Data Type		DL130 Range
	A	aaa
V-memory	V	All (See page 4-29)
Constant	K	0–9999

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

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

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

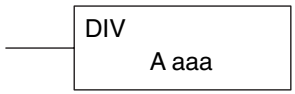


Handheld Programmer Keystrokes

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

Divide (DIV)

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

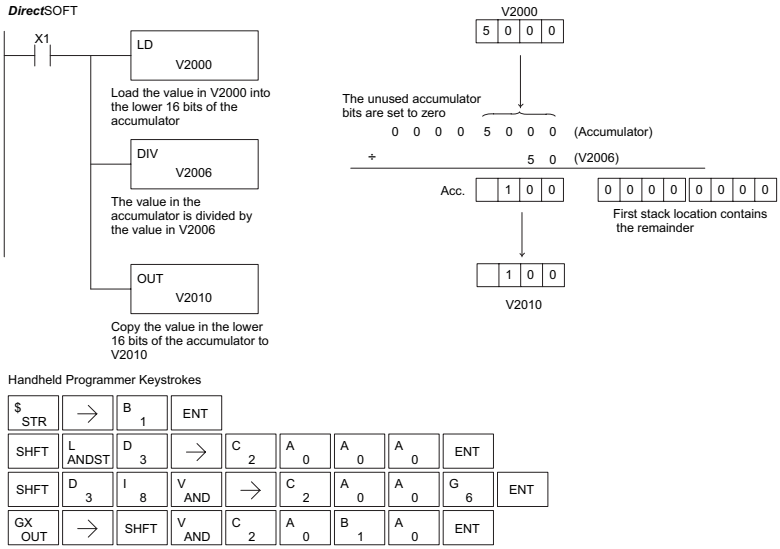


Operand Data Type		DL130 Range
	A	aaa
V-memory	V	All (See page 4-29)
Constant	K	0–9999

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

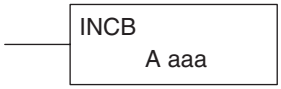
NOTE: *The status flags are only valid until another instruction that uses the same flags is executed.*

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



Increment Binary (INCB)

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



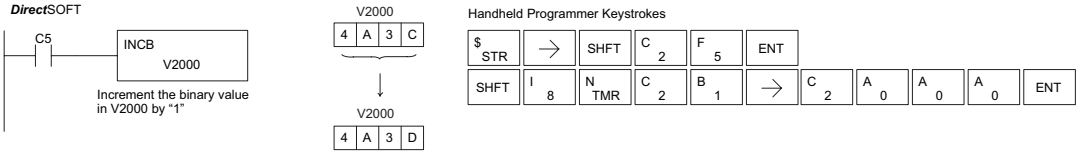
Operand Data Type		DL130 Range
	A	aaa
V-memory	V	All (See page 4-29)

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

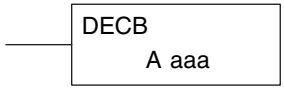
NOTE: *The status flags are only valid until another instruction that uses the same flags is executed.*

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

Decrement Binary (DECB)



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

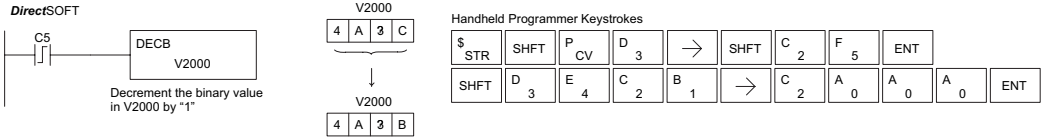


Operand Data Type		DL130 Range
	A	aaa
V-memory	V	All (See page 4-29)

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

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

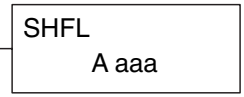
In the following example, when C5 is on, the value in V2000 is decreased by 1.



Bit Operation Instructions

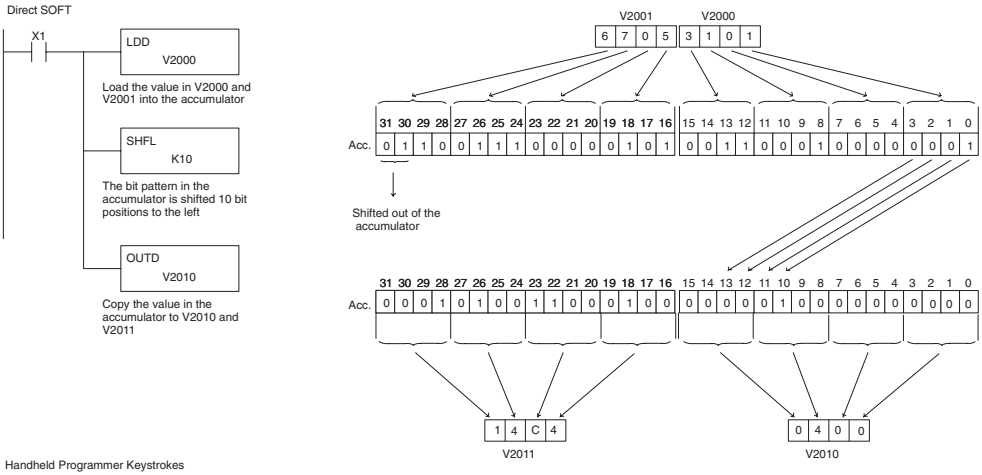
Shift Left (SHFL)

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



Operand Data Type		DL130 Range
	A	aaa
V-memory	V	All (See page 4-29)
Constant	K	0–32

In the following example, when X1 is on, the value in V2000 and V2001 will be loaded into the accumulator using the Load Double instruction. The bit pattern in the accumulator is shifted 10 bits to the left using the Shift Left instruction. The value in the accumulator is copied to V2010 and V2011 using the Out Double instruction.

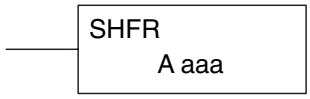


Handheld Programmer Keystrokes

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

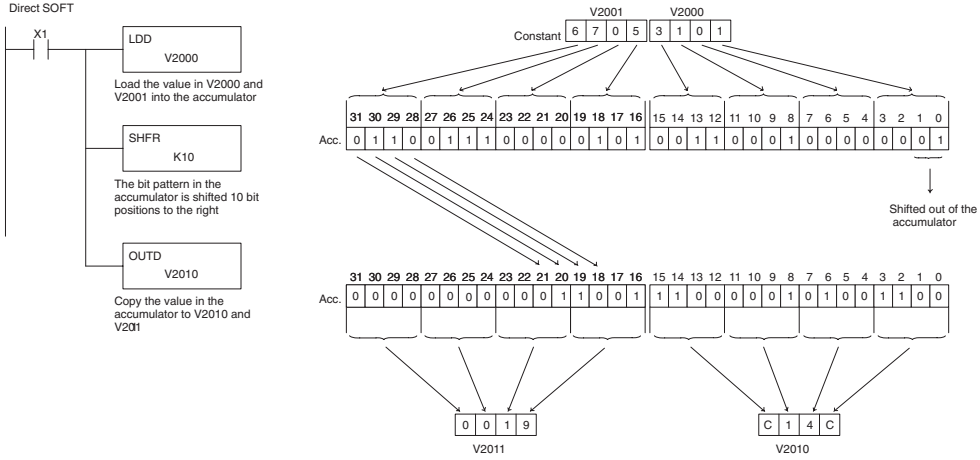
Shift Right (SHFR)

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



Operand Data Type		DL130 Range
	A	aaa
V-memory	V	All (See page 4-29)
Constant	K	0–32

In the following example, when X1 is on, the value in V2000 and V2001 will be loaded into the accumulator using the Load Double instruction. The bit pattern in the accumulator is shifted 10 bits to the right using the Shift Right instruction. The value in the accumulator is copied to V2010 and V2011 using the Out Double instruction.



Handheld Programmer Keystrokes

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

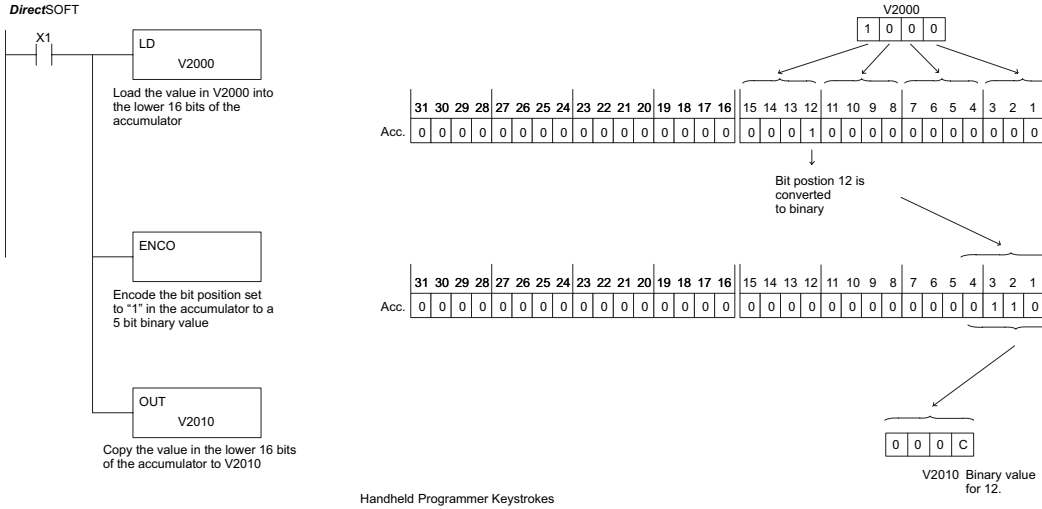
Encode (ENCO)

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

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

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

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



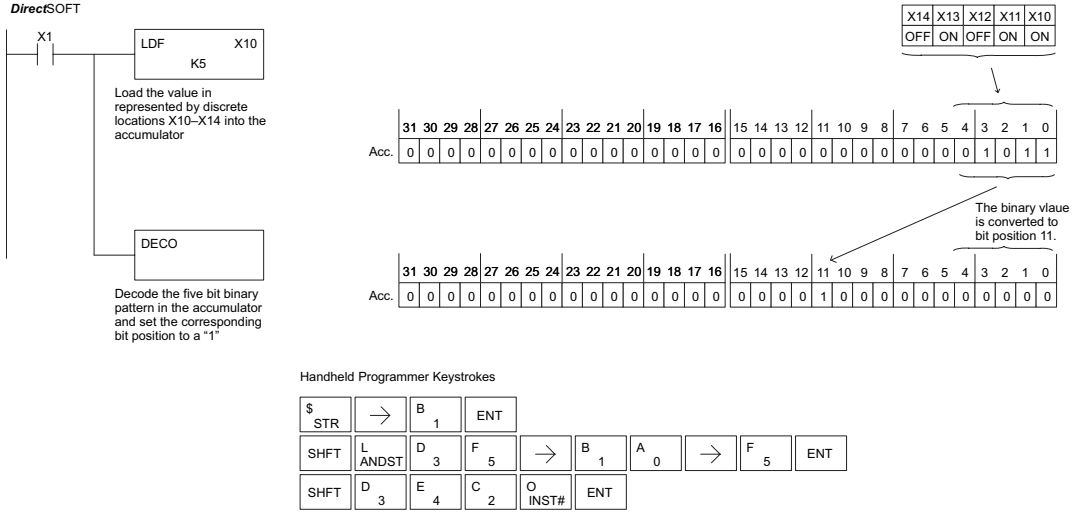
Handheld Programmer Keystrokes

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

Decode (DECO)

The Decode instruction decodes a 5-bit binary value of 0 to 31 (0 to 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 5-bit binary pattern in the accumulator is decoded by setting the corresponding bit position to a "1" using the Decode instruction.



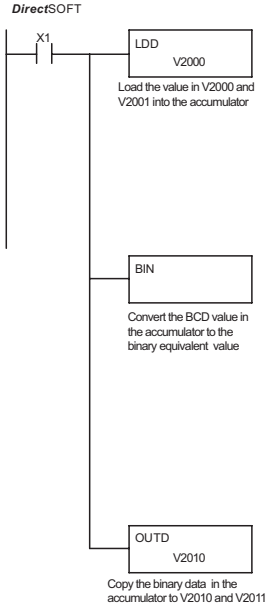
Handheld Programmer Keystrokes

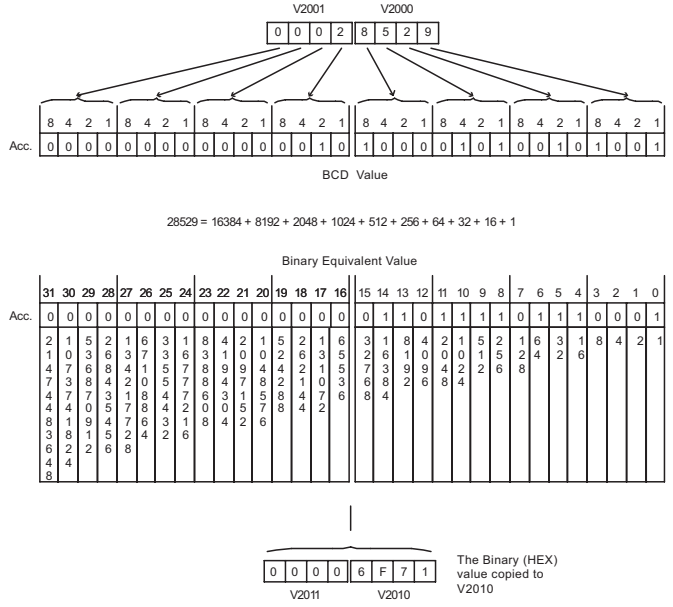
Number Conversion Instructions (Accumulator)

Binary (BIN)

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

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





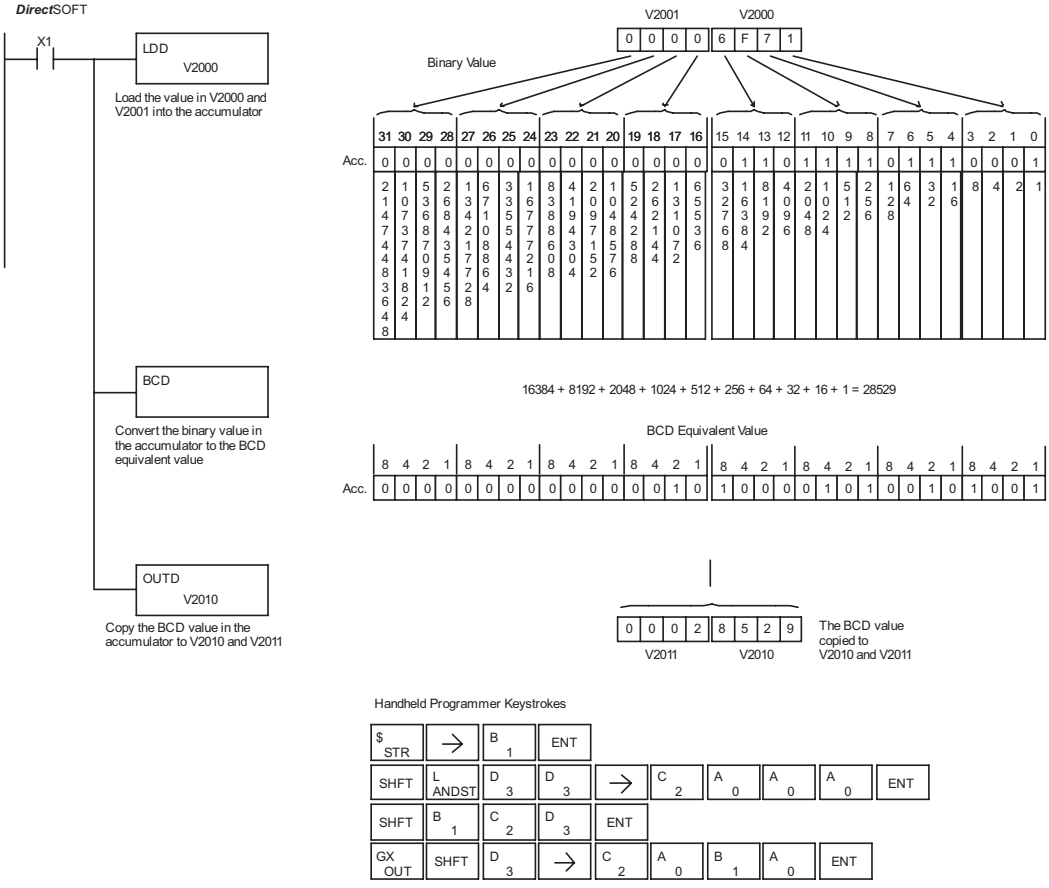
Handheld Programmer Keystrokes

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

Binary Coded Decimal (BCD)

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

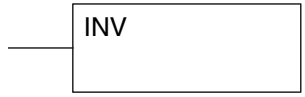
In the following example, when X1 is on, the binary (HEX) value in V2000 and V2001 is loaded into the accumulator using the Load Double instruction. The binary value in the accumulator is converted to the BCD equivalent value using the BCD instruction. The BCD value in the accumulator is copied to V2010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



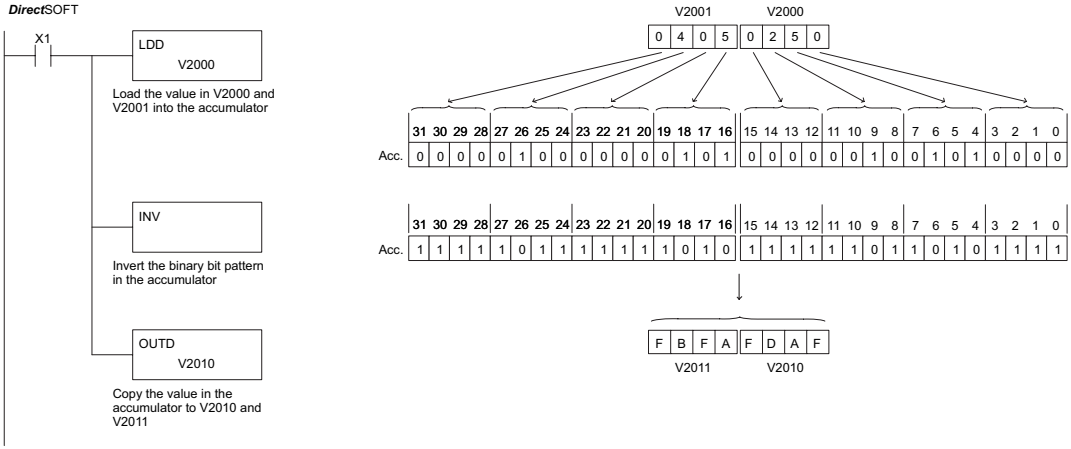
Handheld Programmer Keystrokes

Invert (INV)

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



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



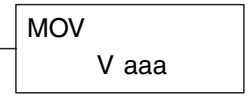
Handheld Programmer Keystrokes

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

Table Instructions

Move (MOV)

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



Step 1: Load the number of V-memory locations to be moved into the first level of the accumulator stack. This parameter is a HEX value (KFFF max, 7777 octal).

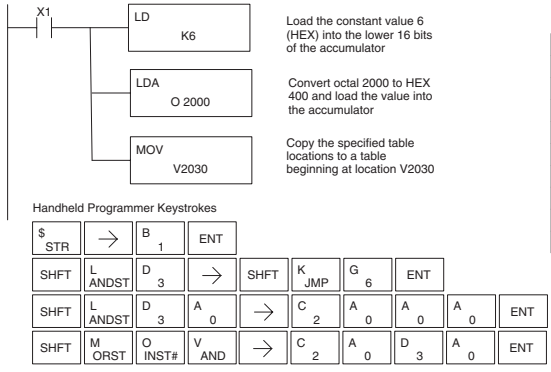
Step 2: Load the starting V-memory location for the locations to be moved into the accumulator. This parameter must be a HEX value.

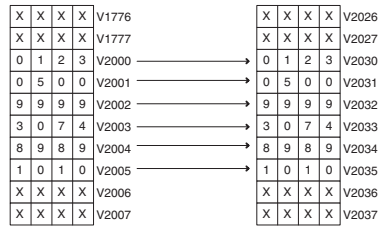
Step 3: Insert the MOVE instruction which specifies starting V-memory location (Vaaa) for the destination table.

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		DL130 Range
	A	aaa
V-memory	V	All (See page 4-28)

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

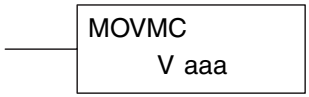




Move Memory Cartridge (MOVMC)

Load Label (LDLBL)

The Move Memory Cartridge instruction is used to copy data between V-memory and program ladder memory. The Load Label instruction is *only* used with the MOVMC instruction when copying data *from* program ladder memory to V-memory.



To copy data between V-memory and program ladder memory, the function parameters are loaded into the first two levels of the accumulator stack and the accumulator by two additional instructions. Listed below are the steps necessary to program the Move Memory Cartridge and Load Label functions.

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

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

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

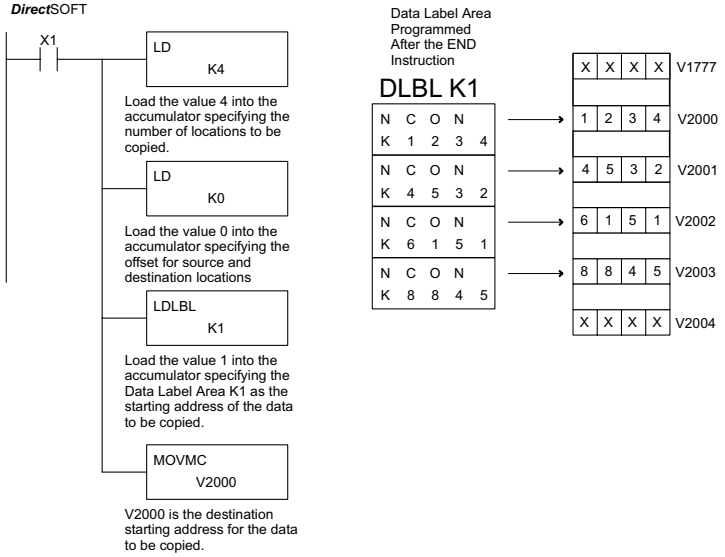
Step 4: Insert the MOVMC instruction which specifies destination (Aaaa). This is where the value will be copied to.

Operand Data Type		DL130 Range
	A	aaa
V-memory	V	All (See page 4-28)

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

Copy Data From a Data Label Area to V-Memory

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



Handheld Programmer Keystrokes

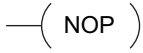
$ STR	→	B 1	ENT														
SHFT	L ANDST	D 3	→	SHFT	K JMP	E 4	ENT										
SHFT	L ANDST	D 3	→	SHFT	K JMP	A 0	ENT										
SHFT	L ANDST	D 3	L ANDST	B 1	L ANDST	→	B 1	ENT									
SHFT	M ORST	O INST#	V AND	M ORST	C 2	→	C 2	A 0	A 0	A 0	ENT						

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.

CPU Control Instructions

No Operation (NOP)

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

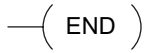


Handheld Programmer Keystrokes

SHFT	N TMR	O INST#	P CV	ENT

End (END)

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

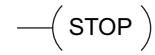


Handheld Programmer Keystrokes

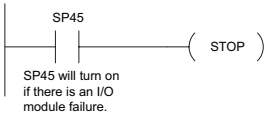
SHFT	E 4	N TMR	D 3	ENT

Stop (STOP)

The Stop instruction changes the operational mode of the CPU from Run to Program (Stop) mode. This instruction is typically used to stop PLC operation in a shutdown condition such as an I/O module failure.



In the following example, when SP45 comes on indicating an I/O module failure, the CPU will stop operation and switch to the program mode.



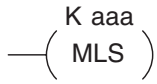
Handheld Programmer Keystrokes

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

Program Control Instructions

Master Line Set (MLS)

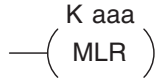
The Master Line Set instruction allows the program to control sections of ladder logic by forming a new power rail controlled by the main left power rail. The main left rail is always master line 0. When an MLS K1 instruction is used, a new power rail is created at level 1. Master Line Sets and Master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



Operand Data Type		DL130 Range
	A	aaa
Constant	K	0–7

Master Line Reset (MLR)

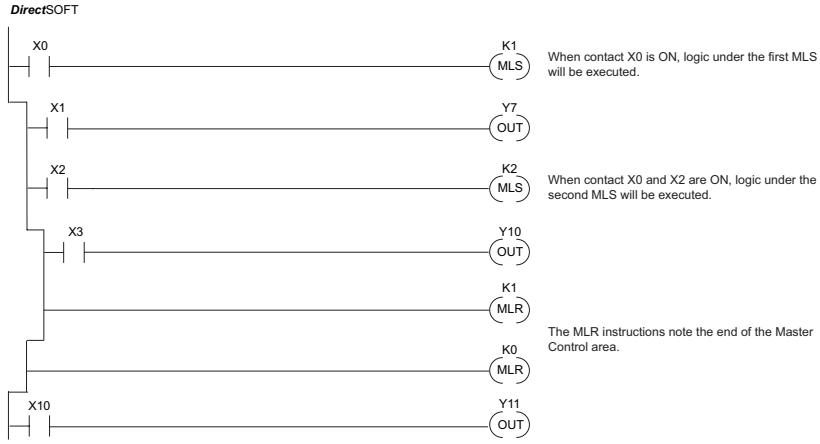
The Master Line Reset instruction marks the end of control for the corresponding MLS instruction. The MLR reference is one less than the corresponding MLS.



Operand Data Type		DL130 Range
	A	aaa
Constant	K	0–6

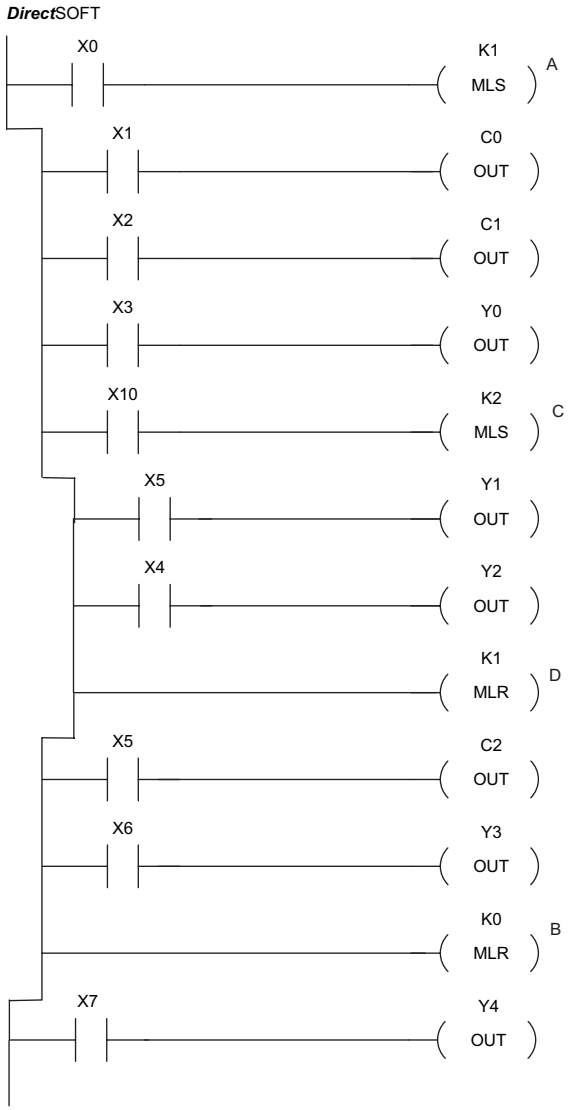
Understanding Master Control Relays

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



MLS/MLR Example

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

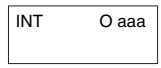


Handheld Programmer Keystrokes

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

Interrupt Instructions

Interrupt (INT)



The Interrupt instruction allows a section of ladder logic to be placed outside the main body of the program and executed only when needed. High-Speed I/O Modes 10, 20, and 40 can generate an interrupt. With Mode 40, you may select an external interrupt (input X0), or a time-based interrupt (5– 999 ms).

Typically, interrupts are used in an application when a fast response to an input is needed or a program section must execute faster than the normal CPU scan. The interrupt label and all associated logic must be placed after the End statement in the program. When an interrupt occurs, the CPU will complete execution of the current instruction it is processing in ladder logic, then execute the interrupt routine. After interrupt routine execution, the ladder program resumes from the point at which it was interrupted.

See Chapter 3, the section on Mode 40 (Interrupt) Operation for more details on interrupt configuration. In the DL105, only one interrupt is available.

Operand Data Type	DL130 Range
Constant 0	0

Interrupt Return (IRT)

When an Interrupt Return is executed in the interrupt routine, the CPU will return to the point in the main body of the program from which it was called. The Interrupt Return is programmed as the last instruction in an interrupt routine and is a stand alone instruction (no input contact on the rung).

Interrupt Return Conditional (IRTC)

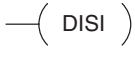
The Interrupt Return Conditional instruction is a optional instruction used with an input contact to implement a conditional return from the interrupt routine. The Interrupt Return is required to terminate the interrupt routine.

Enable Interrupts (ENI)

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

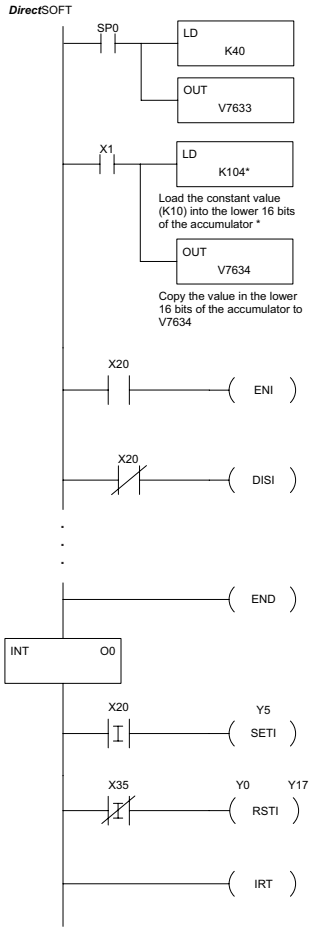
Disable Interrupts (DISI)

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



External Interrupt Program Interrupt

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



Handheld Programmer Keystrokes

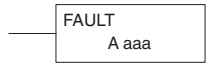
$ STR	→	SHFT	SP STRN	A 0	ENT											
SHFT	L ANDST	D 3	→	SHFT	K JMP	B 4	A 0	ENT								
GX OUT	→	SHFT	V	H 7	G 6	D 3	D 3	ENT								
$ STR	→	B 1	ENT													
SHFT	L ANDST	D 3	→	SHFT	K JMP	B 1	A 0	E 4	ENT							
GX OUT	→	SHFT	V AND	H 7	G 6	D 3	E 4	ENT								
$ STR	→	C 2	A 0	ENT												
SHFT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

* The value entered, 3-999, must be followed by the digit 4 to complete the instruction.

NOTE: *Only one software interrupt is allowed in the DL105 and it must be Int 0.*

Message Instructions

Fault (FAULT)



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

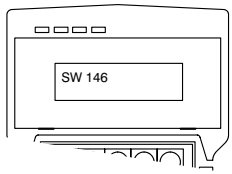
To display the value in a V-memory location, specify the V-memory location in the instruction. To display the data in ACON (ASCII constant) or NCON (Numerical constant) instructions, specify the constant (K) value for the corresponding data label area.

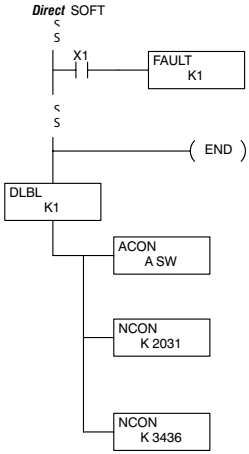
Operand Data Type		DL130 Range
	A	**aaa**
V-memory	V	All (See page 4-28)
Constant	K	1–FFFF

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

Fault Example

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





Handheld Programmer Keystrokes

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

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

Data Label (DLBL)

The Data Label instruction marks the beginning of an ASCI
are programmed after the End statement. A maximum
instructions can be used in a program. Multiple NCON
can be used in a DLBL area.

s
s

DLBL
K aaa

Operand Data Type		DL130 Range
		aaa
Constant	K	1–FFFF

ASCII Constant (ACON)

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

ACON
A aaa

Operand Data Type		DL130 Range
		aaa
ASCII	A	0–9 A–Z

Numerical Constant (NCON)

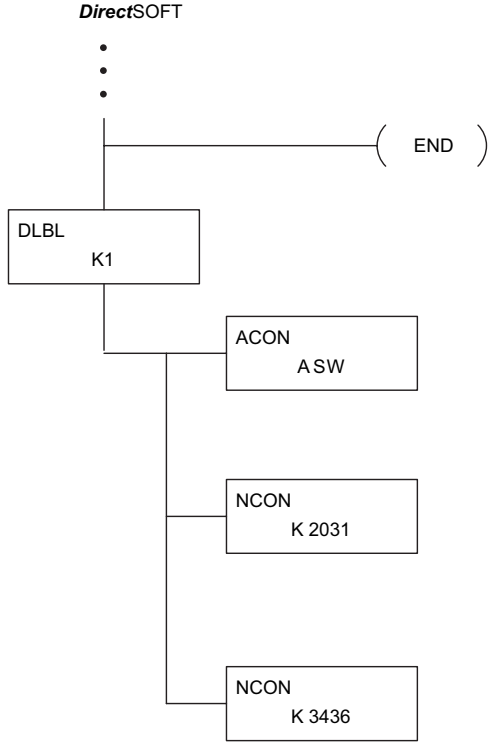
The Numerical Constant instruction is used with the DLBL instruction to store the HEX ASCII equivalent of numerical data for use with other instructions. Two digits can be stored in an NCON instruction.

NCON
K aaa

Operand Data Type		DL130 Range
		aaa
Constant	K	0–FFFF

Data Label Example

In the following example, an ACON and two NCON instructions are used within a DLBL instruction to build a text message. See the FAULT instruction for information on displaying messages. The DV-1000 Manual also has information on displaying messages.



Handheld Programmer Keystrokes

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