

NOTE: As of 07/2021 CPU D2-260 has been retired. Please consider CPU D2-262 as a replacement.

RLL AND INTELLIGENT BOX INSTRUCTIONS



CHAPTER 5

In This Chapter...

Introduction	5-2
Using Boolean Instructions	5-5
Boolean Instructions	5-10
Comparative Boolean	5-27
Immediate Instructions	5-33
Timer, Counter and Shift Register Instructions.....	5-41
Accumulator/Stack Load and Output Data Instructions	5-53
Logical Instructions (Accumulator)	5-71
Math Instructions.....	5-88
Transcendental Functions (D2-260 and D2-262 only)	5-121
Bit Operation Instructions	5-123
Number Conversion Instructions (Accumulator).....	5-130
Table Instructions.....	5-144
Clock/Calendar Instructions.....	5-175
CPU Control Instructions	5-177
Program Control Instructions.....	5-179
Interrupt Instructions.....	5-187
Intelligent I/O Instructions	5-191
Network Instructions	5-193
Message Instructions	5-197
Modbus RTU Instructions (D2-260/D2-262).....	5-205
ASCII Instructions (D2-260/D2-262)	5-211
Intelligent Box (IBox) Instructions (D2-250-1, D2-260 and D2-262 Only)	5-230

Introduction

The DL205 CPUs offer a wide variety of instructions to perform many different types of operations. Several instructions are not available in all of the CPUs. This chapter shows you how to use these individual instructions. There are two ways to quickly find the instruction you need:

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

Instruction		Page
ACON	ASCII Constant	5-199
ACOSR	Arc Cosine Real	5-122
ACRB	ASCII Clear Buffer	5-229
ADD	Add BCD	5-88
ADDB	Add Binary	5-101
ADDBD	Add Binary Double	5-102
ADDBS	Add Binary Top of Stack	5-117
ADDD	Add Double BCD	5-89
ADDF	Add Formatted	5-109
ADDR	Add Real	5-90
ADDs	Add Top of Stack	5-113
AEX	ASCII Extract	5-220
AFIND	ASCII Find	5-217
AIN	ASCII IN	5-212
AND	And for contacts or boxes	5-14, 5-32, 5-71
AND STR	And Store	5-16
ANDB	And Bit-of-Word	5-15
ANDD	And Double	5-72
ANDE	And if Equal	5-29
ANDF	And Formatted	5-73
ANDI	And Immediate	5-35
ANDMOV	And Move	5-171
ANDN	And Not	5-14, 5-32
ANDNB	And Not Bit-of-Word	5-15
ANDND	And Negative Differential	5-23
ANDNE	And if Not Equal	5-29
ANDNI	And Not Immediate	5-35
ANDPD	And Positive Differential	5-23
ANDS	And Stack	5-74
ASINR	Arc Sine Real	5-121
ATANR	Arc Tangent Real	5-122
ATH	ASCII to Hex	5-137
ATT	Add to Top of Table	5-166
BCD	Binary Coded Decimal	5-131
BCDCPL	Tens Complement	5-133

Instruction		Page
BIN	Binary	5-130
BCALL	Block Call (Stage)	7-27
BEND	Block End (Stage)	7-27
BLK	Block (Stage)	7-27
BTOR	Binary to Real	5-134
CMP	Compare	5-83
CMPD	Compare Double	5-84
CMPF	Compare Formatted	5-85
CMPR	Compare Real Number	5-87
CMPS	Compare Stack	5-86
CMPV	ASCII Compare	5-221
CNT	Counter	5-46
COSR	Cosine Real	5-121
CV	Converge (Stage)	7-25
CVJMP	Converge Jump (Stage)	7-25
DATE	Date	5-175
DEC	Decrement	5-100
DECB	Decrement Binary	5-108
DECO	Decode	5-129
DEGR	Degree Real Conversion	5-136
DISI	Disable Interrupts	5-188
DIV	Divide	5-97
DIVB	Divide Binary	5-106
DIVBS	Divide Binary Top of Stack	5-120
DIVD	Divide Double	5-98
DIVF	Divide Formatted	5-112
DIVR	Divide Real Number	5-99
DIVS	Divide Top of Stack	5-116
DLBL	Data Label	5-199
DRUM	Timed Drum	6-12
EDRUM	Event Drum	6-14
ENCO	Encode	5-128
END	End	5-177
ENI	Enable Interrupts	5-188

Instruction		Page
FAULT	Fault	5-197
FDGT	Find Greater Than	5-152
FILL	Fill	5-150
FIND	Find	5-151
FINDB	Find Block	5-173
FOR	For/Next	5-180
GOTO	Goto/Label	5-179
GRAY	Gray Code	5-141
GTS	Goto Subroutine	5-182
HTA	Hex to ASCII	5-138
INC	Increment	5-100
INCB	Increment Binary	5-107
INT	Interrupt	5-187
INV	Invert	5-132
IRT	Interrupt Return	5-188
IRTC	Interrupt Return Conditional	5-188
ISG	Initial Stage	7-24
JMP	Jump	7-24
LBL	Label	5-179
LD	Load	5-58
LDI	Load Immediate	5-39
LDIF	Load Immediate Formatted	5-40
LDA	Load Address	5-61
LDD	Load Double	5-59
LDF	Load Formatted	5-60
LDR	Load Real Number	5-64
LDX	Load Indexed	5-62
LDLBL	Load Label	5-145
LDSX	Load Indexed from Constant	5-63
MDRMD	Masked Drum Event Discrete	6-19
MDRMW	Masked Drum Event Word	6-21
MLR	Master Line Reset	5-185
MLS	Master Line Set	5-185
MOV	Move	5-144
MOVMC	Move Memory Cartridge	5-145
MRX	Read from MODBUS Network	5-205
MWX	Write to MODBUS	5-208
MUL	Multiply	5-94
MULB	Multiply Binary	5-105
MULBS	Multiply Binary top of stack	5-119
MULD	Multiply Double	5-95
MULF	Multiply Formatted	5-111
MULR	Multiply Real	5-96
MULS	Multiply Top of Stack	5-115
NCON	Numeric Constand	5-199
NEXT	Next (For/Next)	5-180

Instruction		Page
NJMP	Not Jump (Stage)	7-24
NOP	No Operation	5-177
NOT	Not	5-19
OR	Or	5-12, 5-31, 5-75
OR OUT	Or Out	5-19
OR OUTI	Or Out Immediate	5-36
OR STR	Or Store	5-16
ORB	Or Bit-of-Word	5-13
ORD	Or Double	5-76
ORE	Or if Equal	5-28
ORF	Or Formatted	5-77
ORI	Or Immediate	5-34
ORMOV	Or Move	5-171
ORN	Or Not	5-12, 5-31
ORNB	Or Not Bit-of-Word	5-13
ORND	Or Negative Differential	5-22
ORNE	Or if Not Equal	5-28
ORNI	Or Not Immediate	5-34
ORPD	Or Positive Differential	5-22
ORS	Or Stack	5-78
OUT	Out	5-17, 5-65
OUTB	Out Bit-of-Word	5-18
OUTD	Out Double	5-66
OUTF	Out Formatted	5-67
OUTI	Out Immediate	5-36
OUTIF	Out Immediate Formatted	5-37
OUTL	Out Least	5-69
OUTM	Out Most	5-69
OUTX	Out Indexed	5-68
PAUSE	Pause	5-26
PD	Positive Differential	5-20
POP	Pop	5-70
PRINT	Print	5-201
PRINTV	ASCII Print from V-Memory	5-227
RADR	Radian Real Conversion	5-136
RD	Read from Intelligent Module	5-191
RFB	Remove from Bottom of Table	5-157
RFT	Remove from Top of Table	5-163
ROTL	Rotate Left	5-126
ROTR	Rotate Right	5-127
RST	Reset	5-24
RSTB	Reset Bit-of-Word	5-25
RSTBIT	Reset Bit	5-148
RSTI	Reset Immediate	5-38
RSTWT	Reset Watch Dog Timer	5-178

Chapter 5: Standard RLL Instructions

Instruction		Page
RT	Subroutine Return	5-182
RTC	Subroutine Return Conditional	5-182
RTOB	Real to Binary	5-135
RX	Read from Network	5-193
SBR	Subroutine (Goto Subroutine)	5-182
SEG	Segment	5-140
SET	Set	5-24
SETB	Set Bit-of-Word	5-25
SETBIT	Set Bit	5-148
SETI	Set Immediate	5-38
SFLDGT	Shuffle Digits	5-142
SG	Stage	7-23
SGCNT	Stage Counter	5-48
SHFL	Shift Left	5-124
SHFR	Shift Right	5-125
SINR	Sine Real	5-121
SQRTR	Square Root Real	5-122
SR	Shift Register	5-52
STOP	Stop	5-177
STR	Store	5-10, 5-30
STRB	Store Bit-of-Word	5-11
STRE	Store if Equal	5-27
STRI	Store Immediate	5-33
STRN	Store Not	5-10, 5-30
STRNB	Store Not Bit-of-Word	5-11
STRND	Store Negative Differential	5-21
STRNE	Store if Not Equal	5-27
STRNI	Store Not Immediate	5-33
STRPD	Store Positive Differential	5-21
STT	Source to Table	5-160

Instruction		Page
SUB	Subtract	5-91
SUBB	Subtract Binary	5-103
SUBBD	Subtract Binary Double	5-104
SUBBS	Subtract Binary Top of Stack	5-118
SUBD	Subtract Double	5-92
SUBF	Subtract Formatted	5-110
SUBS	Subtract Top of Stack	5-114
SUBR	Subtract Real Number	5-93
SUM	Sum	5-123
SWAP	Swap Table Data	5-174
SWAPB	ASCII Swap Bytes	5-228
TANR	Tangent Real	5-121
TIME	Time	5-176
TMR	Timer	5-42
TMRF	Fast Timer	5-42
TMRA	Accumulating Timer	5-44
TMRAF	Fast Accumulating Timer	5-44
TSHFL	Table Shift Left	5-169
TSHFR	Table Shift Right	5-169
TTD	Table to Destination	5-154
UDC	Up Down Counter	5-50
VPRINT	ASCII Print to V-Memory	5-222
WT	Write to Intelligent Module	5-192
WX	Write to Network	5-195
XOR	Exclusive Or	5-79
XORD	Exclusive Or Double	5-80
XORF	Exclusive Or Formatted	5-81
XORMOV	Exclusive Or Move	5-171
XORS	Exclusive Or Stack	5-82

Using Boolean Instructions

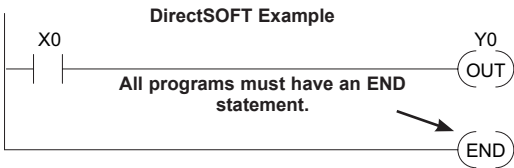
Do you ever wonder why so many PLC manufacturers always quote the scan time for a 1K boolean program? Simple, most programs utilize many boolean instructions. These are typically very simple instructions designed to join input and output contacts in various series and parallel combinations. Our DirectSOFT 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 DirectSOFT, 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 DirectSOFT 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 DirectSOFT and the HPP.

DS	Implied	The following paragraphs describe how these instructions are used to build simple ladder programs.
HPP	Used	

END Statement

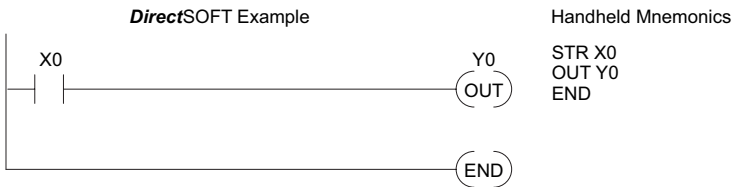
All DL205 programs require an END statement as the last instruction. This tells the CPU that this is the end of the program. Normally, any instructions placed after the END statement will not be executed. There are exceptions to this such as interrupt routines, etc. Chapter 5 discusses the instruction set in detail.



Simple Rungs

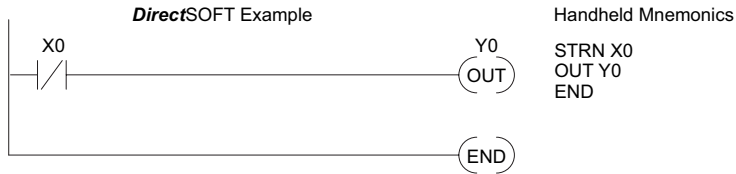
You use a contact to start rungs that contain both contacts and coils. The boolean instruction that does this is called a Store or, STR instruction. The output point is represented by the Output or, OUT instruction.

The following example shows how to enter a single contact and a single output coil.



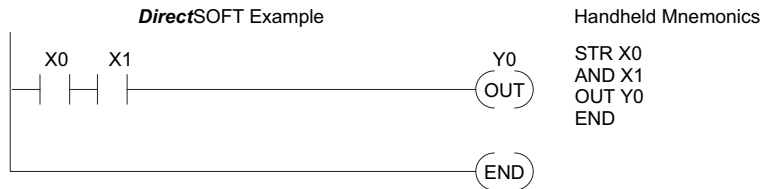
Normally Closed Contact

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



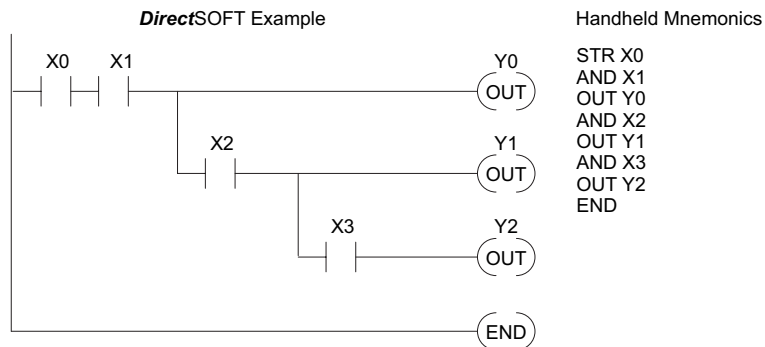
Contacts in Series

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



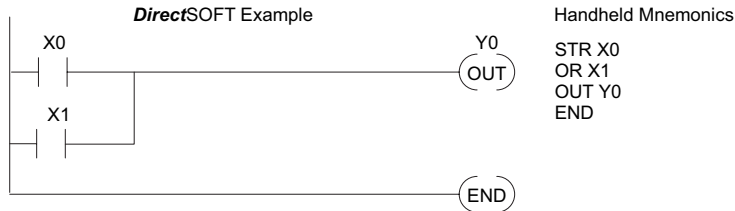
Midline Outputs

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



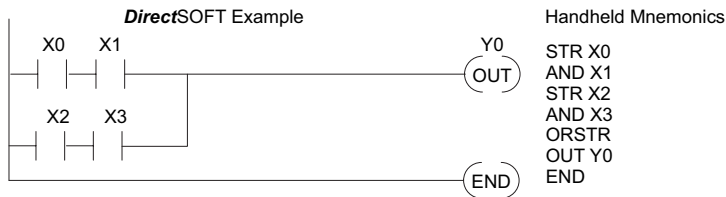
Parallel Elements

You 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.



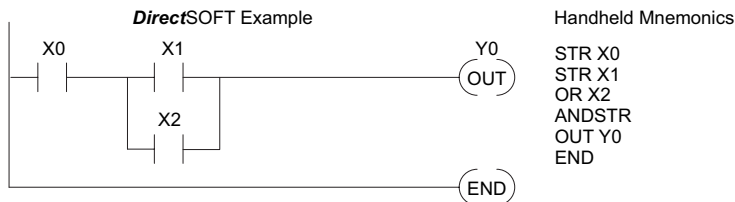
Joining Series Branches in Parallel

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.



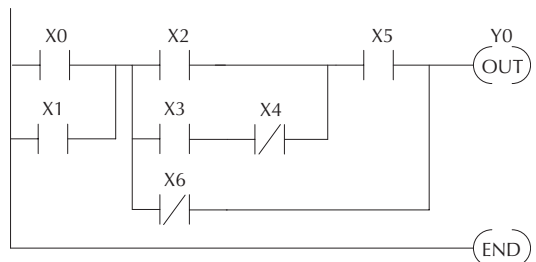
Joining Parallel Branches in Series

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

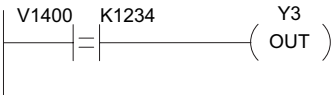
You can combine the various types of series and parallel branches to solve most any application problem. The following example shows a simple combination network.



Comparative Boolean

The DL205 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.

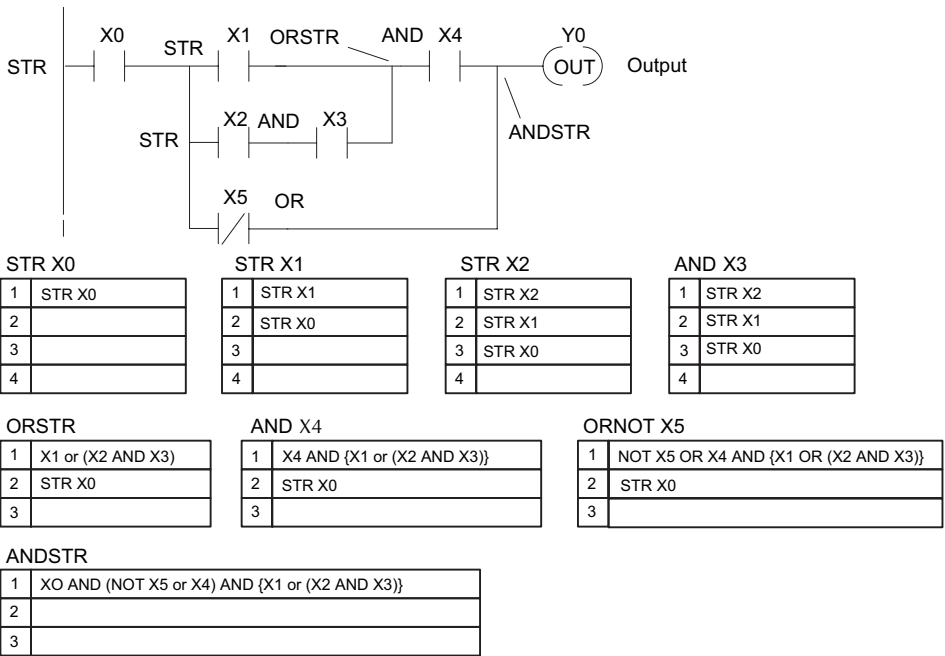
In the example ,when the BCD value in V-memory location V1400 is equal to the constant value 1234, Y3 will energize.



Boolean Stack

There are limits to how many elements you can include in a rung. This is because the DL205 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.

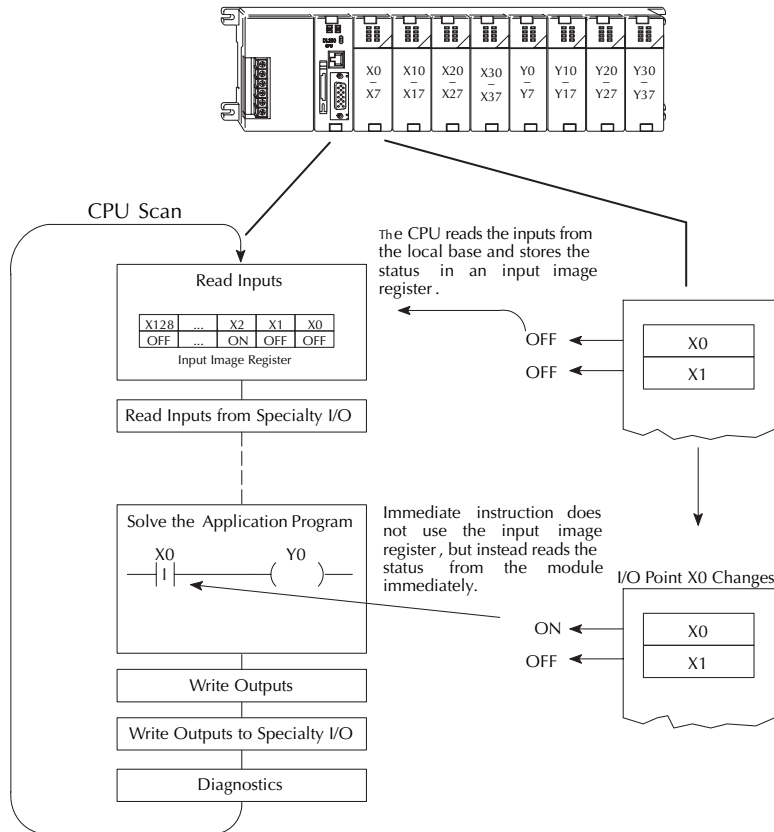
The following example shows how the boolean stack is used to solve boolean logic.



Immediate Boolean

The DL205 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 DL205 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.

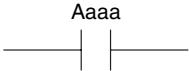
NOTE: Even though the immediate input instruction reads the most current status from the input point, it only uses the results to solve that one instruction. It does not use the new status to update the image register. Therefore, any regular instructions that follow will still use the image register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

- ☒ 230
- ☒ 240
- ☒ 250-1
- 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)

- ☒ 260
- ☒ 262
- 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			
		D2-230	D2-240	D2-250-1	D2-260/D2-262
	A	aaa	aaa	aaa	aaa
Inputs	X	0 – 177	0 – 477	0 – 777	0 – 1777
Outputs	Y	0 – 177	0 – 477	0 – 777	0 – 1777
Control Relays	C	0 – 377	0 – 377	0 – 1777	0 – 3777
Stage	S	0 – 377	0 – 777	0 – 1777	0 – 1777
Timer	T	0 – 77	0 – 177	0 – 377	0 – 377
Counter	CT	0 – 77	0 – 177	0 – 177	0 – 377
Special Relay	SP	0 – 117, 540 – 577	0 – 137 540 – 617	0 – 777	0 – 777
Global	GX	–	–	–	0 – 3777
Global	GY	–	–	–	0 – 3777

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

DS	Used
HPP	Used

DirectSOFT



Handheld Programmer Keystrokes

\$ STR	→	B 1	ENT
GX OUT	→	C 2	ENT

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

DirectSOFT



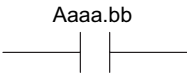
Handheld Programmer Keystrokes

SP STRN	→	B 1	ENT
GX OUT	→	C 2	ENT

Store Bit-of-Word (STRB)

- 230
- 240
- 250-1
- 260
- 262

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



Store Not Bit-of-Word (STRNB)

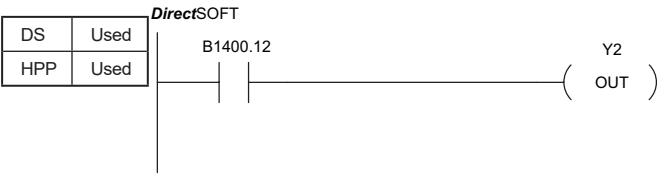
- 230
- 240
- 250-1
- 260
- 262

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

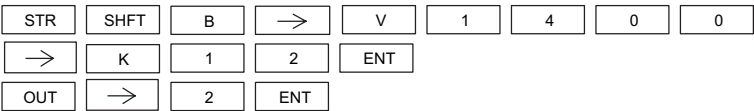


Operand Data Type		D2-250-1 Range		D2-260/D2-262 Range	
	A	aaa	bb	aaa	bb
V-memory	B	See memory map page 3-56	BCD, 0 to 15	See memory map page 3-57	BCD, 0 to 15
Pointer	PB	See memory map page 3-56	BCD	See memory map page 3-57	BCD

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



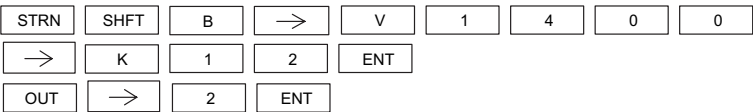
Handheld Programmer Keystrokes



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

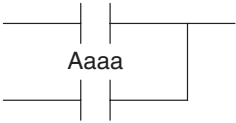


Handheld Programmer Keystrokes



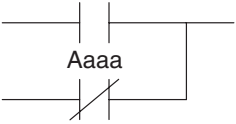
Or (OR)

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.



- ✓ 230
- ✓ 240
- ✓ 250-1
- ✓ 260
- ✓ 262

Operand Data Type		Range			
		D2-230	D2-240	D2-250-1	D2-260/ D2-262
	A	aaa	aaa	aaa	aaa
Inputs	X	0-177	0-477	0-777	0-1777
Outputs	Y	0-177	0-477	0-777	0-1777
Control Relays	C	0-377	0-377	0-1777	0-3777
Stage	S	0-377	0-777	0-1777	0-1777
Timer	T	0-77	0-177	0-377	0-377
Counter	CT	0-77	0-177	0-177	0-377
Special Relay	SP	0-117, 540-577	0-137, 540-617	0-137, 540-717	0-137, 540-717
Global	GX	-	-	-	0-3777
Global	GY	-	-	-	0-3777

DS	Implied
HPP	Used

DirectSOFT



Handheld Programmer Keystrokes

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

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

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

DirectSOFT



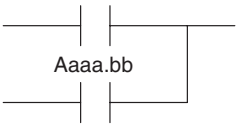
Handheld Programmer Keystrokes

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

Or Bit-of-Word (ORB)

- 230
- 240
- 250-1
- 260
- 262

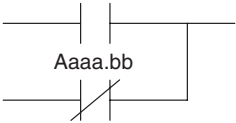
The Or Bit-of-Word instruction logically ors a normally open Bit-of-Word contact in parallel with another contact in a rung. Status of the contact will be the same state as the bit referenced in the associated memory location.



Or Not Bit-of-Word (ORNB)

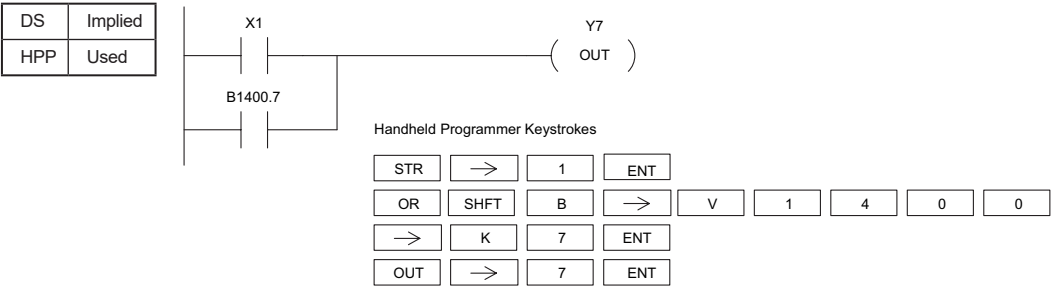
- 230
- 240
- 250-1
- 260
- 262

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

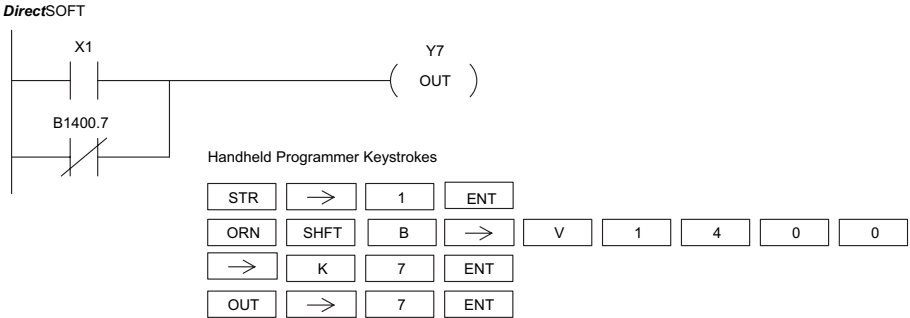


Operand	Data Type	D2-250-1 Range		D2-260/D2-262 Range	
	A	aaa	bb	aaa	bb
V-memory	B	See memory map page 3-56	BCD, 0 to 15	See memory map page 3-57	BCD, 0 to 15
Pointer	PB	See memory map page 3-56	BCD	See memory map page 3-57	BCD

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

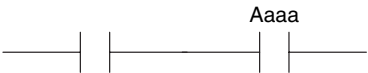


In the following Or Not Bit-of-Word example, when input X1 is on or bit 7 of V1400 is off, output Y7 will energize.



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.



- 230
- 240
- 250-1
- 260
- 262

Operand Data Type		D2-230	D2-240	D2-250-1	D2-260/D2-262
		Range			
	A	aaa	aaa	aaa	aaa
Inputs	X	0–177	0–477	0–777	0–1777
Outputs	Y	0–177	0–477	0–777	0–1777
Control Relays	C	0–377	0–377	0–1777	0–3777
Stage	S	0–377	0–777	0–1777	0–1777
Timer	T	0–77	0–177	0–377	0–377
Counter	CT	0–77	0–177	0–177	0–377
Special Relay	SP	0-117, 540-577	0-137, 540-617	0-137, 540-717	0-137, 540-717
Global	GX	-	-	-	0-3777
Global	GY	-	-	-	0-3777

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

DS	Implied
HPP	Used

DirectSOFT

Handheld Programmer Keystrokes

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

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

DirectSOFT

Handheld Programmer Keystrokes

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

AND Bit-of-Word (ANDB)

230
240

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

250-1
260
262

And Not Bit-of-Word (ANDNB)

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

Operand Data Type		D2-250-1 Range		D2-260/D2-262 Range	
	A	aaa	bb	aaa	bb
V-memory	B	See memory map page 3-56	BCD, 0 to 15	See memory map page 3-57	BCD, 0 to 15
Pointer	PB	See memory map page 3-56	BCD	See memory map page 3-57	BCD

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

DS	Implied
HPP	Used

DirectSOFT

Handheld Programmer Keystrokes

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

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

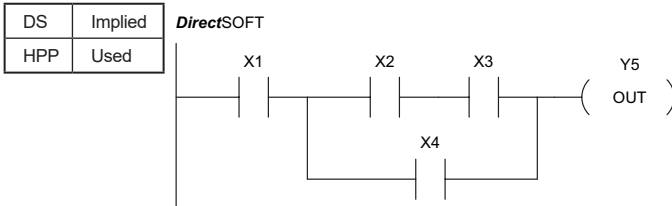
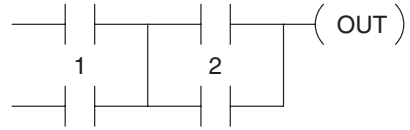
DirectSOFT

Handheld Programmer Keystrokes

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

And Store (ANDSTR)

- ✓ 230 The And Store instruction logically ands two
 - ✓ 240 branches of a rung in series. Both branches must
 - ✓ 250-1 begin with the Store instruction.
 - ✓ 260 In the following And Store example, the branch
 - ✓ 262 consisting of contacts X2, X3, and X4 have been
- anded with the branch consisting of contact X1.

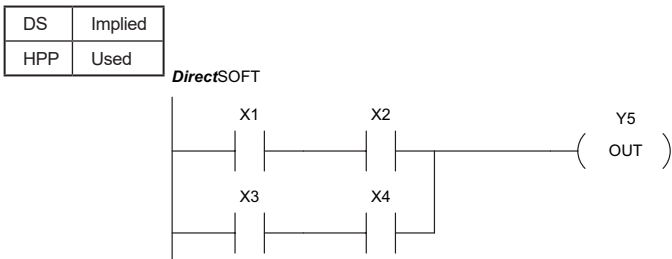
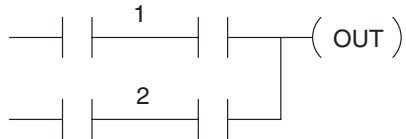


Handheld Programmer Keystrokes

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

Or Store (ORSTR)

- ✓ 230 The Or Store instruction logically ors two branches
 - ✓ 240 of a rung in parallel. Both branches must begin
 - ✓ 250-1 with the Store instruction.
 - ✓ 260 In the following Or Store example, the branch
 - ✓ 262 consisting of X1 and X2 have been OR'd with the
- branch consisting of X3 and X4.



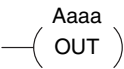
Handheld Programmer Keystrokes

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

Out (OUT)

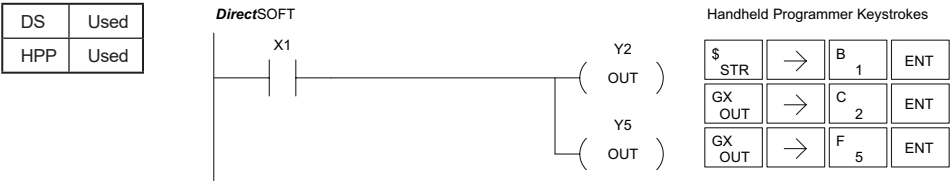
- 230
- 240
- 250-1
- 260
- 262

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.

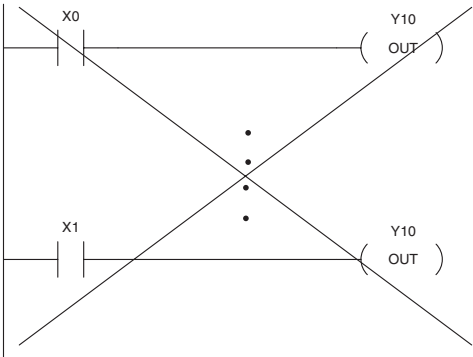


Operand Data Type	Range			
	D2-230	D2-240	D2-250-1	D2-260/D2-262
A	aaa	aaa	aaa	aaa
Inputs	X	0-177	0-477	0-777
Outputs	Y	0-177	0-477	0-1777
Control Relays	C	0-377	0-377	0-3777
Global	GX	-	-	0-3777
Global	GY	-	-	0-3777

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



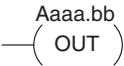
In the following Out example, the program contains two Out instructions using the same location (Y10). The physical output of Y10 is ultimately controlled by the last rung of logic referencing Y10. X1 will override the Y10 output being controlled by X0. To avoid this situation, multiple outputs using the same location should not be used in programming. If you need to have an output controlled by multiple inputs, see the OROUT instruction on page 5-19.



Out Bit-of-Word (OUTB)

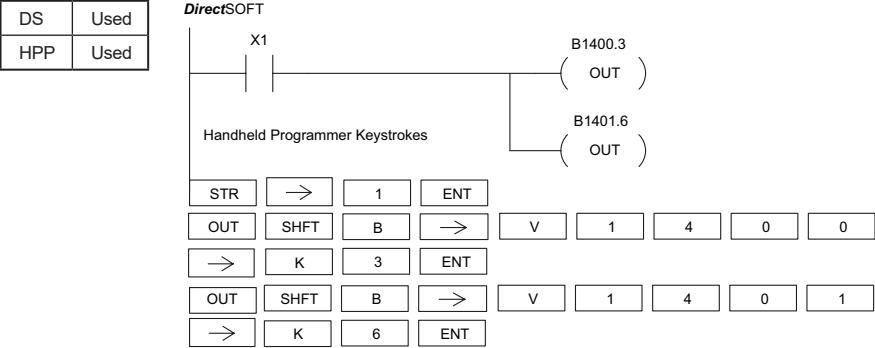
- 230
- 240
- 250-1
- 260
- 262

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

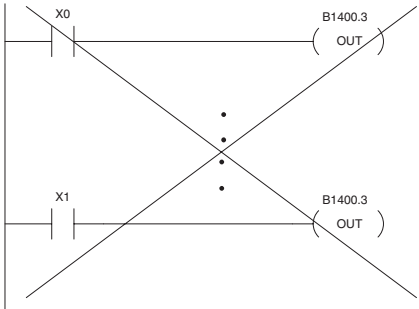


Operand Data Type		D2-250-1 Range		D2-260/D2-262 Range	
	A	aaa	bb	aaa	bb
V-memory	B	See memory map page 3-56	BCD, 0 to 15	See memory map page 3-57	BCD, 0 to 15
Pointer	PB	See memory map page 3-56	BCD	See memory map page 3-57	BCD

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

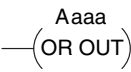


The following Out Bit-of-Word example contains two Out Bit-of-Word instructions using the same bit in the same memory word. The final state bit 3 of V1400 is ultimately controlled by the last rung of logic referencing it. X1 will override the logic state controlled by X0. To avoid this situation, multiple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



Or Out (OROUT)

- ☒ 230
- ☒ 240
- ☒ 250-1
- ☒ 260
- ☒ 262
- 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.

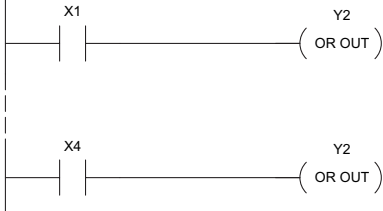


Operand Data Type		Range			
		D2-230	D2-240	D2-250-1	D2-260/D2-262
	A	aaa	aaa	aaa	aaa
Inputs	X	0-177	0-477	0-777	0-1777
Outputs	Y	0-177	0-477	0-777	0-1777
Control Relays	C	0-377	0-377	0-1777	0-3777
Global	GX	-	-	-	0-3777
Global	GY	-	-	-	0-3777

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

DS	Used
HPP	Used

DirectSOFT

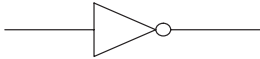


Handheld Programmer Keystrokes

\$	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

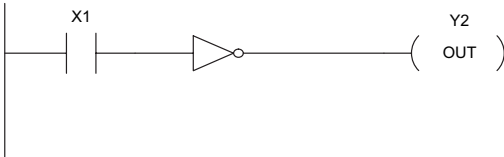
Not (NOT)

- ☒ 230
- ☒ 240
- ☒ 250-1
- ☒ 260
- ☒ 262
- The Not instruction inverts the status of the rung at the point of the instruction.
- In the following example, when X1 is off, Y2 will energize. This is because the Not instruction inverts the status of the rung at the Not instruction.



DS	Used
HPP	Used

DirectSOFT



Handheld Programmer Keystrokes

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

✓ 230 Positive Differential (PD)

✓ 240 The Positive Differential instruction is typically known as
✓ 250-1 a one shot. When the input logic produces an off-to-on
✓ 260 transition, the output will energize for one CPU scan.
✓ 262

(A aaa
PD)

Operand Data Type	Range			
	D2-230	D2-240	D2-250-1	D2-260/D2-262
A	aaa	aaa	aaa	aaa
Inputs X	0-177	0-477	0-777	0-1777
Outputs Y	0-177	0-477	0-777	0-1777
Control Relays C	0-377	0-377	0-1777	0-3777

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



NOTE: To generate a “one-shot” pulse on an on-to-off transition, place a NOT instruction immediately before the PD instruction. The D2-250-1, D2-260 and D2-262 CPUs support the STRND instruction.

DS	Used
HPP	Used

DirectSOFT



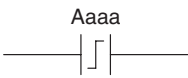
Handheld Programmer Keystrokes

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

Store Positive Differential (STRPD)

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

- 230
- 240
- 250-1
- 260
- 262



Store Negative Differential (STRND)

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

- 230
- 240
- 250-1
- 260
- 262



Operand	Data Type	D2-250-1 Range	D2-260/D2-262 Range
	A	aaa	aaa
Inputs	X	0-777	0-1777
Outputs	Y	0-777	0-1777
Control Relays	C	0-1777	0-3777
Stage	S	0-1777	0-1777
Timer	T	0-377	0-377
Counter	CT	0-177	0-377
Global	GX	-	0-3777
Global	GY	-	0-3777

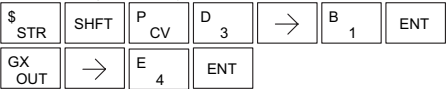
DS	Used
HPP	Used

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

DirectSOFT



Handheld Programmer Keystrokes

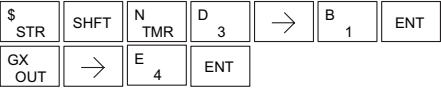


In the following example, each time X1 makes an on-to-off transition, Y4 will energize for one scan.

DirectSOFT

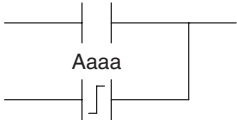


Handheld Programmer Keystrokes



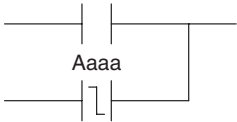
Or Positive Differential (ORPD)

- ☒ 230
- ☒ 240
- ☒ 250-1
- ☒ 260
- ☒ 262
- The Or Positive Differential instruction logically ORs a contact in parallel with another contact in a rung. The status of the contact will be open until the associated image register point makes an off-to-on transition, closing it for one CPU scan. Thereafter, it remains open until another off-to-on transition.



Or Negative Differential (ORND)

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



Operand Data Type		D2-250-1 Range	D2-260/D2-262 Range
	A	aaa	aaa
Inputs	X	0-777	0-1777
Outputs	Y	0-777	0-1777
Control Relays	C	0-1777	0-3777
Stage	S	0-1777	0-1777
Timer	T	0-377	0-377
Counter	CT	0-177	0-377
Global	GX	-	0-3777
Global	GY	-	0-3777

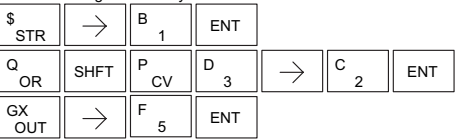
DS	Implied
HPP	Used

In the following example, Y5 will energize whenever X1 is on, or for one CPU scan when X2 transitions from off to on.

DirectSOFT



Handheld Programmer Keystrokes

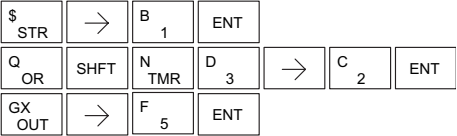


In the following example, Y5 will energize whenever X1 is on, or for one CPU scan when X2 transitions from on to off.

DirectSOFT



Handheld Programmer Keystrokes



And Positive Differential (ANDPD)

- 230
- 240
- 250-1
- 260
- 262

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



And Negative Differential (ANDND)

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

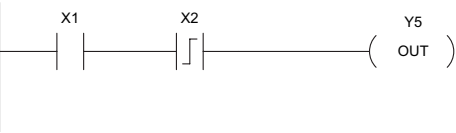


Operand Data Type		D2-250-1 Range	D2-260/D2-262 Range
	A	aaa	aaa
Inputs	X	0-777	0-1777
Outputs	Y	0-777	0-1777
Control Relays	C	0-1777	0-3777
Stage	S	0-1777	0-1777
Timer	T	0-377	0-377
Counter	CT	0-177	0-377
Global	GX	-	0-3777
Global	GY	-	0-3777

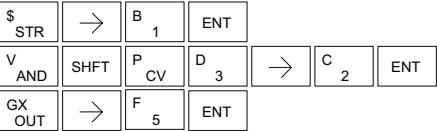
DS	Implied
HPP	Used

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

DirectSOFT

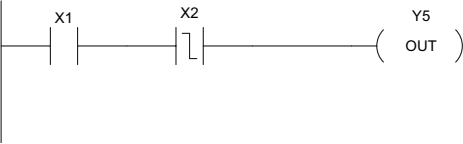


Handheld Programmer Keystrokes

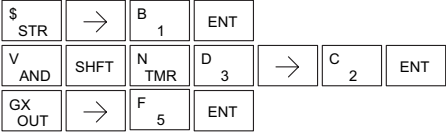


In the following example, Y5 will energize for one CPU scan whenever X1 is on and X2 transitions from on to off.

DirectSOFT

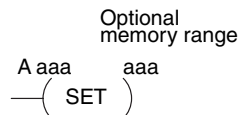


Handheld Programmer Keystrokes



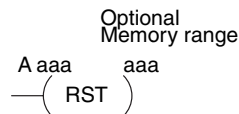
Set (SET)

- ✓ 230 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.
- ✓ 240
- ✓ 250-1
- ✓ 260
- ✓ 262



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		D2-230	D2-240	D2-250-1	D2-260/D2-262
		Range			
	A	aaa	aaa	aaa	aaa
Inputs	X	0-177	0-477	0-777	0-1777
Outputs	Y	0-177	0-477	0-777	0-1777
Control Relays	C	0-377	0-377	0-1777	0-3777
Stage	S	0-377	0-777	0-1777	0-1777
Timer	T	0-77	0-177	0-377	0-377
Counter*	CT	0-77	0-177	0-177	0-377
Global	GX	-	-	-	0-3777
Global	GY	-	-	-	0-3777

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



NOTE: You cannot set inputs (Xs) that are assigned to input modules

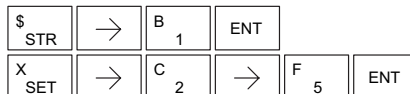
In the following example, when X1 is on, Y2 through Y5 will energize.

DS	Used
HPP	Used

DirectSOFT



Handheld Programmer Keystrokes

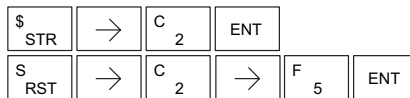


In the following example, when X2 is on, Y2 through Y5 will be reset or de-energized.

DirectSOFT



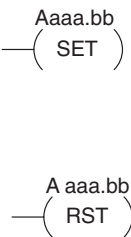
Handheld Programmer Keystrokes



Set Bit-of-Word (SETB)

- 230
- 240
- 250-1
- 260
- 262

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



Reset Bit-of-Word (RSTB)

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

Operand Data Type		D2-250-1 Range		D2-260/D2-262 Range	
	A	aaa	bb	aaa	bb
V-memory	B	See memory map page 3-56	BCD, 0 to 15	See memory map page 3-57	BCD, 0 to 15
Pointer	PB	See memory map page 3-56	BCD	See memory map page 3-57	BCD

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

DS	Used
HPP	Used

DirectSOFT



Handheld Programmer Keystrokes

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

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

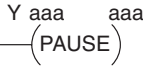
DirectSOFT



Handheld Programmer Keystrokes

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

- ✓ 230 **Pause (PAUSE)**
- ✓ 240 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
- ✓ 250-1 outputs in the range specified in the Pause instruction will be turned off at the output points.
- ✓ 260
- ✓ 262

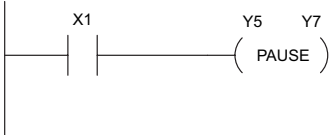


Operand Data Type	Range			
	D2-230	D2-240	D2-250-1	D2-260/D2-262
	aaa	aaa	aaa	aaa
Outputs Y	0-177	0-477	0-777	0-1777

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

DirectSOFT



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

Handheld Programmer Keystrokes

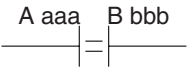
\$ STR	→	B 1	ENT																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																									
-----------	---	--------	-----	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

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

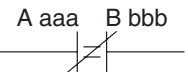
Store If Equal (STRE)

- ☒ 230
- ☒ 240
- ☒ 250-1
- 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)

- ☒ 260
- ☒ 262
- 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							
		D2-230		D2-240		D2-250-1		D2-260/D2-262	
A/B		aaa	bbb	aaa	bbb	aaa	bbb	aaa	bbb
V-memory	V	All. (See memory map page 3-54)	All. (See memory map page 3-54)	All. (See memory map page 3-55)	All. (See memory map page 3-55)	All. (See memory map page 3-55)	All. (See memory map page 3-56)	All. (See memory map page 3-57)	All. (See memory map page 3-57)
Pointer	P	-	-	-	All. (See memory map page 3-55)	-	All. (See memory map page 3-56)	-	All. (See memory map page 3-57)
Constant	K	-	0-FFFF	-	0-FFFF	-	0-FFFF	-	0-FFFF

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

DS

Implied

HPP

Used

DirectSOFT

V2000

K4933

Y3

OUT

Handheld Programmer Keystrokes

\$

STR

SHFT

E

4

→

C

2

A

0

A

0

A

0

→

E

4

J

9

D

3

D

3

ENT

GX

OUT

→

D

3

ENT

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

DirectSOFT

V2000

K5060

Y3

OUT

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)

- ✓

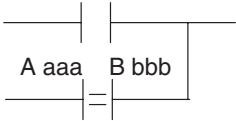
230
- ✓

240
- ✓

250-1
- ✓

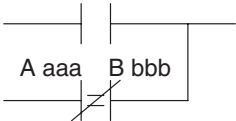
260
- ✓

262
- 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.

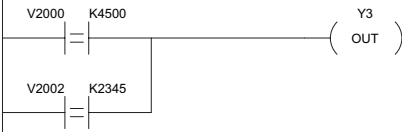


Operand Data Type		Range							
		D2-230		D2-240		D2-250-1		D2-260/D2-262	
A/B		aaa	bbb	aaa	bbb	aaa	bbb	aaa	bbb
V-memory	V	All. (See memory map page 3-54)	All. (See memory map page 3-54)	All. (See memory map page 3-55)	All. (See memory map page 3-55)	All. (See memory map page 3-56)	All. (See memory map page 3-56)	All. (See memory map page 3-57)	All. (See memory map page 3-57)
Pointer	P	—	—	—	All. (See memory map page 3-55)	—	All. (See memory map page 3-56)	—	All. (See memory map page 3-57)
Constant	K	—	0-FFFF	—	0-FFFF	—	0-FFFF	—	0-FFFF

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

DS	Implied
HPP	Used

DirectSOFT

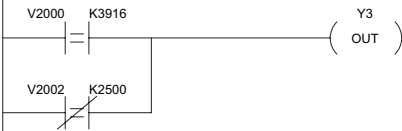


Handheld Programmer Keystrokes

\$ 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.

DirectSOFT

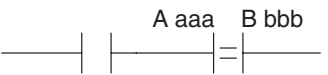


Handheld Programmer Keystrokes

\$ 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)

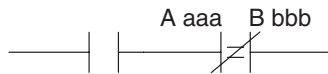
The And If Equal instruction connects a normally open comparative contact in series with another contact. The contact will be on when Aaaa equals Bbbb.



- 230
- 240
- 250-1
- 260
- 262

And If Not Equal (ANDNE)

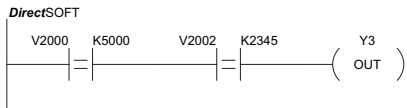
The And If Not Equal instruction connects a normally closed comparative contact in series with another contact. The contact will be on when Aaaa does not equal Bbbb



Operand Data Type		Range							
A/B		D2-230		D2-240		D2-250-1		D2-260/D2-262	
		aaa	bbb	aaa	bbb	aaa	bbb	aaa	bbb
V-memory	V	All. (See memory map page 3-54)	All. (See memory map page 3-54)	All. (See memory map page 3-55)	All. (See memory map page 3-55)	All. (See memory map page 3-56)	All. (See memory map page 3-56)	All. (See memory map page 3-57)	All. (See memory map page 3-57)
Pointer	P	—	—	—	All. (See memory map page 3-55)	—	All. (See memory map page 3-56)	—	All. (See memory map page 3-57)
Constant	K	—	0-FFFF	—	0-FFFF	—	0-FFFF	—	0-FFFF

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

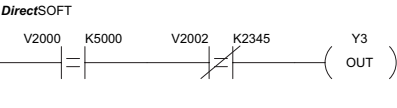
DS	Implied
HPP	Used



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.

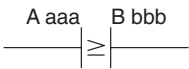


Handheld Programmer Keystrokes

\$	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					

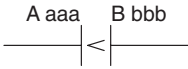
Store (STR)

- ☒ 230
- ☒ 240
- ☒ 250-1
- ☒ 260
- ☒ 262
- 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							
		D2-230		D2-240		D2-250-1		D2-260/D2-262	
A/B		aaa	bbb	aaa	bbb	aaa	bbb	aaa	bbb
V-memory	V	All. (See memory map page 3-54)	All. (See memory map page 3-54)	All. (See memory map page 3-55)	All. (See memory map page 3-55)	All. (See memory map page 3-56)	All. (See memory map page 3-56)	All. (See memory map page 3-57)	All. (See memory map page 3-57)
Pointer	P	–	–	–	All. (See memory map page 3-55)	–	All. (See memory map page 3-56)	–	All. (See memory map page 3-57)
Constant	K	–	0-FFFF	–	0-FFFF	–	0-FFFF	–	0-FFFF

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

DS	Implied
HPP	Used

DirectSOFT



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				

energize.

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

DirectSOFT



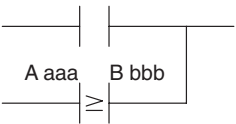
Handheld Programmer Keystrokes

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				

energize.

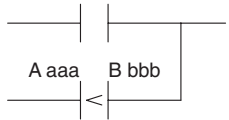
Or (OR)

- ☒ 230
- ☒ 240
- ☒ 250-1
- ☒ 260
- ☒ 262
- 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.

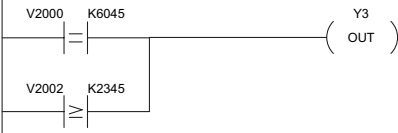


Operand Data Type	Range							
	D2-230		D2-240		D2-250-1		D2-260/D2-262	
A/B	aaa	bbb	aaa	bbb	aaa	bbb	aaa	bbb
V-memory V	All. (See memory map page 3-54)	All. (See memory map page 3-54)	All. (See memory map page 3-55)	All. (See memory map page 3-55)	All. (See memory map page 3-56)	All. (See memory map page 3-56)	All. (See memory map page 3-57)	All. (See memory map page 3-57)
Pointer P	—	—	—	All. (See memory map page 3-55)	—	All. (See memory map page 3-56)	—	All. (See memory map page 3-57)
Constant K	—	0-FFFF	—	0-FFFF	—	0-FFFF	—	0-FFFF

In the following example, when the value in V-memory location V2000 = 6045 or

DS	Implied
HPP	Used

DirectSOFT



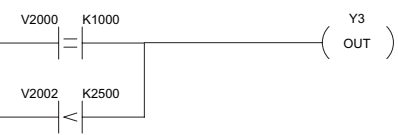
Handheld Programmer Keystrokes

\$	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					

V2002 M 2345, Y3 will energize.

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

DirectSOFT

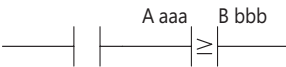


Handheld Programmer Keystrokes

\$	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					

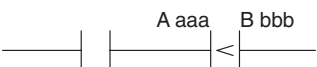
And (AND)

- ☒ 230
- ☒ 240
- ☒ 250-1
- ☒ 260
- ☒ 262
- 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)

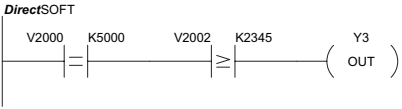
The Comparative And Not instruction connects a normally open comparative contact in series with another contact. The contact will be on when Aaaa < Bbbb.



Operand Data Type	Range							
	D2-230		D2-240		D2-250-1		D2-260/D2-262	
A/B	aaa	bbb	aaa	bbb	aaa	bbb	aaa	bbb
V-memory V	All. (See memory map page 3-54)	All. (See memory map page 3-54)	All. (See memory map page 3-55)	All. (See memory map page 3-55)	All. (See memory map page 3-56)	All. (See memory map page 3-56)	All. (See memory map page 3-57)	All. (See memory map page 3-57)
Pointer P	–	–	–	All. (See memory map page 3-55)	–	All. (See memory map page 3-56)	–	All. (See memory map page 3-57)
Constant K	–	0-FFFF	–	0-FFFF	–	0-FFFF	–	0-FFFF

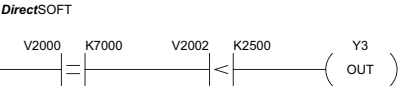
In the following example, when the value in V-memory location V2000 = 5000, and V2002 M 2345, Y3 will energize.

DS	Implied
HPP	Used



\$ 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.



\$ 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)

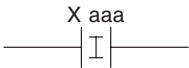
- ✓ 230

✓ 240

✓ 250-1

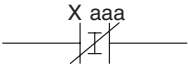
✓ 260

✓ 262
- 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			
	D2-230	D2-240	D2-250-1	D2-260/D2-262
	aaa	aaa	aaa	aaa
Inputs	X 0-177	0-477	0-777	0-1777

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

DS	Implied
HPP	Used

DirectSOFT

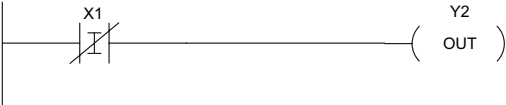


Handheld Programmer Keystrokes

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

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

DirectSOFT

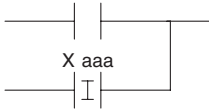


Handheld Programmer Keystrokes

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

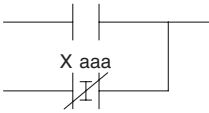
Or Immediate (ORI)

- 230 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.
- 240
- 250-1
- 260



Or Not Immediate (ORNI)

- 262 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.

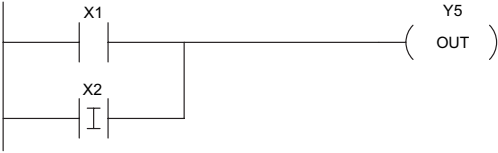


Operand Data Type	Range			
	D2-230	D2-240	D2-250-1	D2-260/D2-262
	aaa	aaa	aaa	aaa
Inputs	X	0-177	0-477	0-777

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

DS	Implied
HPP	Used

DirectSOFT

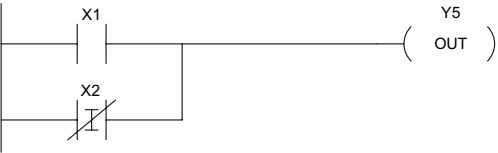


Handheld Programmer Keystrokes

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

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

DirectSOFT



Handheld Programmer Keystrokes

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

And Immediate (ANDI)

- 230 The And Immediate connects two contacts in series. The
- 240 status of the contact will be the same as the status of
- 250-1 the associated input point at the time the instruction is
- 260 executed. The image register is not updated.



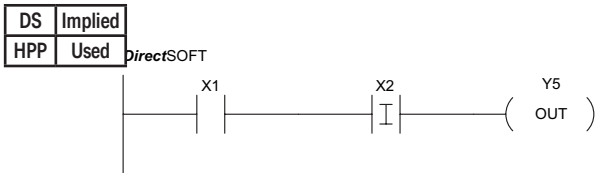
And Not Immediate (ANDNI)

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



Operand Data Type	Range			
	D2-230	D2-240	D2-250-1	D2-260/D2-262
	aaa	aaa	aaa	aaa
Inputs X	0-177	0-477	0-777	0-1777

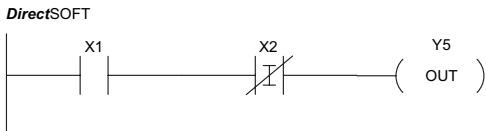
In the following example, when X1 and X2 are on, Y5 will energize.



Handheld Programmer Keystrokes

\$ STR	→	B 1	ENT
V AND	SHFT	I 8	→ C 2 ENT
GX OUT	→	F 5	ENT

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



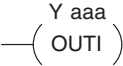
Handheld Programmer Keystrokes

\$ STR	→	B 1	ENT
W ANDN	SHFT	I 8	→ C 2 ENT
GX OUT	→	F 5	ENT

Out Immediate (OUTI)

- 230
- 240
- 250-1
- 260
- 262

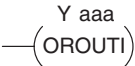
The Out Immediate instruction reflects the status of the rung (on/off) and outputs the discrete (on/off) status to the specified module output point and the image register at the time the instruction is executed. If multiple Out Immediate instructions referencing the same discrete point are used, it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



Or Out Immediate (OROUTI)

- 230
- 240
- 250-1
- 260
- 262

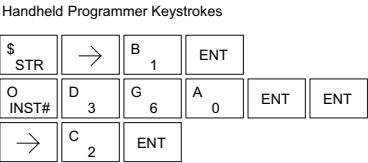
The Or Out Immediate instruction has been designed to use more than one rung of discrete logic to control a single output. Multiple Or Out Immediate instructions referencing the same output coil may be used, since all contacts controlling the output are ORed together. If the status of any rung is on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



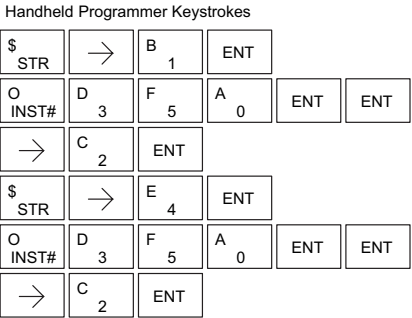
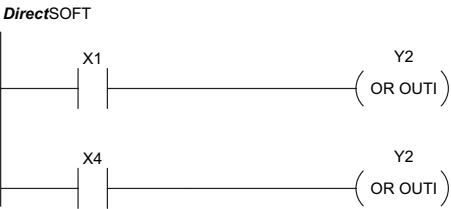
Operand Data Type	Range			
	D2-230	D2-240	D2-250-1	D2-260/D2-262
	aaa	aaa	aaa	aaa
Outputs	Y	0-177	0-777	0-1777

In the following example, when X1 is on, output point Y2 on the output module will turn on. For instruction entry on the Handheld Programmer, you can use the instruction number (#350) as shown, or type each letter of the command.

DS	Used
HPP	Used



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



Out Immediate Formatted (OUTIF)

- 230
- 240
- 250-1
- 260
- 262

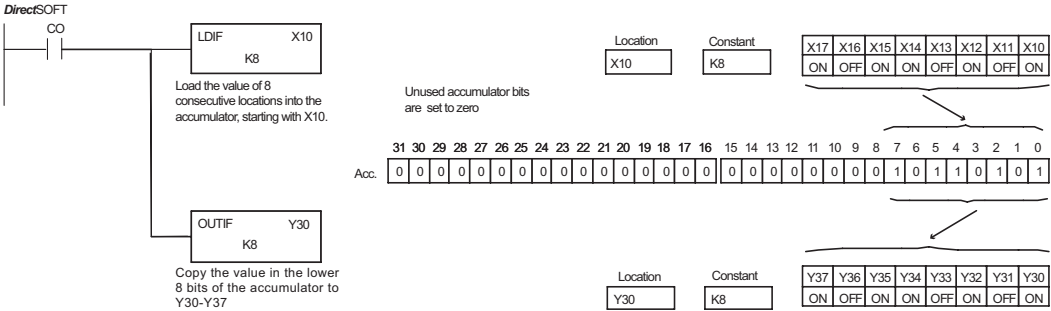
The Out Immediate Formatted instruction outputs a 1 to 32 bit binary value from the accumulator to specified output points at the time the instruction is executed. Accumulator bits that are not used by the instruction are set to zero.



Operand Data Type		D2-260/D2-262 Range	
		aaa	bbb
Outputs	Y	0-1777	-
Constant	K	-	1-32

In the following example, when C0 is on,the binary pattern for X10 –X17 is loaded into the accumulator using the Load Immediate Formatted instruction. The binary pattern in the accumulator is written to Y30–Y37 using the Out Immediate Formatted instruction. This technique is useful to quickly copy an input pattern to outputs (without waiting for the CPU scan).

DS	Used
HPP	Used



Handheld Programmer Keystrokes

\$ STR	→	NEXT	NEXT	NEXT	NEXT	A ₀	ENT				
SHFT	L ANDST	D ₃	I ₈	F ₅	→	B ₁	A ₀	→	I ₈	ENT	
GX OUT	SHFT	I ₈	F ₅	→	D ₃	A ₀	→	I ₈	ENT		

Set Immediate (SETI)

- ✓

230
- ✓

240
- ✓

250-1
- ✓

260
- ✓

262
- 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.

$$\overline{Y} \begin{matrix} \text{aaa} \\ \text{---} \end{matrix} \left(\text{SETI} \right) \begin{matrix} \text{aaa} \\ \text{---} \end{matrix}$$

Reset Immediate (RSTI)

- ✓

230
- ✓

240
- ✓

250-1
- ✓

260
- ✓

262
- 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.

$$\overline{Y} \begin{matrix} \text{aaa} \\ \text{---} \end{matrix} \left(\text{RSTI} \right) \begin{matrix} \text{aaa} \\ \text{---} \end{matrix}$$

Operand Data Type		Range			
		D2-230	D2-240	D2-250-1	D2-260/D2-262
		aaa	aaa	aaa	aaa
Outputs	Y	0-177	0-477	0-777	0-1777

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.

DS	Used
HPP	Used

DirectSOFT

Handheld Programmer Keystrokes

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

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

DirectSOFT

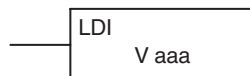
Handheld Programmer Keystrokes

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

Load Immediate (LDI)

- ☐ 230
- ☐ 240
- ☐ 250-1
- ☒ 260
- ☒ 262

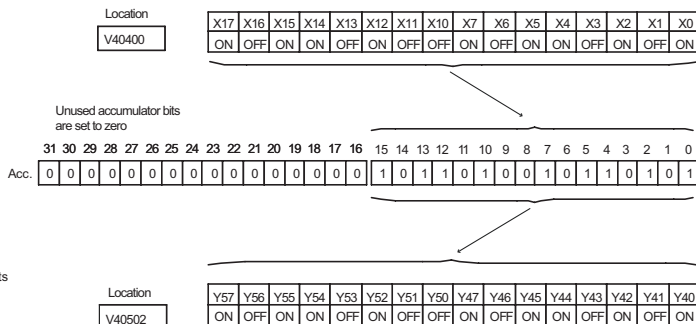
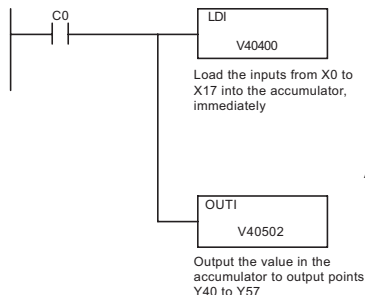
The Load Immediate instruction loads a 16-bit V-memory value into the accumulator. The valid address range includes all input point addresses on the local base. The value reflects the current status of the input points at the time the instruction is executed. This instruction may be used instead of the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



Operand Data Type	D2-260/D2-262 Range
	aaaaa
Inputs V-memory V	40400-40477

In the following example, when C0 is on, the binary pattern of X0–X17 will be loaded into the accumulator using the Load Immediate instruction. The Out Immediate instruction could be used to copy the 16 bits in the accumulator to output points, such as Y40–Y57. This technique is useful to quickly copy an input pattern to output points (without waiting for a full CPU scan to occur).

DS	Used
HPP	Used



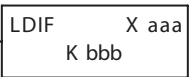
Handheld Programmer Keystrokes

[illegible]

Load Immediate Formatted (LDIF)

- 230
- 240
- 250-1
- 260
- 262

The Load Immediate Formatted instruction loads a 1–32 bit binary value into the accumulator. The value reflects the current status of the input module(s) at the time the instruction is executed. Accumulator bits that are not used by the instruction are set to zero.

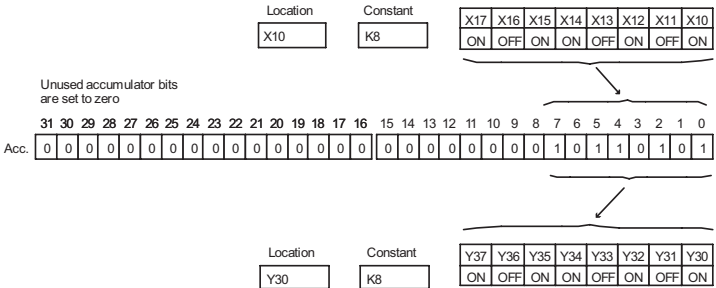
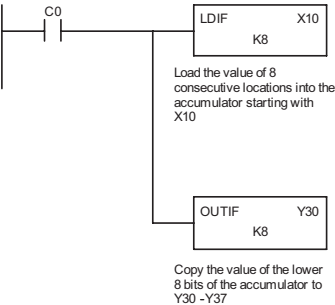


Operand Data Type		D2-260/D2-262 Range	
		aaa	bbb
Inputs	X	0–1777	–
Constant	K	–	1–32

In the following example, when C0 is on, the binary pattern of X10–X17 will be loaded into the accumulator using the Load Immediate Formatted instruction. The Out Immediate Formatted instruction could be used to copy the specified number of bits in the accumulator to the specified outputs on the output module, such as Y30–Y37. This technique is useful to quickly copy an input pattern to outputs (without waiting for the CPU scan).

DS	Used
HPP	Used

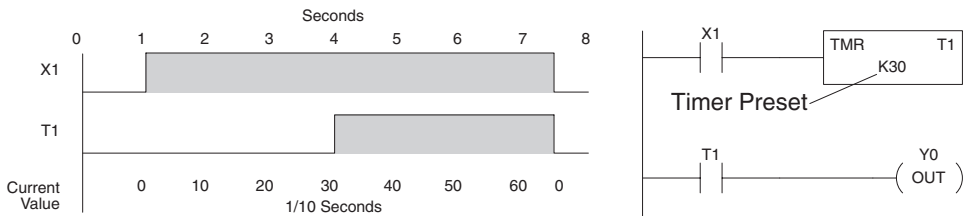
DirectSOFT



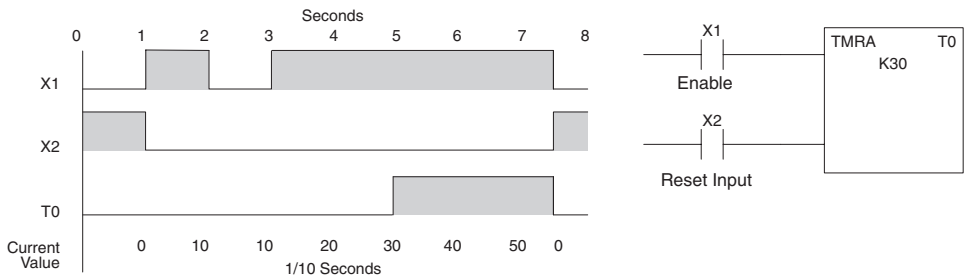
Timer, Counter and Shift Register Instructions

Using Timers

Timers are used to time an event for a desired length of time. The single input timer (TMR) 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 (TMRA) 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)

- ✓ **230** The Timer instruction is a 0.1 second single-input timer that times to a maximum of 999.9 seconds. The Timer Fast
- ✓ **240** instruction is a 0.01 second single input timer that times up to a maximum of 99.99 seconds. These timers will be enabled
- ✓ **250-1** if the input logic is true (on) and will be reset to 0 if the input
- ✓ **260** logic is false (off).

✓ **262** Instruction Specifications

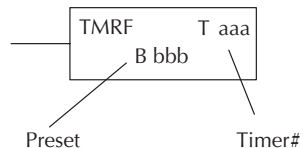
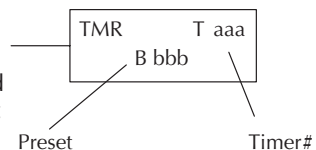
DS	Used
HPP	Used

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: A V-memory preset is required only if the ladder program or an Operator Interface unit must change the preset.

Operand Data Type		Range							
		D2-230		D2-240		D2-250-1		D2-260/D2-262	
B		aaa	bbb	aaa	bbb	aaa	bbb	aaa	bbb
Timers	T	0-77		0-177		0-377		0-377	
V-memory for preset values	V	-	2000-2377	-	2000-3777	-	1400-7377 10000-17777	-	1400-7377 10000-37777
Pointers (presets only)	P				2000-3777		1400-7377 10000-17777		1400-7377 10000-37777
Constants (presets only)	K	-	0-9999	-	0-9999	-	0-9999	-	0-9999
Timer discrete status bits	T/V*	0-77 or V41100-41103		0-177 or V41100-41107		0-377 or V41100-41117		0-377 or V41100-41117	
Timer current values	V/T*	0-77		0-177		0-377		0-377	

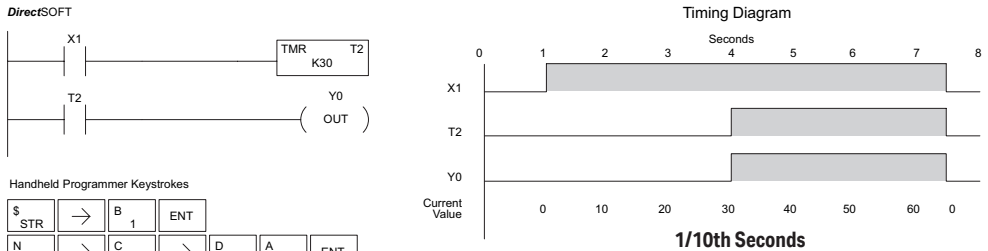


NOTE: *Both the Timer discrete status bits and the current value are accessed with the same data reference with the HPP. DirectSOFT uses separate references, such as "T2" for discrete status bit for Timer T2, and "TA2" for the current value of Timer T2.

You can perform functions when the timer reaches the specified preset using the discrete status bit. Or, use 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.

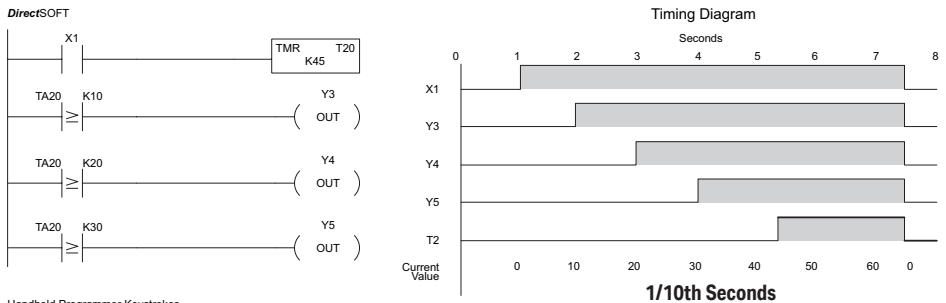
Timer Example Using Discrete Status Bits

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.



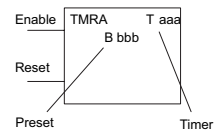
Timer Example Using Comparative Contacts

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



Accumulating Timer (TMRA)

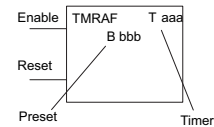
- ✓ 230 The Accumulating Timer is a 0.1 second two-input timer that will time to a maximum of 9999999.9. The TMRA uses two timer registers in V-memory.
- ✓ 240



- ✓ 250-1
 - ✓ 260
 - ✓ 262
- ### 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

DS	Used
HPP	Used

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 TMRA T1 is used, the next available timer number is T3.

NOTE: A V-memory preset is required only if the ladder program or an OIT must be used to change the preset.

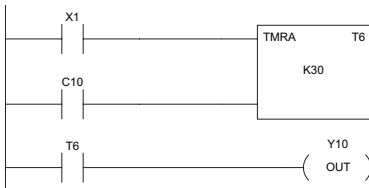
Operand Data Type		Range							
		D2-230		D2-240		D2-250-1		D2-260/D2-262	
B		aaa	bbb	aaa	bbb	aaa	bbb	aaa	bbb
Timers	T	0-77		0-177		0-377		0-377	
V-memory for preset values	V	-	2000-2377	-	2000-3777	-	1400-7377 10000-17777	-	1400-7377 10000-37777
Pointers (presets only)	P				2000-3777		1400-7377 10000-17777		1400-7377 10000-37777
Constants (presets only)	K	-	0-9999	-	0-9999	-	0-9999	-	0-9999
Timer discrete status bits	T/V*	0-77 or V41100-41103		0-177 or V41100-41107		0-377 or V41100-41117		0-377 or V41100-41117	
Timer current values	V/T*	0-77		0-177		0-377		0-377	

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

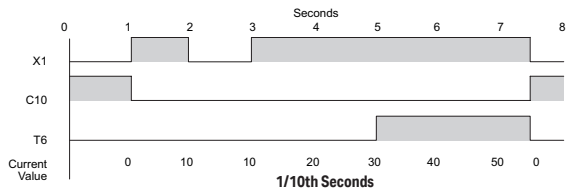
Accumulating Timer Example using Discrete Status Bits

In the following example, a two input timer (accumulating timer) is used with a preset of 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.

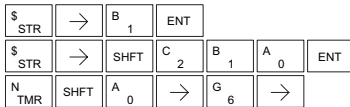
DirectSOFT



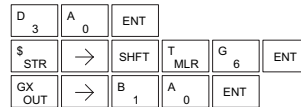
Timing Diagram



Handheld Programmer Keystrokes



Handheld Programmer Keystrokes (cont'd)

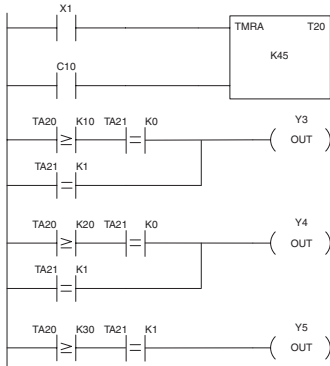


Accumulator Timer Example Using Comparative Contacts

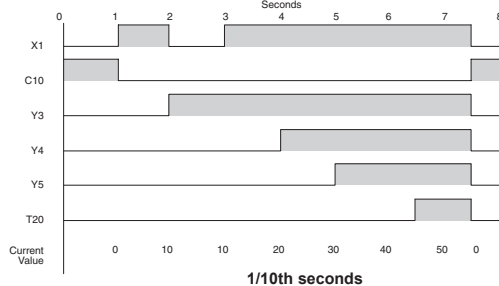
In the following example, a two-input timer is used with a preset of 4.5 seconds. Comparative contacts are used to energize Y3, Y4, and Y5 at one-second intervals respectively. The comparative contacts will turn off when the timer is reset.

Contacts

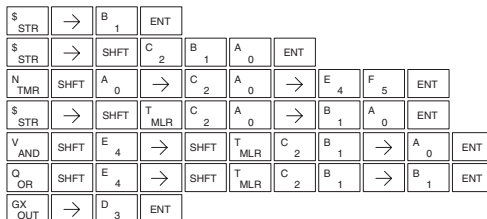
DirectSOFT



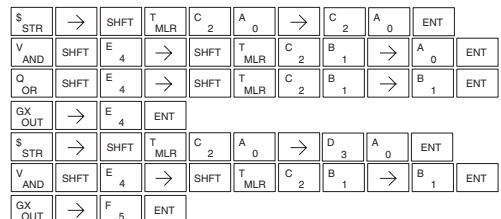
Timing Diagram



Handheld Programmer Keystrokes

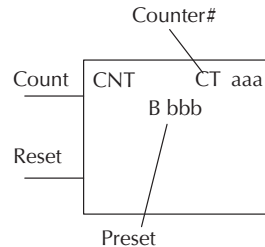


Handheld Programmer Keystrokes (cont'd)



Counter (CNT)

- ✓ 230 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.
- ✓ 240
- ✓ 250-1
- ✓ 260
- ✓ 262



DS	Used
HPP	Used

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: A V-memory preset is required only if the ladder program or an OIT must be used to change the preset.

Operand Data Type	Range							
	D2-230		D2-240		D2-250-1		D2-260/D2-262	
B	aaa	bbb	aaa	bbb	aaa	bbb	aaa	bbb
Counters CT	0-77		0-177		0-377		0-377	
V-memory for preset values V	-	2000-2377	-	2000-3777	-	1400-7377 10000-17777	-	1400-7377 10000-37777
Pointers (presets only) P				2000-3777		1400-7377 10000-17777		1400-7377 10000-37777
Constants (presets only) K	-	0-9999	-	0-9999	-	0-9999	-	0-9999
Counter discrete status bits CT/V*	0-77 or V41100-41103		0-177 or V41100-41107		0-377 or V41100-41117		0-377 or V41100-41117	
Counter current values V/CT*	1000-1077		1000-1177		1000-1177		1000-1377	

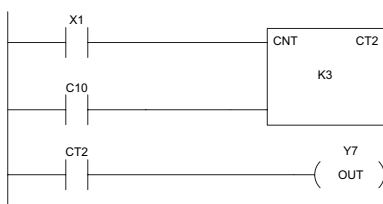


NOTE: * Both the Counter discrete status bits and the current value are accessed with the same data reference with the HPP. DirectSOFT uses separate references, such as "CT2" for discrete status bit for Counter CT2, and "CTA2" for the current value of Counter CT2.

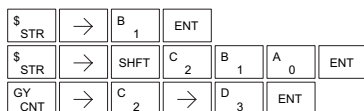
Counter Example Using Discrete Status Bits

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

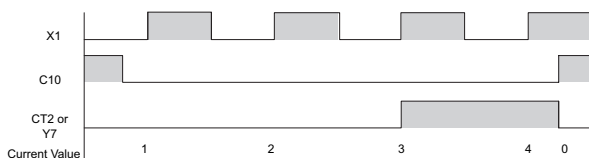
DirectSOFT



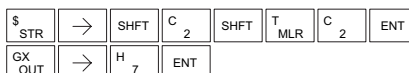
Handheld Programmer Keystrokes



Counting diagram



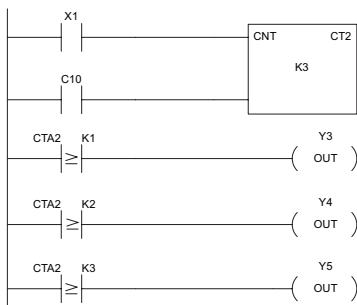
Handheld Programmer Keystrokes (cont)



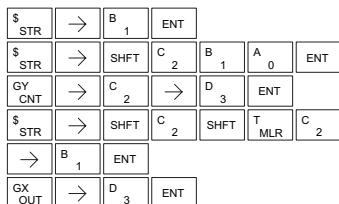
Counter Example Using Comparative Contacts

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

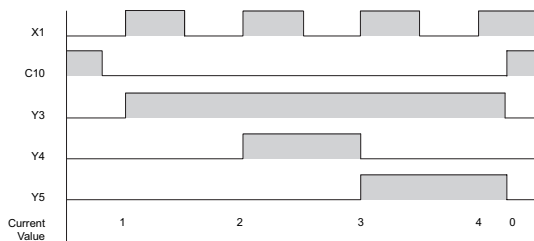
DirectSOFT



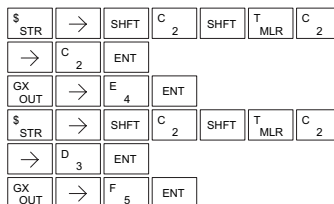
Handheld Programmer Keystrokes



Counting diagram



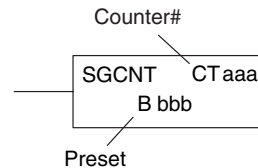
Handheld Programmer Keystrokes (cont)



Stage Counter (SGCNT)

- ✓ 230
- ✓ 240
- ✓ 250-1
- ✓ 260
- ✓ 262

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 RLLPLUS 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

DS	Used
HPP	Used

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: When using a counter inside a stage, the stage must be active for one scan before the input to the counter makes a 0-1 transition. Otherwise, there is no real transition and the counter will not count.



NOTE: A V-memory preset is required only if the ladder program or an OIT must used to change the preset.

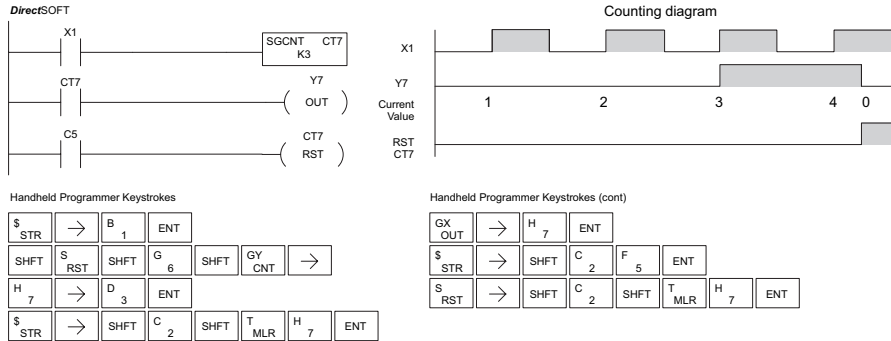
Operand Data Type	Range							
	D2-230		D2-240		D2-250-1		D2-260/D2-262	
B	aaa	bbb	aaa	bbb	aaa	bbb	aaa	bbb
Counters CT	0-77		0-177		0-377		0-377	
V-memory for preset values V	-	2000-2377	-	2000-3777	-	1400-7377 10000-17777	-	1400-7377 10000-37777
Pointers (presets only) P				2000-3777		1400-7377 10000-17777		1400-7377 10000-37777
Constants (presets only) K	-	0-9999	-	0-9999	-	0-9999	-	0-9999
Counter discrete status bits CT/V*	0-77 or V41140-41143		0-177 or V41140-41147		0-377 or V41140-41147		0-377 or V41100-41157	
Counter current values V/CT*	1000-1077		1000-1177		1000-1177		1000-1377	



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

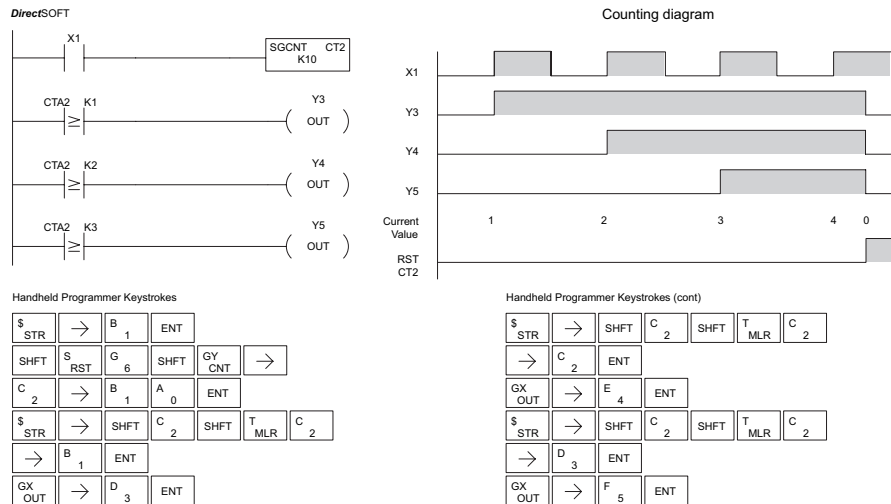
Stage Counter Example Using Discrete Status Bits

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



Stage Counter Example Using Comparative Contacts

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



Up Down Counter (UDC)

- ✓ 230 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.
- ✓ 240
- ✓ 250-1
- ✓ 260
- ✓ 262

DS	Used
HPP	Used

Instruction Specification

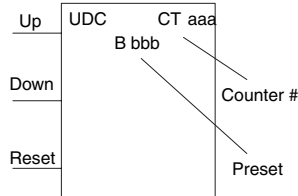
Counter Reference (CTaaa): Specifies the counter number.

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.

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.



NOTE: The UDC uses two consecutive V-memory locations for the 8-digit value, therefore two consecutive counter locations. For example, if UDC CT1 is used, the next available counter number is CT3.



NOTE: A V-memory preset is required only if the ladder program or an OIT must be used to change the preset.

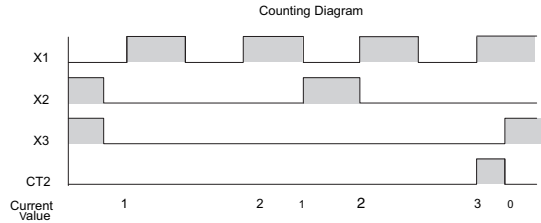
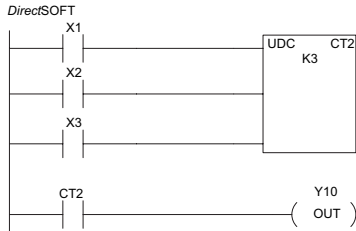
Operand Data Type		Range							
		D2-230		D2-240		D2-250-1		D2-260/D2-262	
	B	aaa	bbb	aaa	bbb	aaa	bbb	aaa	bbb
Counters	CT	0-76		0-176		0-176		0-376	
V-memory for preset values	V	-	2000-2377	-	2000-3777	-	1400-7377 10000-17777	-	1400-7377 10000-37777
Pointers (presets only)	P				2000-3777		1400-7377 10000-17777		1400-7377 10000-37777
Constants (presets only)	K	-	0-99999999	-	0-99999999	-	0-99999999	-	0-99999999
Counter discrete status bits	CT/V*	0-76 or V41140-41143		0-176 or V41140-41147		0-176 or V41140-41147		0-376 or V41100-41157	
Counter current values	V/CT*	1000-1076		1000-1176		1000-1176		1000-1376	



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

Up/Down Counter Example Using Discrete Status Bits

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

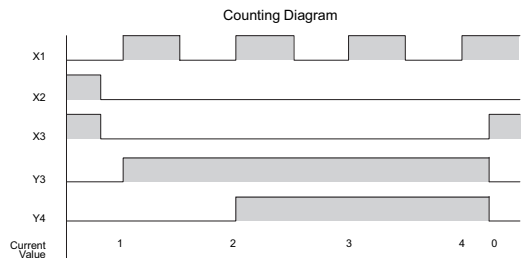
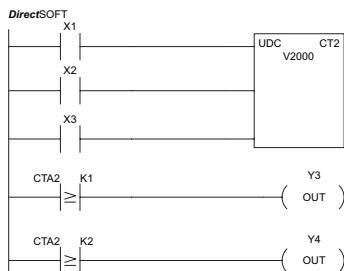
\$	STR	→	B	1	ENT
\$	STR	→	C	2	ENT
\$	STR	→	D	3	ENT
SHFT	U	ISG	D	3	C
				2	→
					C
					2

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																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																									
---	----------------	-----	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

Shift Register (SR)

- ✓ 230

✓ 240

✓ 250-1

✓ 260

✓ 262
- 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.
- DS

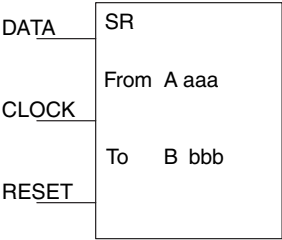
Used

HPP
- Used

Used
- Data — determines the value (1 or 0) that will enter the register

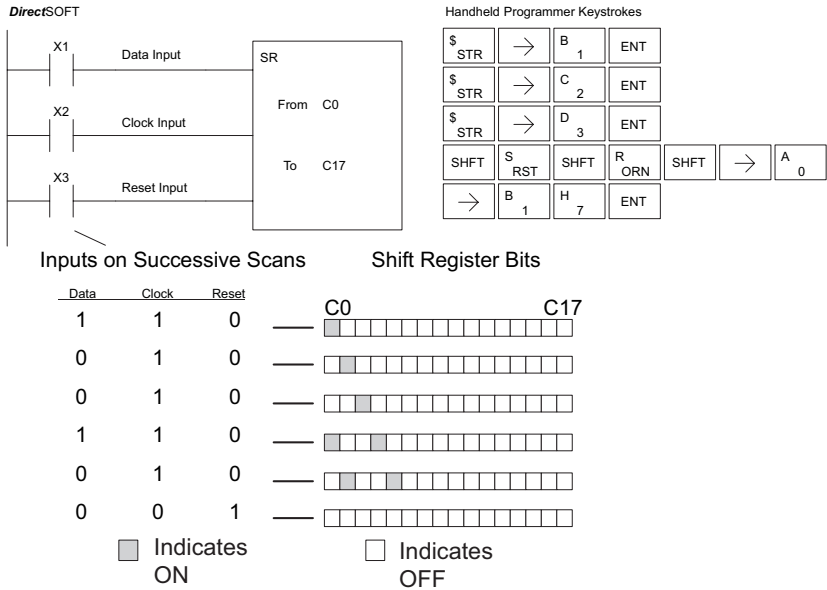
• Clock — shifts the bits one position on each low to high transition

• 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	Range							
	D2-230		D2-240		D2-250-1		D2-260/D2-262	
A/B	aaa	bbb	aaa	bbb	aaa	bbb	aaa	bbb
Control Relay C	0-377	0-377	0-377	0-377	0-1777	0-1777	0-3777	0-3777



Inputs on Successive Scans

Data	Clock	Reset
1	1	0
0	1	0
0	1	0
1	1	0
0	1	0
0	1	0
0	0	1

Shift Register Bits

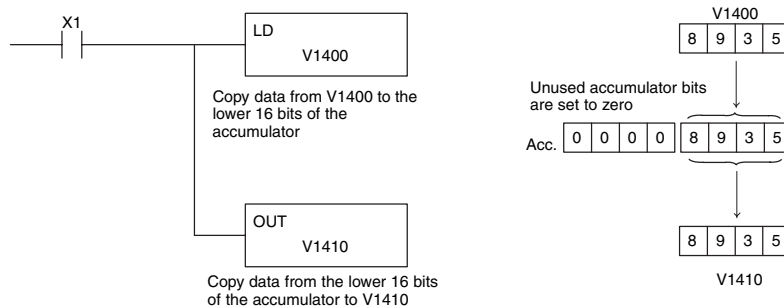
Accumulator/Stack Load and Output Data Instructions

Using the Accumulator

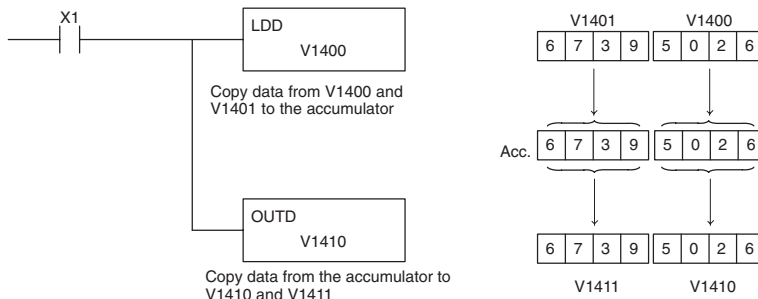
The accumulator in the DL205 series CPUs is a 32-bit register that is used as a temporary storage location for data that is being copied or manipulated in some manner. For example, you have to use the accumulator to perform math operations, such as, add, subtract, multiply, etc. Since there are 32 bits, you can use up to an 8-digit BCD number or a 32-bit 2's complement number. The accumulator is reset to 0 at the end of every CPU scan.

Copying Data to the Accumulator

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

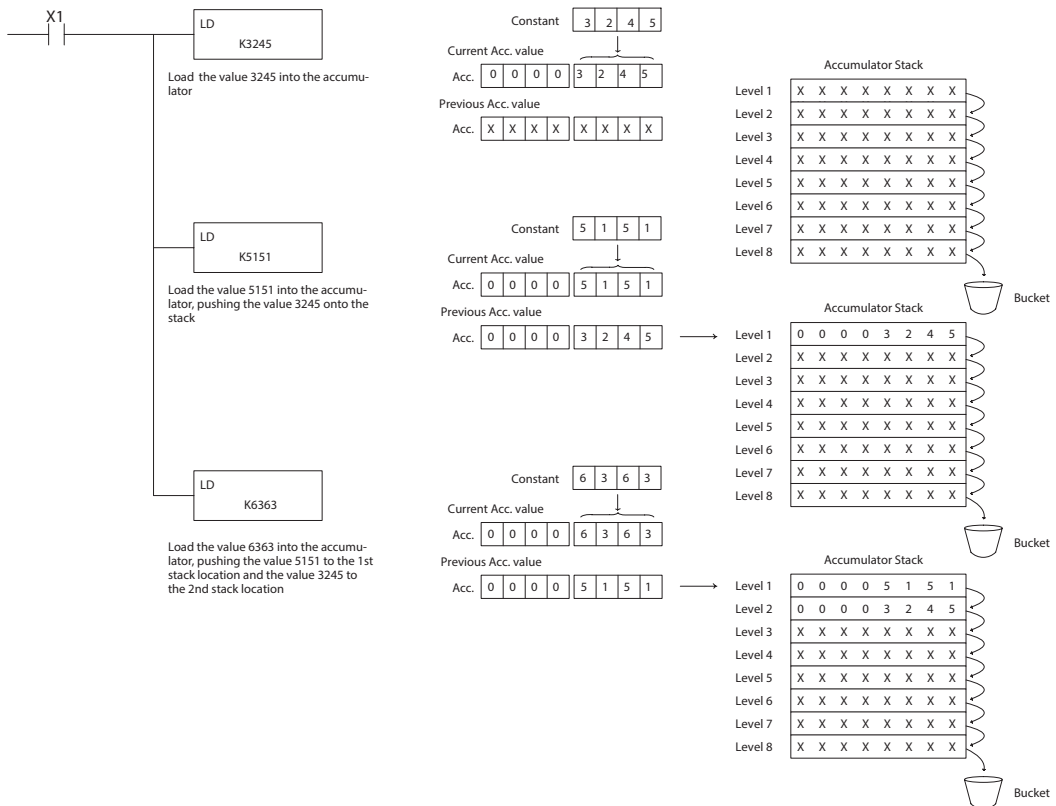


Since the accumulator is 32 bits and V-memory locations are 16 bits, the Load Double and Out Double (or variations thereof) use two consecutive V-memory locations or 8-digit BCD constants to copy data either to the accumulator from a V-memory address or from a V-memory address to the accumulator. For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

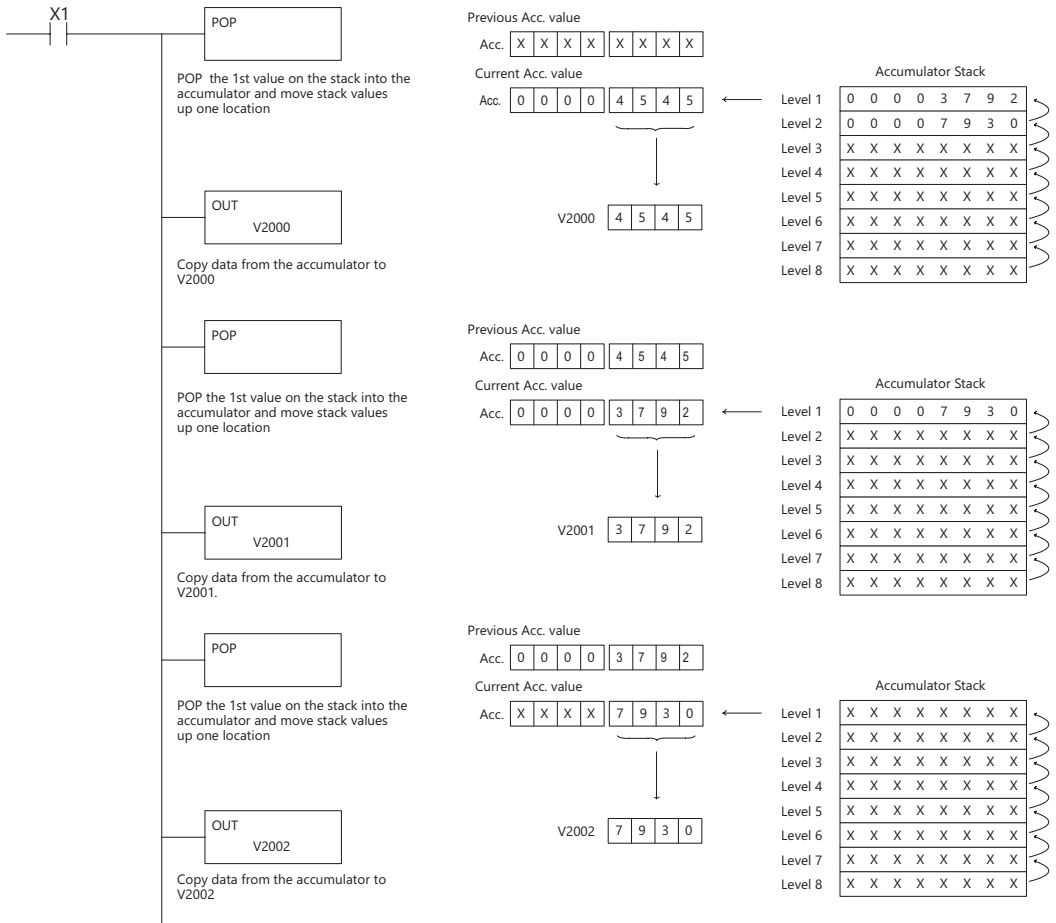


Using the Accumulator Stack

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.

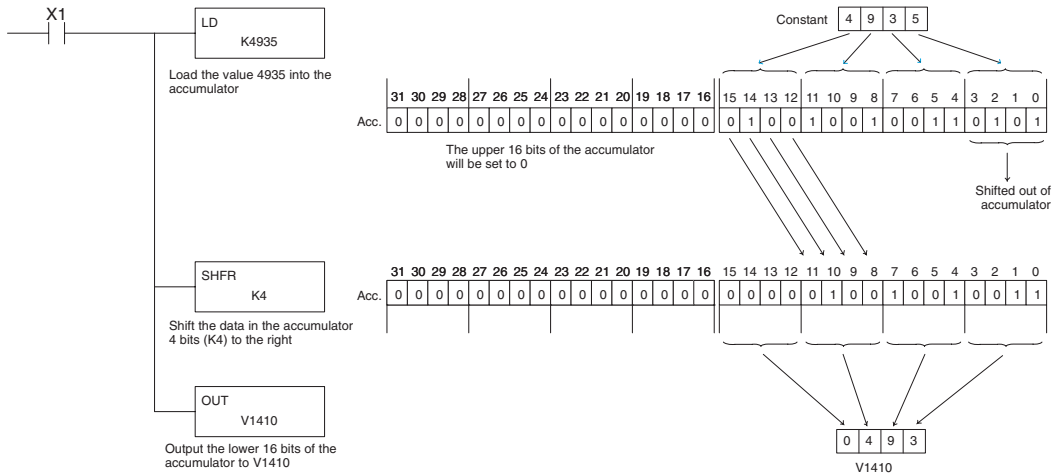


The POP instruction rotates values upward through the stack into the accumulator. When a POP is executed, the value that was in the accumulator is cleared and the value that was on top of the stack is in the accumulator. The values in the stack are shifted up one position in the stack.

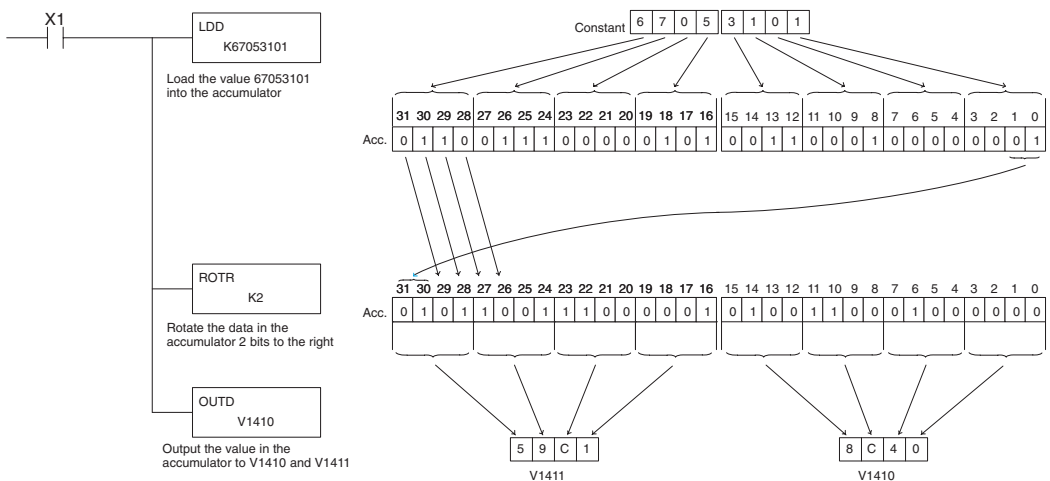


Changing the Accumulator Data

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 V1410.



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. The following example rotates the value 67053101 two bits to the right and outputs the value to V1410 and V1411.



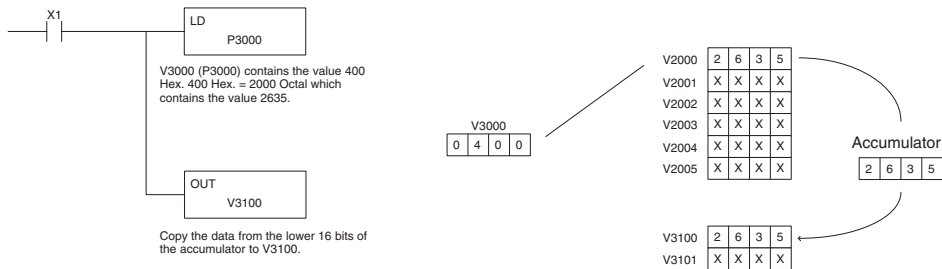
Using Pointers

Many of the DL205 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.

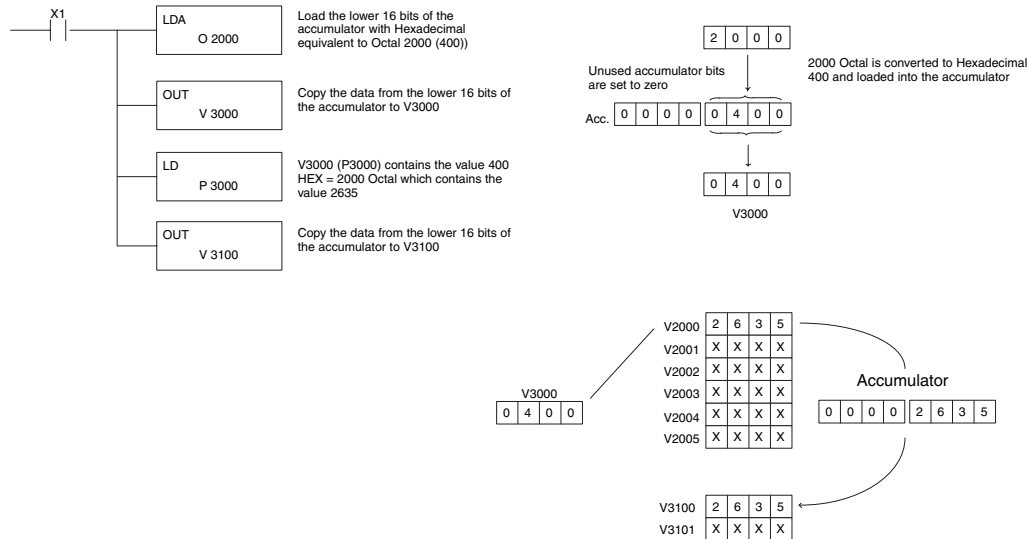


NOTE: In the DL205, V-memory addressing is in octal. However the value in the pointer location which will reference a V-memory location is viewed as HEX. Use the Load Address instruction to move an address into the pointer location. This instruction performs the Octal to Hexadecimal conversion for you.

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

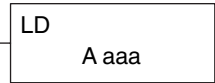


The following example is similar to the one above, except for the LDA (load address) instruction, which automatically converts the Octal address to the Hex equivalent.



230 Load (LD)

- 240 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.
- 250-1
- 260
- 262



Operand Data Type	Range				
	D2-230	D2-240	D2-250-1	D2-260/D2-262	
A	aaa	aaa	aaa	aaa	bbb
V-memory V	All See Memory map		All, see Memory map		
Pointer P	-		All V-memory; see Memory map		
Constant K	0-FFFF		0-FFFF		

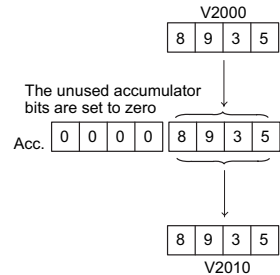
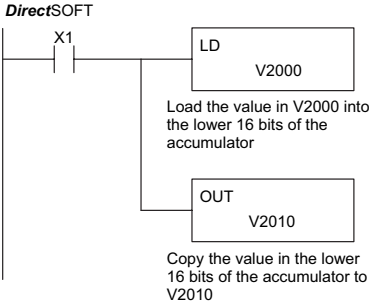
Discrete Bit Flags	Description
SP76	On when the value loaded into the accumulator by any instruction is zero.



NOTE: Two consecutive Load 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.

DS	Used
HPP	Used

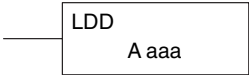


Handheld Programmer Keystrokes

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

230 Load Double (LDD)

240 The Load Double instruction is a 32-bit instruction that loads
250-1 the value (Aaaa), which is either two consecutive V-memory
260 locations or an 8-digit constant value, into the accumulator.
262



Operand Data Type	Range				
	D2-230	D2-240	D2-250-1	D2-260/D2-262	
A	aaa	aaa	aaa	aaa	bbb
V-memory	V	All See Memory map			
Pointer	P	All V-memory; see Memory map			
Constant	K	0-FFFFFFF			

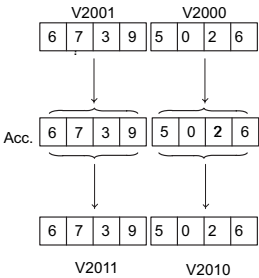
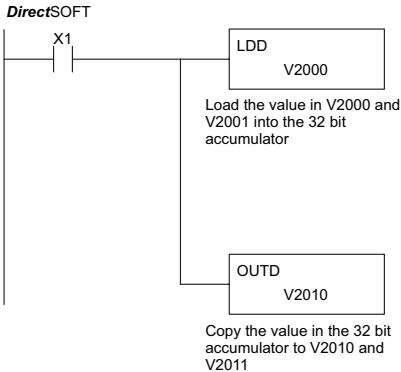
Discrete Bit Flags	Description
SP76	On when the value loaded into the accumulator by any instruction is zero.



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.

DS	Used
HPP	Used



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)

- 230 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.
- 240
- 250-1
- 260
- 262

LDF A aaa
 K bbb

Operand Data Type		Range					
		D2-240		D2-250-1		D2-260/D2-262	
A		aaa	bbb	aaa	bbb	aaa	bbb
Inputs X	X	0-477	-	0-777	-	0-1777	-
Outputs Y	Y	0-477	-	0-777	-	0-1777	-
Control Relays C	C	0-377	-	0-1777	-	0-3777	-
Stage bits S	S	0-777	-	0-1777	-	0-1777	-
Timer bits T	T	0-177	-	0-377	-	0-377	-
Counter bits CT	CT	0-177	-	0-177	-	0-377	-
Special Relays SP	SP	0-137, 540-617	-	0-777	-	0-777	-
Global I/O GX/GY	GX/GY	-	-	-	-	0-3777	-
Constant K	K	-	1-32	-	1-32	-	1-32

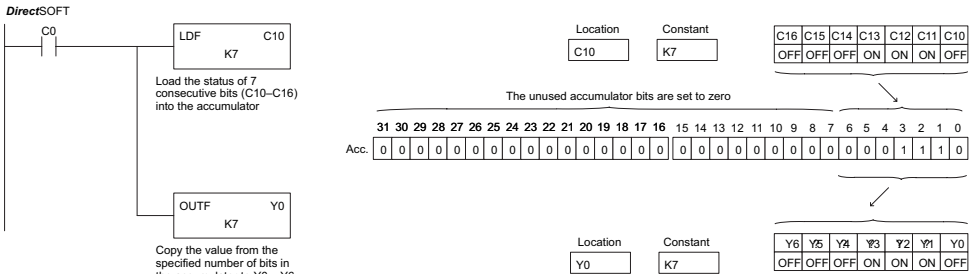
Discrete Bit Flags	Description
SP76	ON when the value loaded into the accumulator by any instruction is zero.



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

DS	Used
HPP	Used

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.



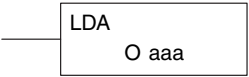
Handheld Programmer Keystrokes

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

Load Address (LDA)

- 230
- 240
- 250-1
- 260
- 262

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	Range			
	D2-230	D2-240	D2-250-1	D2-260/D2-262
	aaa	aaa	aaa	aaa
Octal Address	0	All V-memory; see Memory map		

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

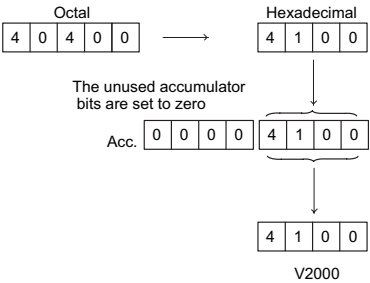
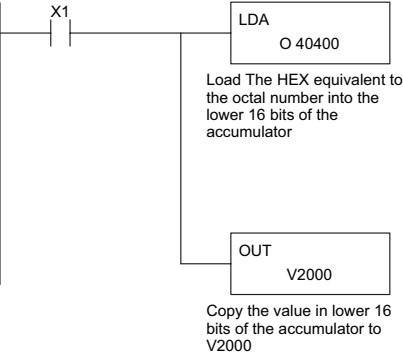


NOTE: Two consecutive Load instructions will place the value of the first Load instruction onto the accumulator stack.

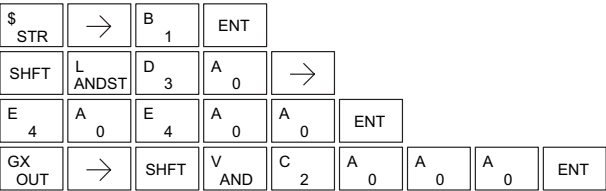
In the following example, when X1 is on, the octal number 40400 will be converted to a HEX 4100 and loaded into the accumulator using the Load Address instruction. The value in the lower 16 bits of the accumulator is copied to V2000 using the Out instruction.

DS	Used
HPP	Used

DirectSOFT



Handheld Programmer Keystrokes



Load Accumulator Indexed (LDX)

- 230
- 240
- 250-1
- 260
- 262

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

LDX
A aaa

DS	Used
HPP	Used

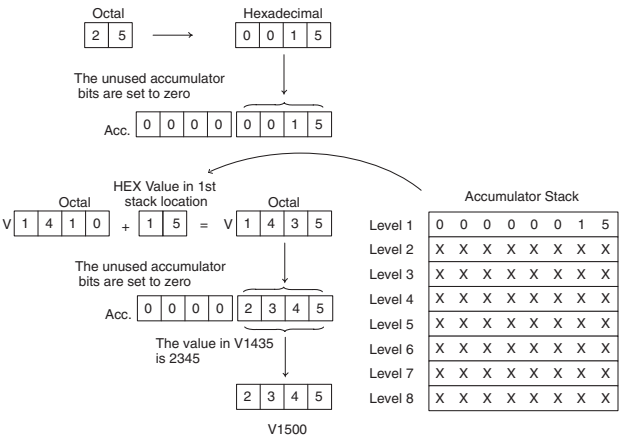
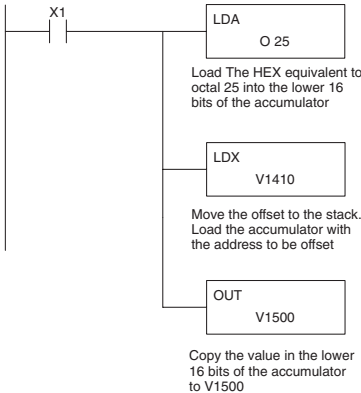
Helpful hint: — The Load Address instruction can be used to convert an octal address to a HEX address and load the value into the accumulator.

Operand Data Type		Range	
		D2-250-1	D2-260/D2-262
	A	aaa	aaa
V-memory	V	All, see Memory map	All, see Memory map
Pointer	P	All V-memory; see Memory map	All V-memory; see Memory map



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

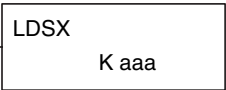
In the following example, when X1 is on, the HEX equivalent for octal 25 will be loaded into the accumulator (this value will be placed on the stack when the Load Accumulator Indexed instruction is executed). V-memory location V1410 will be added to the value in the first level of the stack and the value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



Load Accumulator Indexed from Data Constants (LDSX)

- 230
- 240
- 250-1
- 260
- 262

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



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

DS	Used
HPP	Used

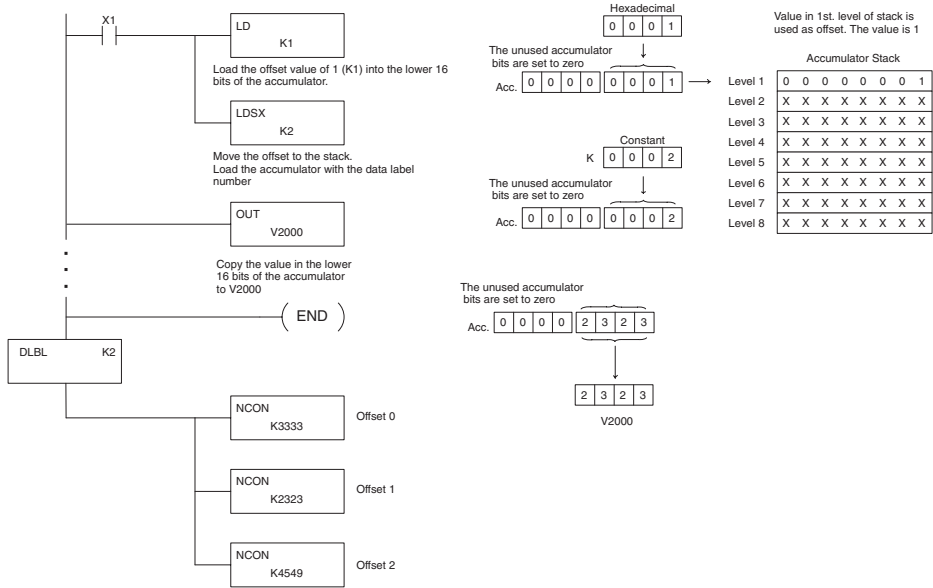
Helpful hint: — The Load Address instruction can be used to convert octal to HEX and load the value into the accumulator.

Operand Data Type	Range		
	D2-240	D2-250-1	D2-260/D2-262
A	aaa	aaa	aaa
Constant	K	1-FFFF	



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

In the following example, when X1 is on, the offset of 1 is loaded into the accumulator. This value will be placed into the first level of the accumulator stack when the LDSX instruction is executed. The LDSX instruction specifies the Data Label (DLBL K2) where the numerical constant(s) are located in the program and loads the constant value, indicated by the offset in the stack, into the lower 16 bits of the accumulator.



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

Load Real Number (LDR)

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

LDR	A aaa
-----	-------

- ☐ 230
☐ 240
☒ 250-1
☒ 260
☒ 262

DS	Used
HPP	N/A

Operand Data Type		Range	
		D2-250-1	D2-260/D2-262
	A	aaa	aaa
V-memory	V	All, see Memory map	All, see Memory map
Pointer	P	All V-memory; see Memory map	All V-memory; see Memory map
Real Constants	R	-3.402823E+038 to +3.402823E+038	-3.402823E+038 to +3.402823E+038

DirectSOFT allows you to enter real numbers directly, by using the leading "R" to indicate a real number entry. You can enter a constant such as Pi, shown in the example to the right. To enter negative numbers, use a minus (-) after the "R".

LDR	R3.14159
-----	----------

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

LDR R5.3E6

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

OUTD
V1400

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

LDR V1400

Out (OUT)

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

OUT
A aaa

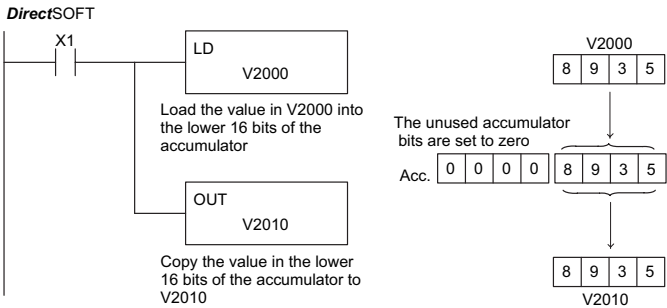
- 230
- 240
- 250-1
- 260
- 262

Operand Data Type	Range			
	D2-230	D2-240	D2-250-1	D2-260/D2-262
A	aaa	aaa	aaa	aaa
V-memory V	All; see Memory map			
Pointer P	-	All V-memory; see Memory map		

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

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.

DS	Used
HPP	Used



Handheld Programmer Keystrokes

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

Out Double (OUTD)

- 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).

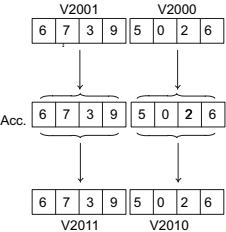
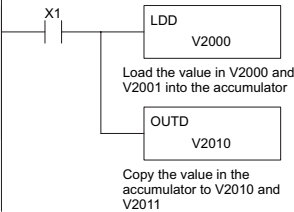
OUTD
A aaa

Operand Data Type		Range			
		D2-230	D2-240	D2-250-1	D2-260/D2-262
A		aaa	aaa	aaa	aaa
V-memory	V	All; see Memory map			
Pointer	P	-	All V-memory; see Memory map		

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.

DS	Used
HPP	Used

DirectSOFT



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)

- ✗

230

✓

240

✓

250-1

✓

260

✓

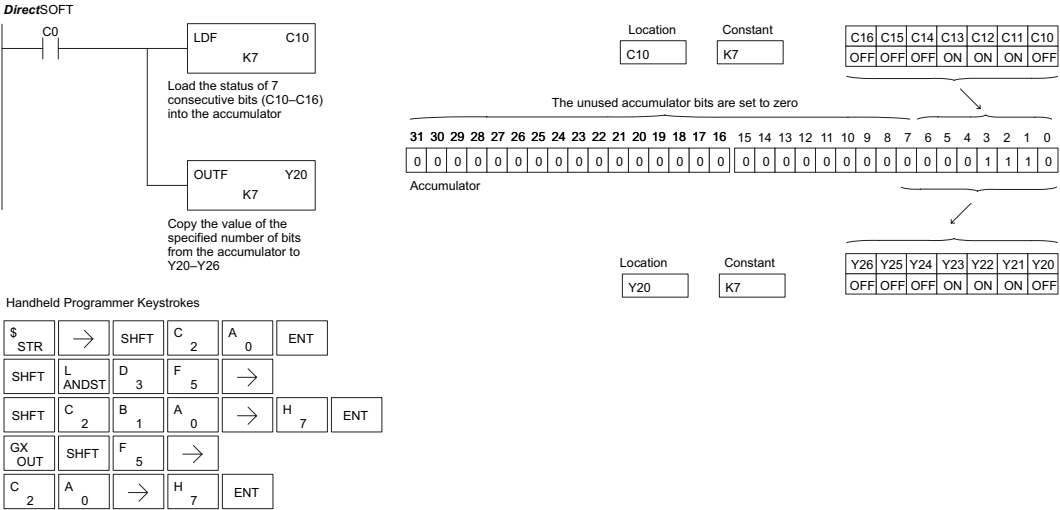
262
- The Out Formatted instruction outputs 1 to 32 bits from the accumulator to the specified discrete memory locations.
The instruction requires a starting location (Aaaa) for the destination and the number of bits (Kbbb) to be output.

OUTF A aaa

K bbb

DS	Used	Operand Data Type	Range			
			D2-240		D2-250-1	
HPP	Used	A	aaa	bbb	aaa	bbb
		Constant	K	1-32	1-32	1-32

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



Out Indexed (OUTX)

- | | |
|-------------------------------------|-------|
| <input type="checkbox"/> | 230 |
| <input type="checkbox"/> | 240 |
| <input checked="" type="checkbox"/> | 250-1 |
| <input checked="" type="checkbox"/> | 260 |
| <input checked="" type="checkbox"/> | 262 |

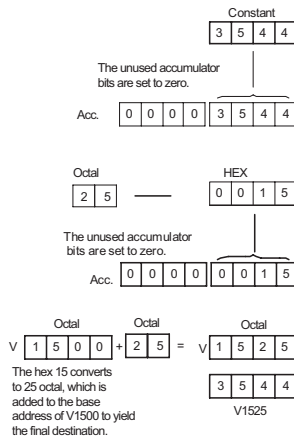
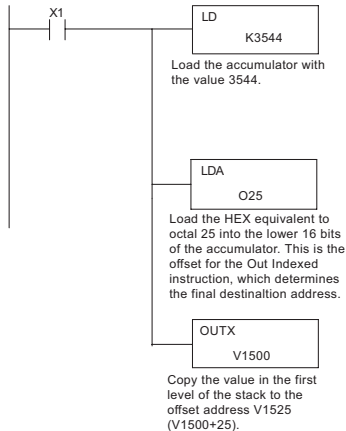
The Out Indexed instruction is a 16-bit instruction. It copies a 16-bit or 4-digit value from the first level of the accumulator stack to a source address offset by the value in the accumulator (V-memory + offset). This instruction interprets the offset value as a HEX number. The upper 16 bits of the accumulator are set to zero.



DS	Used
HPP	Used

Operand Data Type		Range	
		D2-250-1 Range	D2-260/D2-262
A		aaa	aaa
V-memory	V	All. Seememory map	All. See memory map
Pointer	P	All V-memory. See memory map	All V-memory. See memory map

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



Handheld Programmer Keystrokes

[illegible]

Accumulator Stack

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

Out Least (OUTL)

- ✗

230
- ✗

240
- ✗

250-1
- ✓

260
- ✓

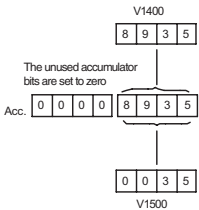
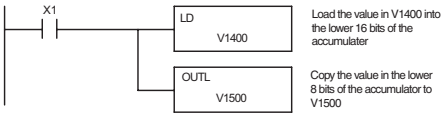
262

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

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

OUTL
A aaa

Operand Data Type	D2-260/D2-262 Range	
	A	aaa
V-memory	V	All V-memory. See memory map
Pointer	P	All V-memory. See memory map



Handheld Programmer Keystrokes

\$	STR	→	B	1	ENT
SHFT	L	ANDST	D	3	→
			B	1	E 4 A 0 A 0 ENT
GX	OUT	SHFT	L	ANDST	→
			B	1	F 5 A 0 A 0 ENT

Out Most (OUTM)

- ✗

230
- ✗

240
- ✗

250-1
- ✓

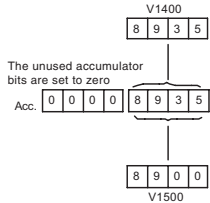
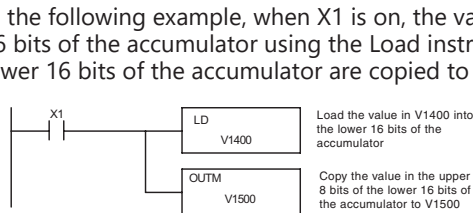
260
- ✓

262

The Out Most instruction copies the value in the upper eight bits of the lower 16 bits of the accumulator to the upper eight bits of the specified V-memory location (i.e., it copies the high byte of the low word of the accumulator).

OUTM
A aaa

Operand Data Type	D2-260/D2-262 Range	
	A	aaa
V-memory	V	All V-memory. See memory map
Pointer	P	All V-memory. See memory map



Handheld Programmer Keystrokes

\$	STR	→	B	1	ENT
SHFT	L	ANDST	D	3	→
			B	1	E 4 A 0 A 0 ENT
GX	OUT	SHFT	M	ORST	→
			B	1	F 5 A 0 A 0 ENT

Pop (POP)

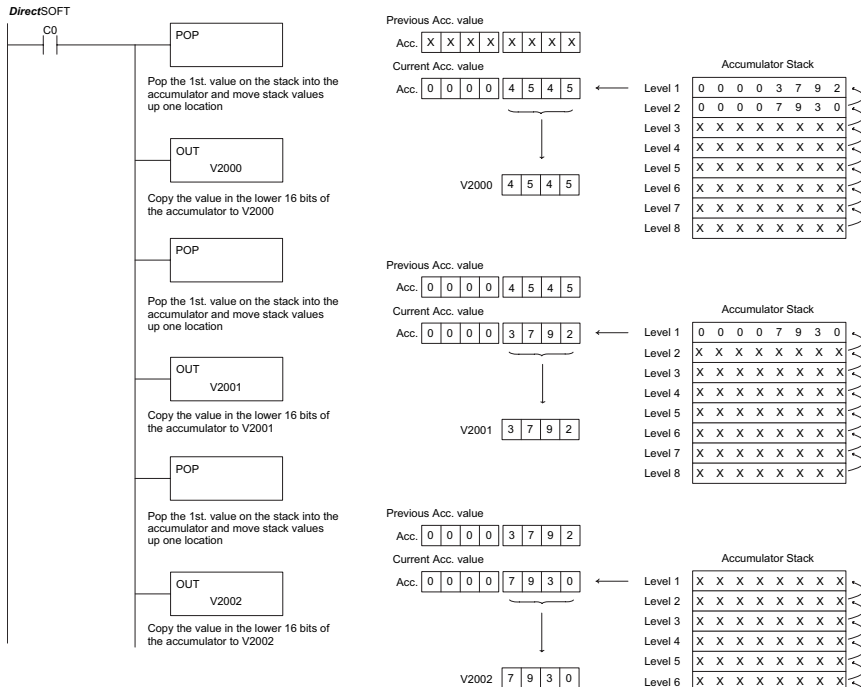
- ✓ 230
- ✓ 240
- ✓ 250-1
- ✓ 260
- ✓ 262

DS	Used
HPP	Used

The Pop instruction moves the value from the first level of the accumulator stack (32 bits) to the accumulator and shifts each value in the stack up one level. 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.

POP

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



Handheld Programmer Keystrokes

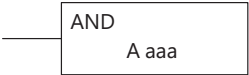
\$ STR	→	SHFT	C 2	A 0	ENT
SHFT	P CV	SHFT	O INST#	P CV	ENT
GX OUT	→	SHFT	V AND	C 2	A 0 A 0 A 0 ENT
SHFT	P CV	SHFT	O INST#	P CV	ENT
GX OUT	→	SHFT	V AND	C 2	A 0 A 0 B 1 ENT
SHFT	P CV	SHFT	O INST#	P CV	ENT
GX OUT	→	SHFT	V AND	C 2	A 0 A 0 C 2 ENT

Logical Instructions (Accumulator)

And (AND)

- ✓ 230
- ✓ 240
- ✓ 250-1
- ✓ 260
- ✓ 262

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	Range			
	D2-230	D2-240	D2-250-1	D2-260/D2-262
A	aaa	aaa	aaa	aaa
V-memory	V	All; see Memory map		
Pointer	P	All V-memory; see Memory map		

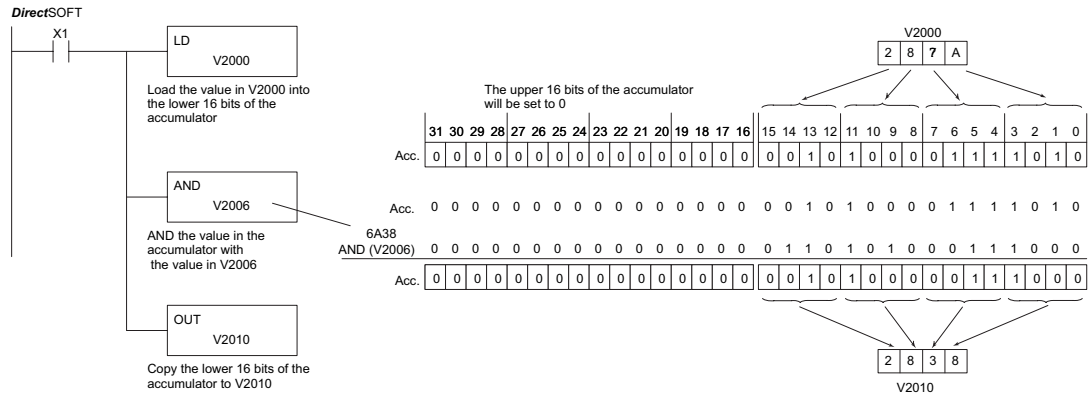
Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.



NOTE: 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.

DS	Used
HPP	Used



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)

- | | |
|---------|---|
| ✓ 230 | 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). |
| ✓ 240 | |
| ✓ 250-1 | |
| ✓ 260 | |
| ✓ 262 | |

ANDD
A aaa

DS	Used
HPP	Used

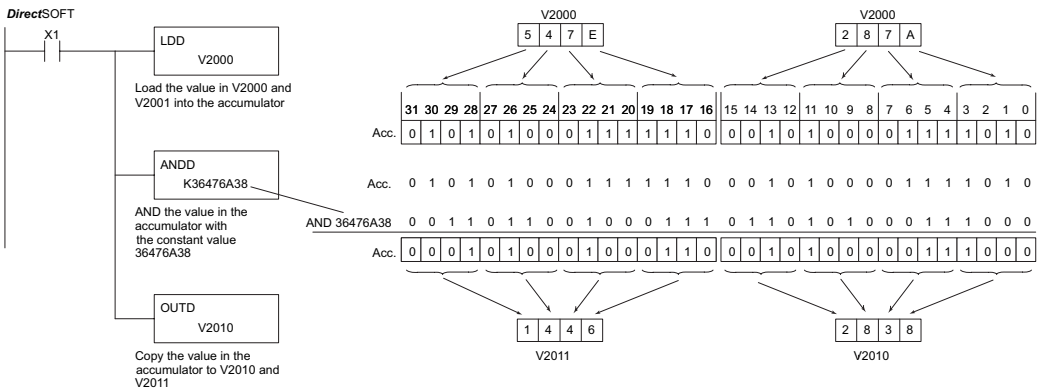
Operand Data Type		Range			
		D2-230	D2-240	D2-250-1	D2-260/D2-262
	A	aaa	aaa	aaa	aaa
V-memory	V	-	-	All; see Memory map	
Pointer	P	-	-	All V-memory; see Memory map	
Constant	K	0-FFFFFFFF	0-FFFFFFFF	0-FFFFFFFF	0-FFFFFFFF

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																	

And Formatted (ANDF)

230

240

☒ 250-1

☒ 260

262

The And Formatted instruction logically ANDs the binary value in the accumulator and a specified range of discrete memory bits (1 to 32). The instruction requires a starting location (Aaaa) and number of bits (Kbbb) to be ANDed. Discrete status flags indicate if the result is zero or a negative number (the most significant bit = 1).

ANDF	A aaa
	K bbb

DS	Used
HPP	Used

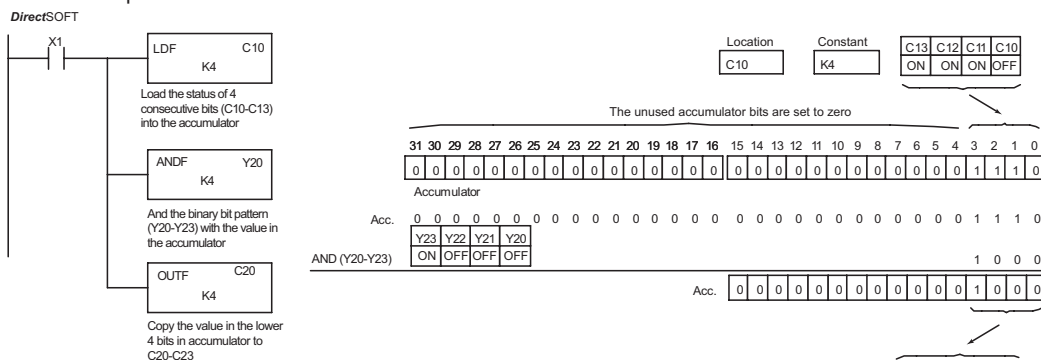
Operand Data Type		D2-250-1 Range		D2-260/D2-262 Range	
	A	aaa	bbb	aaa	bbb
Inputs	X	0-777	-	0-1777	-
Outputs	Y	0-777	-	0-1777	-
Control Relays	C	0-1777	-	0-3777	-
Stage bits	S	0-1777	-	0-1777	-
Timer bits	T	0-377	-	0-377	-
Counter bits	CT	0-177	-	0-377	-
Special Relay	SP	0-777	-	0-777	-
Global I/O	GX/ SY	-	-	0-3777	-
Constant	K	-	1-32	-	1-32

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



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

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



Handheld Programmer Keystrokes

\$	STR	→	B ₁	ENT											C ₀
SHFT	L	AND	D ₃	F ₅	→	NEXT	NEXT	NEXT	NEXT	B ₁	A ₀	→	E ₄	ENT	
V	AND	SHFT	F ₅	→	NEXT	C ₂	A ₀	→	E ₄	ENT					
GX	OUT	SHFT	F ₅	→	PREV	PREV	C ₂	A ₀	→	E ₄	ENT				

And with Stack (ANDS)

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

ANDS

- 230
- 240
- 250-1
- 260
- 262

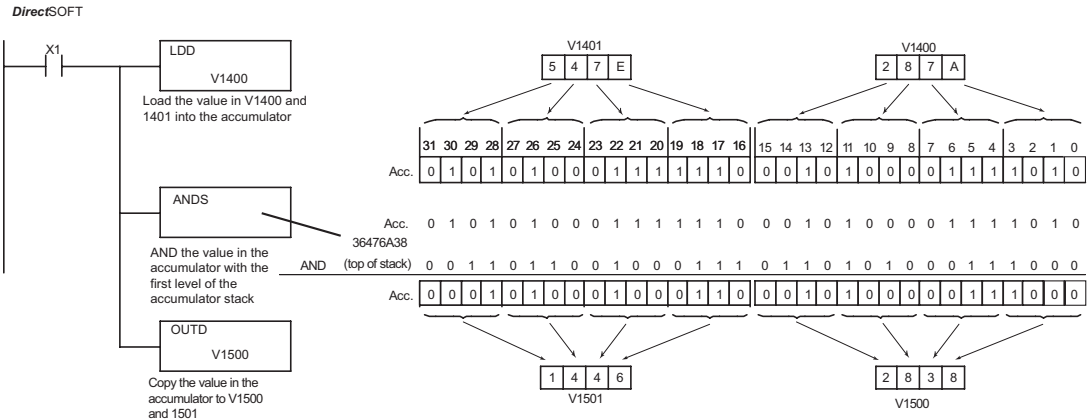
DS	Used
HPP	Used

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



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

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



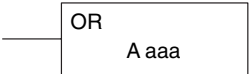
Handheld Programmer Keystrokes

\$ STR	→	B 1	ENT
SHFT	L ANDST	D 3	D 3 → B 1 E 4 A 0 A 0 ENT
V AND	SHFT	S RST	ENT
GX OUT	SHFT	D 3	→ B 1 F 5 A 0 A 0 ENT

Or (OR)

- ✓ 230
- ✓ 240
- ✓ 250-1
- ✓ 260
- ✓ 262

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.



DS	Used
HPP	Used

Operand Data Type	Range			
	D2-230	D2-240	D2-250-1	D2-260/D2-262
A	aaa	aaa	aaa	aaa
V-memory	V	All; see Memory map		
Pointer	P	All V-memory; see Memory map		

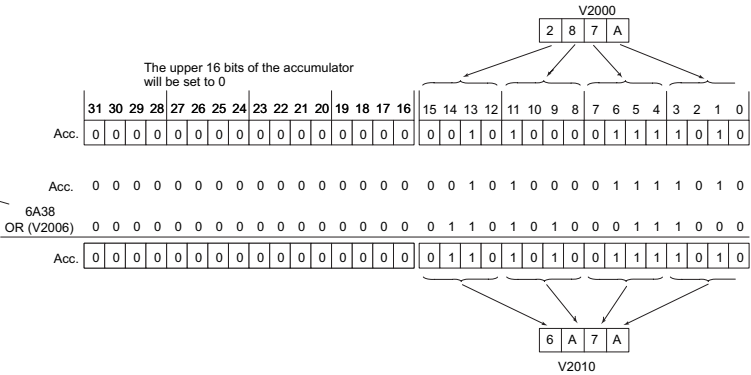
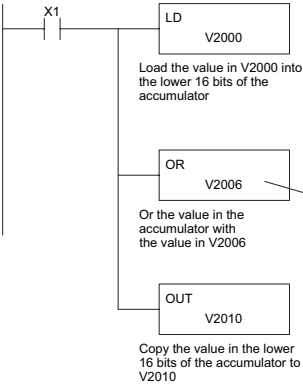
Discrete Bit Flags	Description
SP63	Will be ON if the result in the accumulator is zero.



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.

DirectSOFT



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)

- ✓ 230
- ✓ 240
- ✓ 250-1
- ✓ 260
- ✓ 262

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).

ORD
A aaa

DS	Used
HPP	Used

Operand Data Type		Range			
		D2-230	D2-240	D2-250-1	D2-260/D2-262
A		aaa	aaa	aaa	aaa
V-memory	V	-	-	All; see Memory map	
Pointer	P	-	-	All V-memory; see Memory map	
Constant	K	0-FFFFFFF	0-FFFFFFF	0-FFFFFFF	0-FFFFFFF

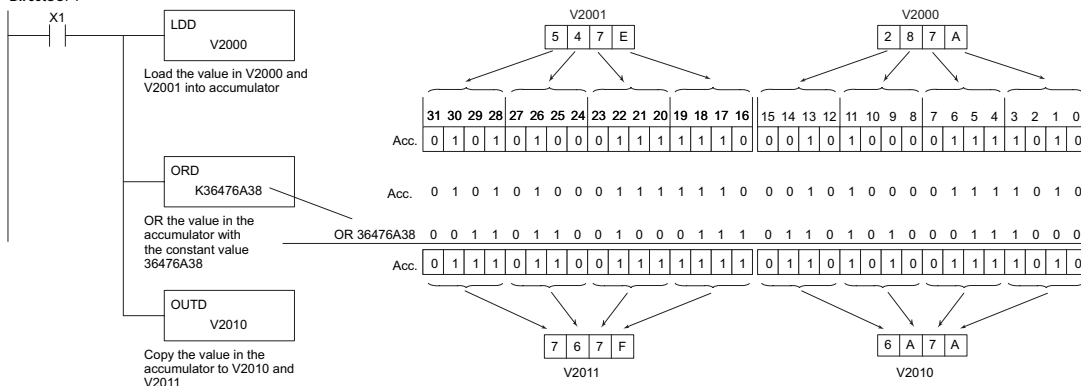
Discrete Bit Flags	Description
SP63	Will be on if the result in the accumulator is zero
SP70	Will be on if the result in the accumulator is negative



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.

DirectSOFT



Handheld Programmer Keystrokes

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

Or Formatted (ORF)

- 230
- 240
- 250-1
- 260
- 262

The Or Formatted instruction logically ORs the binary value in the accumulator and a specified range of discrete bits (1 to 32). The instruction requires a starting location (Aaaa) and the number of bits (Kbbb) to be ORed. Discrete status flags indicate if the result is zero or negative (the most significant bit = 1).

ORF A aaa
 K bbb

DS	Used
HPP	Used

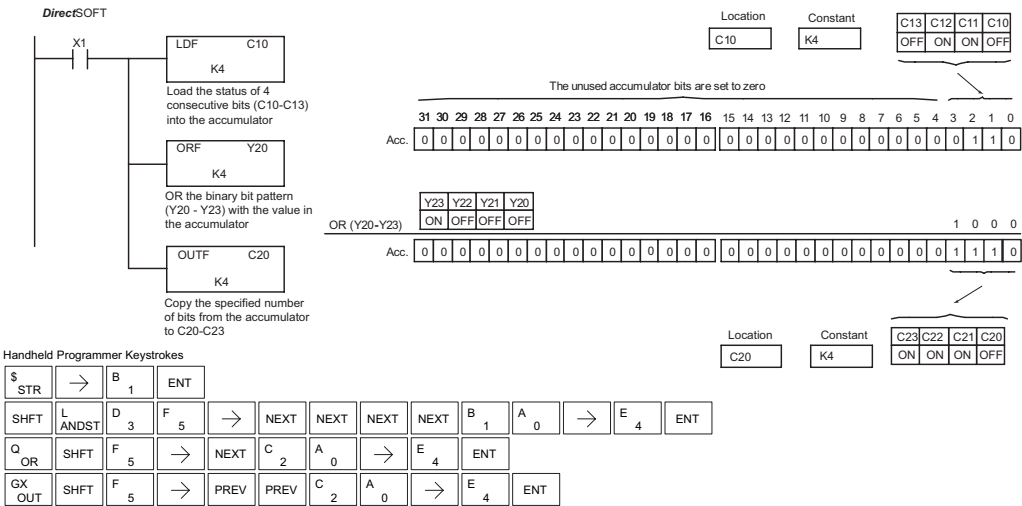
Operand Data Type		D2-250-1 Range	D2-260/D2-262 Range
	A	aaa	bbb
Inputs	X	0-777	-
Outputs	Y	0-777	-
Control Relays	C	0-1777	-
Stage bits	S	0-1777	-
Timer bits	T	0-377	-
Counter bits	CT	0-177	-
Special Relay	SP	0-777	-
Global I/O	GX/ GY	-	0-3777
Constant	K	-	1-32

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



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

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



Or with Stack (ORS)

- ✗ 230
- ✗ 240
- ✗ 250-1
- ✓ 260
- ✓ 262

DS	Used
HPP	Used

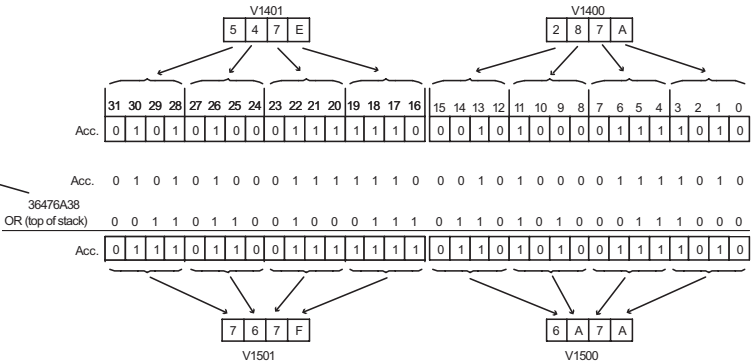
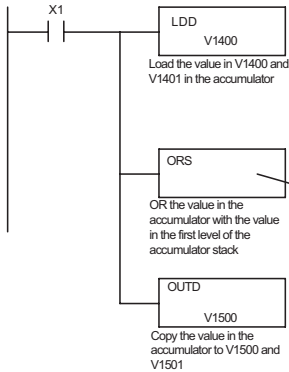
The Or with Stack instruction is a 32-bit instruction that logically ORs the value in the accumulator with the first level of the accumulator stack. The result resides in the accumulator. The value in the first level of the accumulator stack is removed from the stack and all values are moved up one level.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

ORS

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

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

DirectSOFT



Handheld Programmer Keystrokes

\$	STR	→	B ₁	ENT																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																				</
----	-----	---	----------------	-----	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	----

Exclusive Or Double (XORD)

- ✓

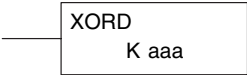
230
- ✓

240
- ✓

250-1
- ✓

260
- ✓

262
- 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).



DS	Used
HPP	Used

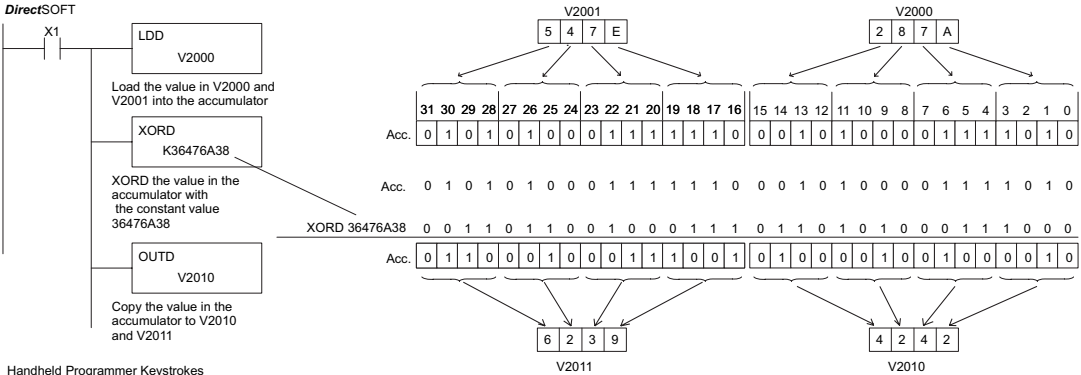
Operand Data Type	Range			
	D2-230	D2-240	D2-250-1	D2-260/D2-262
A	aaa	aaa	aaa	aaa
Constant	K	0-FFFFFFF	0-FFFFFFF	0-FFFFFFF



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

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

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

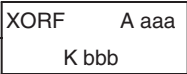
\$	STR	→	B	1	ENT
SHFT	L	ANDST	D	3	→
SHFT	X	SET	Q	OR	SHFT
D	3	G	6	E	4
GX	OUT	SHFT	D	3	→
C	2	A	0	B	1
A	0	B	1	A	0
A	0	ENT			

Exclusive OR Formatted (XORF)

- 230
- 240
- 250-1
- 260
- 262

The Exclusive Or Formatted instruction performs an exclusive OR of the binary value in the accumulator and a specified range of discrete memory bits (1 to 32).

The instruction requires a starting location (Aaaa) and the number of bits (Kbbb) to be exclusive OR'd. Discrete status flags indicate if the result of the Exclusive Or Formatted is zero or negative (the most significant bit is on).



DS	Used
HPP	Used

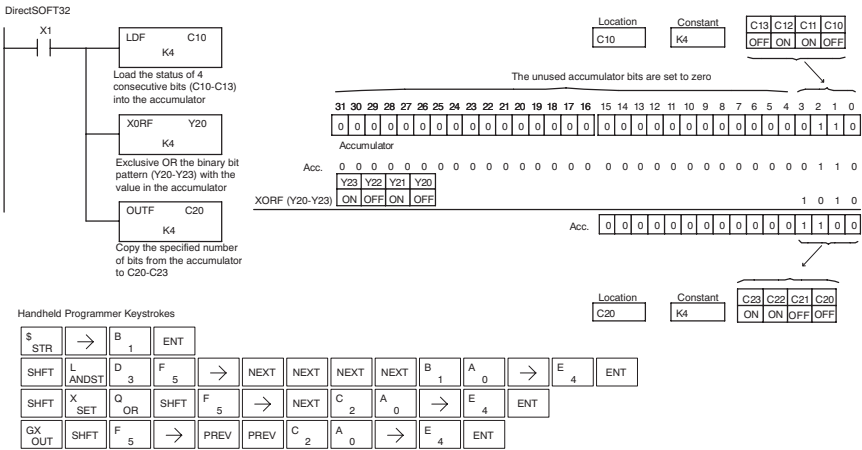
Operand Data Type		D2-250-1 Range		D2-260/D2-262 Range	
	A	aaa	bbb	aaa	bbb
Inputs	X	0-777	-	0-1777	-
Outputs	Y	0-777	-	0-1777	-
Control Relays	C	0-1777	-	0-3777	-
Stage bits	S	0-1777	-	0-1777	-
Timer bits	T	0-377	-	0-377	-
Counter bits	CT	0-177	-	0-377	-
Special Relay	SP	0-777	-	0-777	-
Global I/O	GX/ GY	-	-	0-3777	-
Constant	K	-	1-32		1-32

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



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

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



Exclusive Or with Stack (XORS)

-  230
-  240
-  250-1
-  260
-  262

DS	Used
HPP	Used

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

XORS

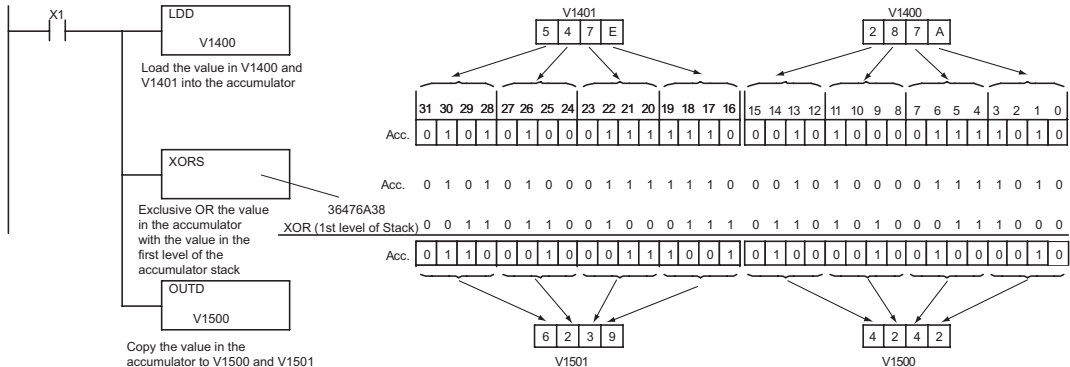
Discrete Bit Flags	Description
SP63	Will be on if the result in the accumulator is zero
SP70	Will be on if the result in the accumulator is negative



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

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

DirectSOFT



Handheld Programmer Keystrokes

\$	STR	→	B	1	ENT									
SHFT	L	AND	D	3	→	B	1	E	4	A	0	A	0	ENT
SHFT	X	SET	Q	OR	SHFT	S	RST	ENT						
GX	OUT	SHFT	D	3	→	B	1	F	5	A	0	A	0	ENT

Compare (CMP)

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.

CMP
A aaa

- 230
- 240
- 250-1
- 260
- 262

DS	Used
HPP	Used

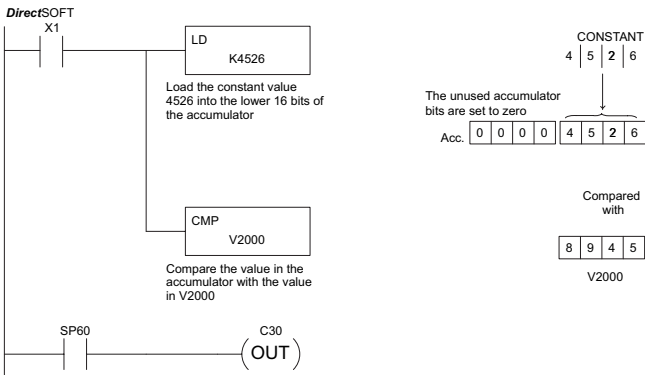
Operand Data Type	Range			
	D2-230	D2-240	D2-250-1	D2-260/D2-262
A	aaa	aaa	aaa	aaa
V-memory V	All; see Memory map			
Pointer P	-	All V-memory; see Memory map		

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



NOTE: 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)

- 230
- 240
- 250-1
- 260
- 262

DS	Used
HPP	Used

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

CMPD
A aaa

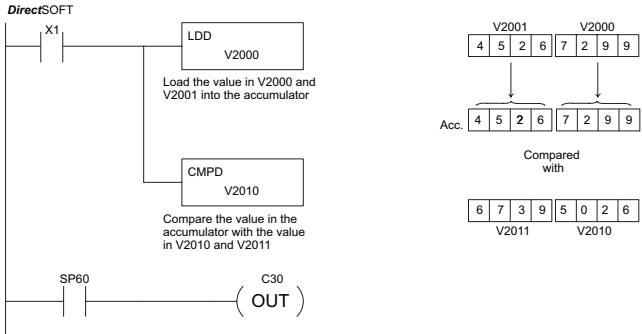
Operand Data Type		Range			
		D2-230	D2-240	D2-250-1	D2-260/D2-262
	A	aaa	aaa	aaa	aaa
V-memory	V	All; see Memory map			
Pointer	P	–	All V-memory; see Memory map		
Constant	K	0-FFFFFFF		0-FFFFFFF	

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

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

Compare Formatted (CMPF)

- ✗

230

✗

240

✓

250-1

✓

260

✓

262
- The Compare Formatted compares the value in the accumulator with a specified number of discrete locations (1–32). The instruction requires a starting location (Aaaa) and the number of bits (Kbbb) to be compared. The corresponding status flag will be turned on indicating the result of the comparison.

CMPF A aaa

K bbb

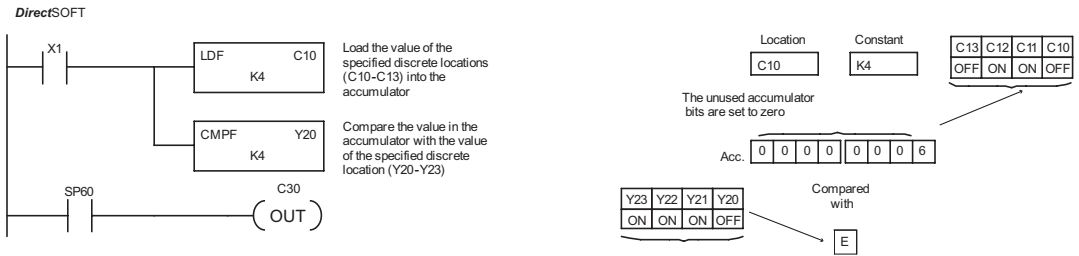
DS	Used	Operand Data Type	D2-250-1 Range		D2-260/D2-262 Range		
HPP	Used	A	aaa	bbb	aaa	bbb	
		Inputs	X	0-777	-	0-1777	-
		Outputs	Y	0-777	-	0-1777	-
		Control Relays	C	0-1777	-	0-3777	-
		Stage bits	S	0-1777	-	0-1777	-
		Timer bits	T	0-377	-	0-377	-
		Counter bits	CT	0-177	-	0-377	-
		Special Relay	SP	0-777	-	0-777	-
		Global I/O	GX/ GY	-	-	0-3777	-
		Constant	K	-	1-32		1-32

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



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

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



Compare with Stack (CMPS)

- 230
- 240
- 250-1
- 260
- 262

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

CMPS

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

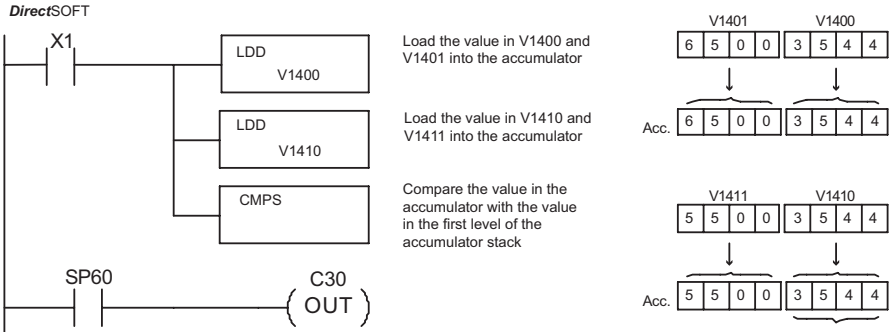
DS	Used
HPP	Used

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



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

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



Handheld Programmer Keystrokes

\$ STR	→	B 1	ENT
SHFT	L ANDST	D 3	D 3 → B 1 E 4 A 0 A 0 ENT
SHFT	L ANDST	D 3	D 3 → B 1 E 4 B 1 A 0 ENT
SHFT	C 2	SHFT	M ORST P CV S RST ENT
\$ STR	→	SHFT	SP STRN G 6 A 0 ENT
GX OUT	→	SHFT	C 2 D 3 A 0 ENT

Compared with Top of Stack

Compare Real Number (CMPR)

- ☒

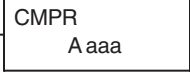
230
- ☒

240
- ☒

250-1
- ☒

260
- ☒

262
- The Compare Real Number instruction compares a real number value in the accumulator with two consecutive V-memory locations containing a real number. The corresponding status flag will be turned on indicating the result of the comparison. Both numbers being compared are 32 bits long.



DS	Used
HPP	N/A

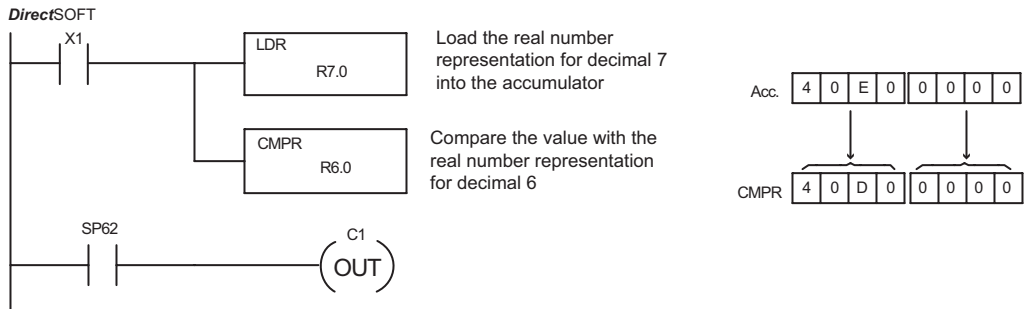
Operand Data Type		Range	
		D2-250-1	D2-260/D2-262
	A	aaa	aaa
V-memory	V	All V-memory; see Memory map	All V-memory; see Memory map
Pointer	P	All V-memory; see Memory map	All V-memory; see Memory map
Constant	R	-3.402823E+038 to +3.402823E+038	-3.402823E+038 to +3.402823E+038

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



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

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



Math Instructions

Add (ADD)

- ✓ 230
- ✓ 240
- ✓ 250-1
- ✓ 260
- ✓ 262

DS	Used
HPP	Used

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

ADD
A aaa

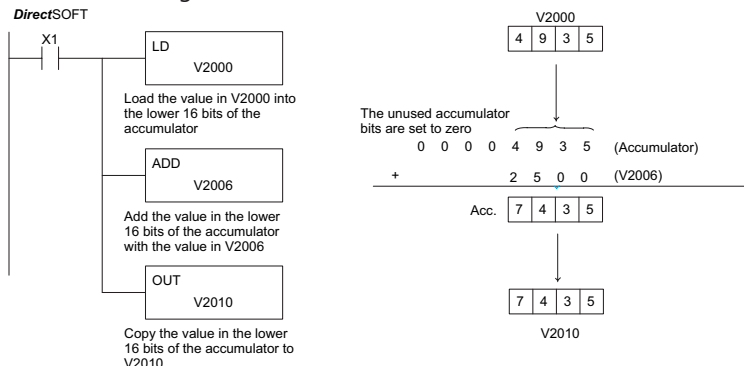
Operand Data Type	Range			
	D2-230	D2-240	D2-250-1	D2-260/D2-262
A	aaa	aaa	aaa	aaa
V-memory V	All; see Memory map			
Pointer P	-	All V-memory; see Memory map		

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



Handheld Programmer Keystrokes

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

Add Double (ADDD)

- 230
- 240
- 250-1
- 260
- 262

DS	Used
HPP	Used

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

ADDD
A aaa

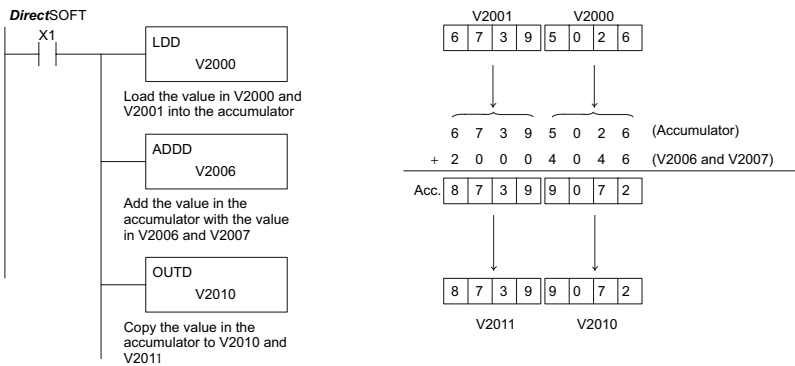
Operand Data Type		Range			
		D2-230	D2-240	D2-250-1	D2-260/D2-262
	A	aaa	aaa	aaa	aaa
V-memory	V	All; see Memory map			
Pointer	P	-	All V-memory; see Memory map		
Constant	K	0-99999999	0-99999999		

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

Add Real (ADDR)

- ☒ 230
- ☒ 240
- ☒ 250-1
- ☒ 260
- ☒ 262

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

ADDR

A aaa

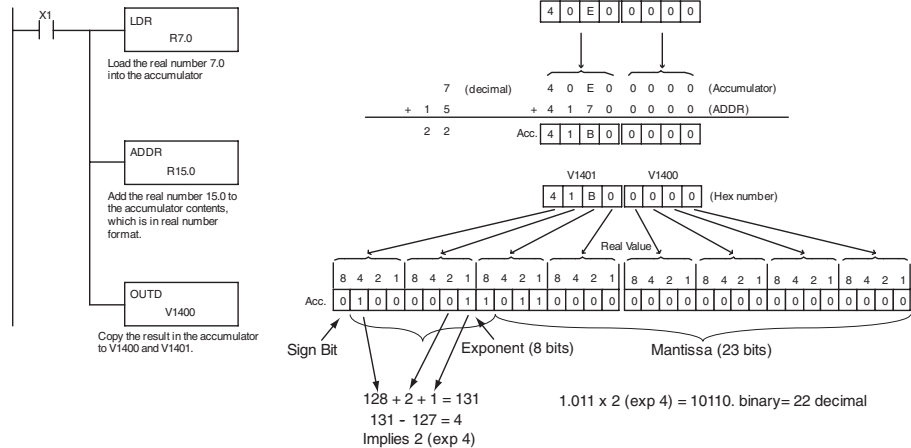
DS	Used
HPP	N/A

Operand Data Type		Range	
		D2-250-1	D2-260/D2-262
	A	aaa	aaa
V-memory	V	All V-memory; see Memory map	All V-memory; see Memory map
Pointer	P	All V-memory; see Memory map	All V-memory; see Memory map
Constant	R	-3.402823E+038 to +3.402823E+038	-3.402823E+038 to +3.402823E+038

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

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

DirectSOFT



NOTE 1: The current HPP does not support real number entry with automatic conversion to the 32-bit IEEE format. You must use DirectSOFT for this feature.

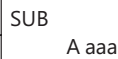
NOTE 2: If the value being added to a real number is 16,777,216 times smaller than the real number, the calculation will not work.

Subtract (SUB)

- 230
- 240
- 250-1
- 260
- 262

DS	Used
HPP	Used

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



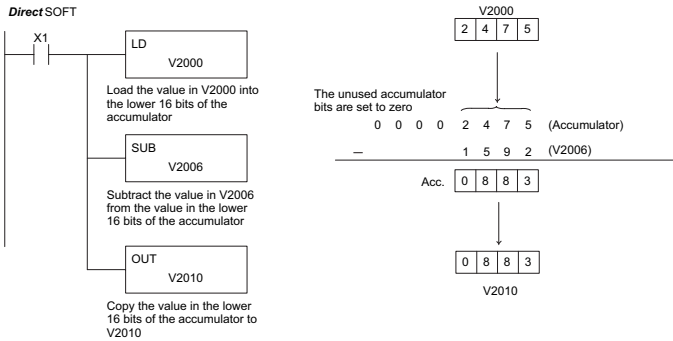
Operand Data Type	Range			
	D2-230	D2-240	D2-250-1	D2-260/D2-262
A	aaa	aaa	aaa	aaa
V-memory	V	All; see Memory map		
Pointer	P	-	All V-memory; see Memory map	

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero
SP66	On when the 16-bit addition instruction results in a carry
SP67	On when the 32-bit addition instruction results in a carry
SP70	On anytime the value in the accumulator is negative
SP75	On when a BCD instruction is executed and a NON-BCD number 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	→
SHFT	S	RST	U	ISG	B
GX	OUT	→	SHFT	V	AND
			C	2	A
			A	0	B
			A	0	A
			A	0	ENT
			A	0	G
			A	0	6
			A	0	ENT

230 Subtract Double (SUBD)

- 240
- 250-1
- 260
- 262

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

SUBD
A aaa

Operand Data Type		Range			
		D2-230	D2-240	D2-250-1	D2-260/D2-262
	A	aaa	aaa	aaa	aaa
V-memory	V	All (See page 3-54)	All (See page 3-55)	All (See page 3-56)	All (See page 3-57)
Pointer	P	-	All V-memory (See page 3-55)	All V-memory (See page 3-56)	All V-memory (See page 3-57)
Constant	K	0-99999999	0-99999999	0-99999999	0-99999999

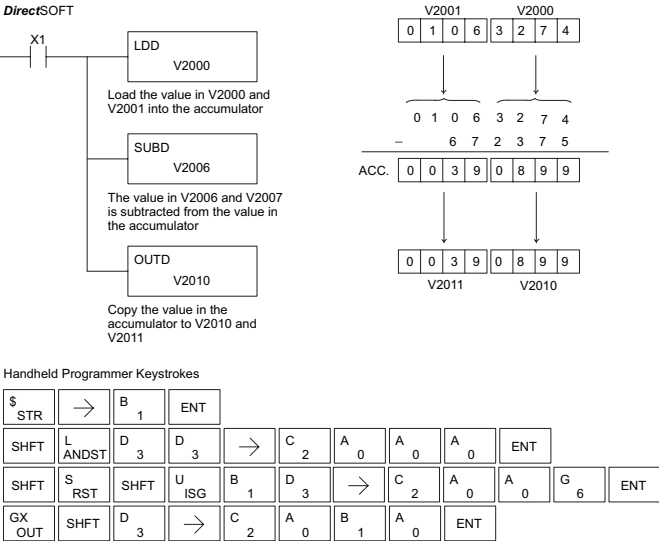
Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero
SP66	On when the 16-bit addition instruction results in a carry
SP67	On when the 32-bit addition instruction results in a carry
SP70	On anytime the value in the accumulator is negative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

DS	Used
HPP	Used

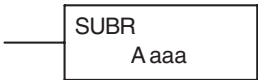


Subtract Real (SUBR)

- 230
- 240
- 250-1
- 260
- 262

DS	Used
HPP	N/A

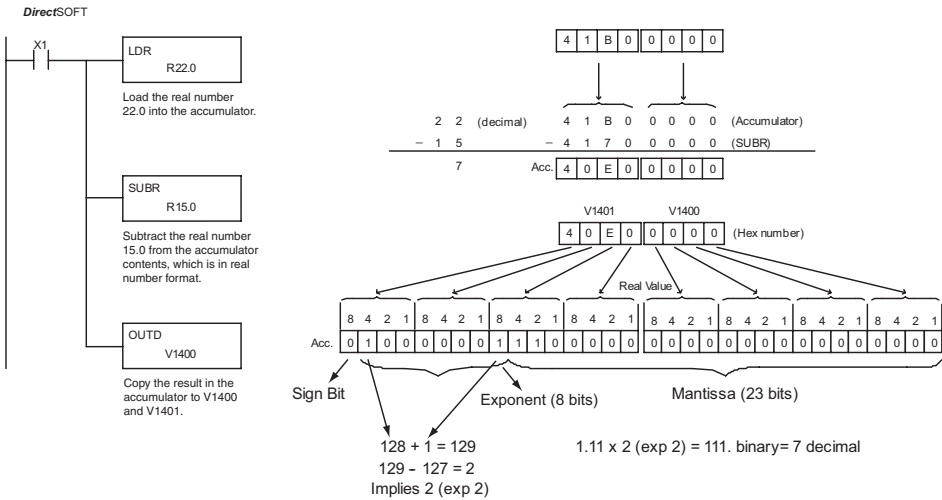
The Subtract Real is a 32-bit instruction that subtracts a real number, which is either two consecutive V-memory locations or a 32-bit constant, from a real number in the accumulator. The result is a 32-bit real number that resides in the accumulator. Both numbers must be Real data type (IEEE floating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



Operand Data Type		Range	
		D2-250-1	D2-260/D2-262
	A	aaa	aaa
V-memory	V	All. (See page 3-56)	All. (See page 3-57)
Pointer	P	All V-memory (See page 3-56)	All V-memory (See page 3-57)
Constant	R	-3.402823E+038 to + 3.402823E+038	-3.402823E+038 to + 3.402823E+038

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

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

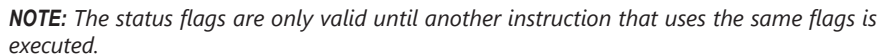


NOTE: The current HPP does not support real number entry with automatic conversion to the 32-bit IEEE format. You must use DirectSOFT for this feature.

- ☒ 230
- ☒ 240
- ☒ 250-1
- ☒ 260
- ☒ 262

MUL A aaa

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 is encountered



Double
Action.

```

graph LR
    X1((X1)) --- LD[LD V2000]
    LD --- MUL[MUL V2006]
    MUL --- OUTD[OUTD V2010]
  
```

DirectSOFT

LD V2000

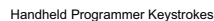
Load the value in V2000 into the lower 16 bits of the accumulator

MUL V2006

The value in V2006 is multiplied by the value in the accumulator

OUTD V2010

Copy the value in the accumulator to V2010 and V2011

5-94

Multiply Double (MULD)

- 230
- 240
- 250-1
- 260
- 262

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

MULD
A aaa

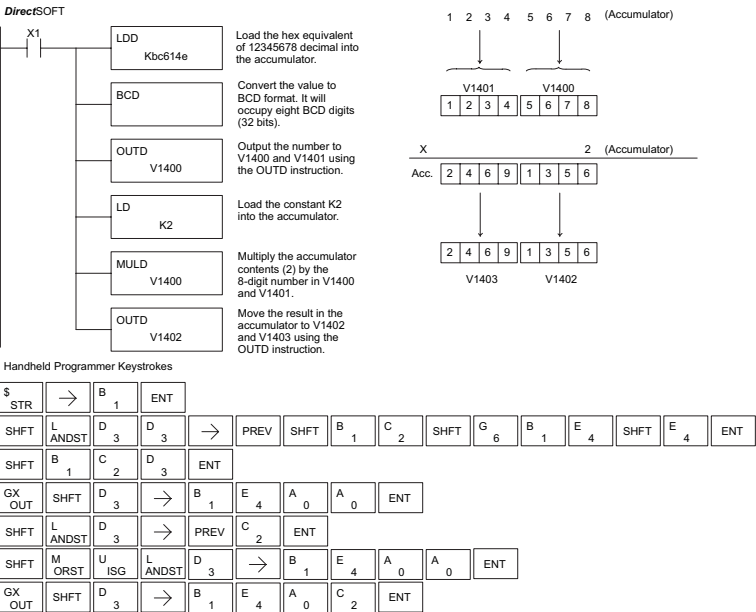
Operand Data Type	Range	
	D2-250-1 Range	D2-260/D2-262
A	aaa	aaa
V-memory	All V-mem (See page 3-56)	All V-mem (See page 3-57)
Pointer	All V-mem (See page 3-56)	All V-mem (See page 3-57)

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 is encountered

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

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

DS	Used
HPP	Used



Multiply Real (MULR)

- 230
- 240
- 250-1
- 260
- 262

DS	Used
HPP	N/A

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

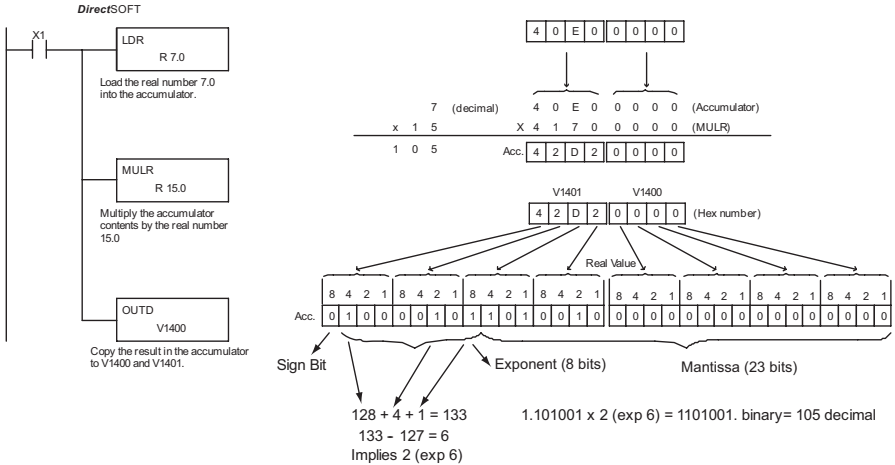
MULR
A aaa

Operand Data Type		Range	
		D2-250-1	D2-260/D2-262
	A	aaa	aaa
V-memory	V	All. (See page 3-56)	All. (See page 3-57)
Pointer	P	All V-memory (See page 3-56)	All V-memory (See page 3-57)
Constant	R	-3.402823E+038 to +3.402823E+038	-3.402823E+038 to +3.402823E+038

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



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

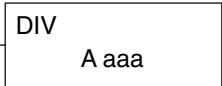


NOTE: The current HPP does not support real number entry with automatic conversion to the 32-bit IEEE format. You must use DirectSOFT for this feature.

- 230
- 240
- 250-1
- 260
- 262

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.



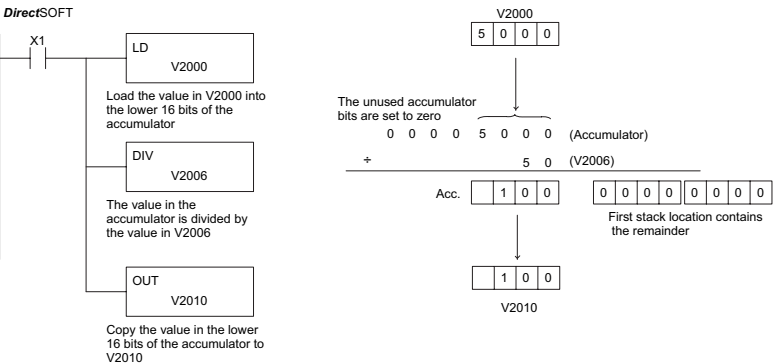
Operand Data Type		Range			
		D2-230	D2-240	D2-250-1	D2-260/D2-262
A		aaa	aaa	aaa	aaa
V-memory	V	All (See page 3-54)	All (See page 3-55)	All (See page 3-56)	All (See page 3-57)
Pointer	P	-	All V-memory (See page 3-55)	All V-memory (See page 3-56)	All V-memory (See page 3-57)
Constant	K	1-9999	1-9999	1-9999	1-9999

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.

DS	Used
HPP	Used



Handheld Programmer Keystrokes

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

Divide Double (DIVD)

☐ 230

☐ 240

☒ **250-1**

☒ **260**

☒ **262**

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

DIVD
A aaa

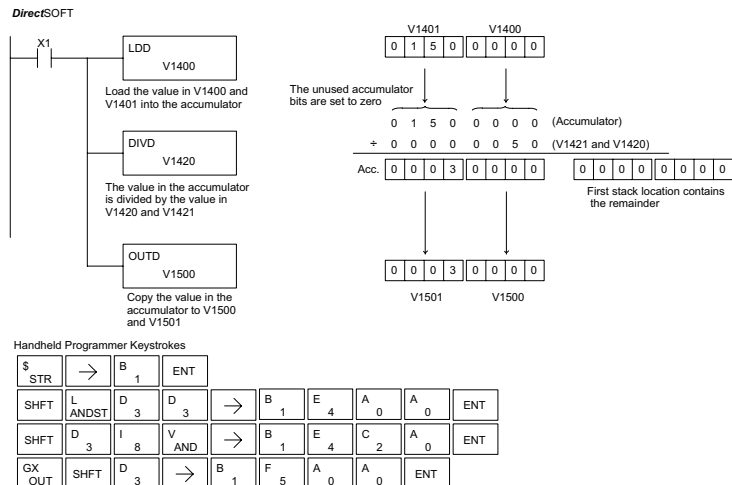
DS	Used
HPP	Used

Operand Data Type		Range	
		D2-250-1 Range	D2-260/D2-262
A		aaa	aaa
V-memory	V	All V-mem (See page 3-56)	All V-mem (See page 3-57)
Pointer	P	All V-mem (See page 3-56)	All V-mem (See page 3-57)

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

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

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



Divide Real (DIVR)

- 230
- 240
- 250-1
- 260
- 262

DS	Used
HPP	N/A

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

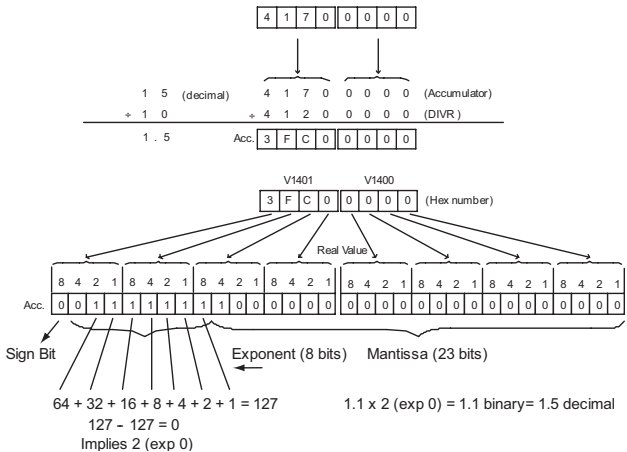
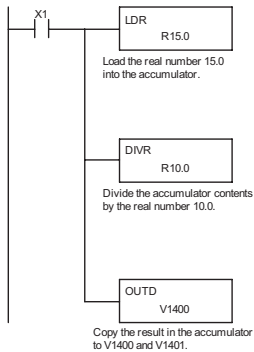
DIVR
A aaa

Operand Data Type		Range	
		D2-250-1	D2-260/D2-262
	A	aaa	aaa
V-memory	V	All. (See page 3-56)	All. (See page 3-57)
Pointer	P	All V-memory (See page 3-56)	All V-memory (See page 3-57)
Constant	R	-3.402823E+038 to +3.402823E+038	-3.402823E+038 to +3.402823E+038

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

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

DirectSOFT



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

Increment (INC)

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

INC
A aaa

✗ 230

✗ 240

✓ 250-1

✓ 260

✓ 262

Decrement (DEC)

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

DEC
A aaa

DS	Used
HPP	Used

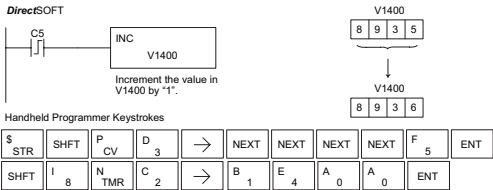
Operand Data Type		Range	
		D2-250-1 Range	D2-260/D2-262
A		aaa	aaa
V-memory	V	All V-mem (See page 3-56)	All V-mem (See page 3-57)
Pointer	P	All V-mem (See page 3-56)	All V-mem (See page 3-57)

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

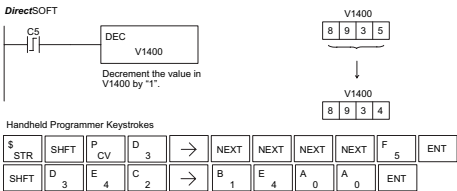


NOTE: Status flags are valid only until another instruction uses the same flag.

In the following increment example, the value in V1400 increases by one each time that C5 is closed (true).



In the following decrement example, the value in V1400 is decreased by one each time that C5 is closed (true).



NOTE: Use a pulsed contact closure to INC/DEC the value in V-memory once per closure.

Add Binary (ADDB)

- 230
- 240
- 250-1
- 260
- 262

DS	Used
HPP	Used

The Add Binary instruction adds a 16-bit number (Aaaa) to the value stored in the accumulator. The number in the accumulator can be up to 32 bits long. The source of the 16-bit operand can be a constant or a data value located in V-memory. Add Binary performs the addition operation on the full binary representation of the operands, which distinguishes it from the Add instruction (see page 5-88), which treats the operands as BCD numbers. Although the addition operation is performed on the underlying binary values, the native display format is hexadecimal. For that reason you will need to load constants in hex.

The sum of the Add Binary operation occupies the full 32-bit accumulator and requires an Out Double to move the sum to V-memory. If the value in the accumulator occupies fewer than 32 bits, leading zeros are loaded in the left-most empty bit positions.

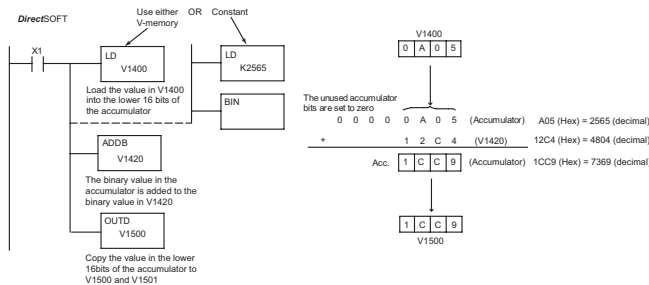
ADDB
A aaa

Operand Data Type	Range	
	D2-250-1	D2-260/D2-262
A	aaa	aaa
V-memory	V	All (See page 3-56)
Pointer	P	All V-memory (See page 3-56)
Constant	K	All V-memory (See page 3-57)
	K	0-FFFF

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

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

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



Handheld Programmer Keystrokes

STR	→	1	ENT
SHFT	L	D	1 4 0 0 ENT
SHFT	A	D	D B → 1 4 2 0 ENT
OUT	SHFT	D	→ 1 5 0 0 ENT

Add Binary Double (ADDBD)

- 230
- 240
- 250-1
- 260
- 262

DS	Used
HPP	Used

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

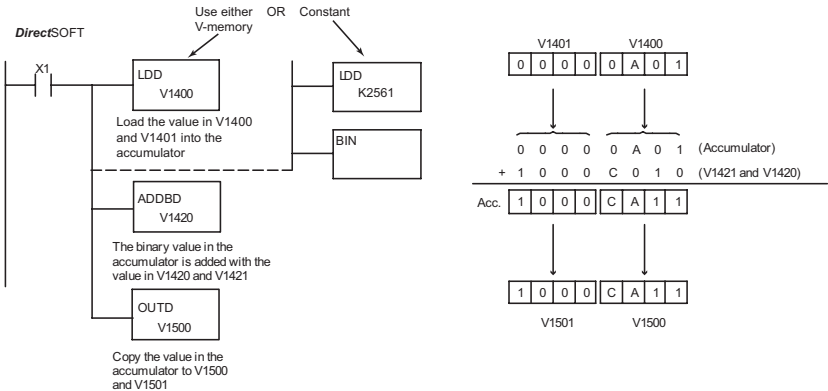
ADDBD
A aaa

Operand Data Type	Range D2-260/D2-262
A	aaa
V-memory	V All (See page 3-57)
Pointer	P All V-memory (See page 3-57)
Constant	K 0-FFFFFFF

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

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

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



Handheld Programmer Keystrokes

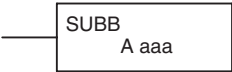
STR	1	←
LD	SHIFT	D
SHIFT	1	4
0	0	←
ADD	SHIFT	B
D	SHIFT	1
4	2	0
←	←	←
OUT	SHIFT	D
SHIFT	1	5
0	0	←

Subtract Binary (SUBB)

- 230
- 240
- 250-1
- 260
- 262

DS	Used
HPP	Used

The Subtract Binary instruction subtracts a 16-bit number (Aaaa) from the value stored in the accumulator. The number in the accumulator can be up to 32 bits long. The source of the 16-bit operand can be a constant or a data value located in V-memory. Subtract Binary performs the subtraction operation on the full binary representation of the operands, which distinguishes it from the Subtract instruction (see page 5-91), which treats the operands as BCD numbers. Although the subtraction operation is performed on the underlying binary values, the native display format is hexadecimal. For that reason, you will need to load constants in hex.



The difference (result) of the Subtract Binary operation occupies the full 32 bits of the accumulator and requires an Out Double to move the value to V-memory. If the value in the accumulator occupies fewer than 32 bits, leading zeros are loaded in the left-most empty bit positions of the accumulator.

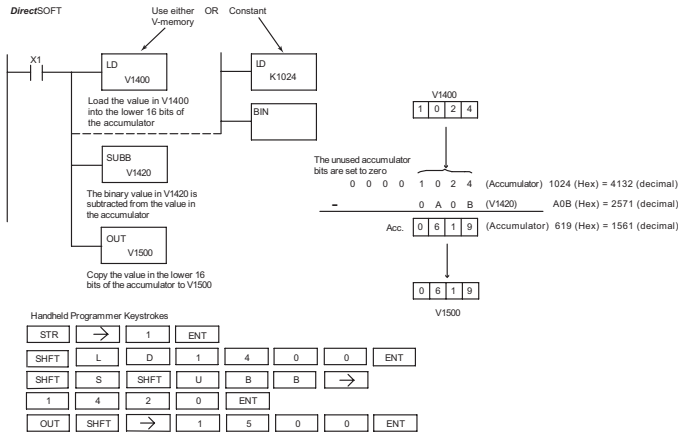
Operand Data Type	Range	
	D2-250-1	D2-260/D2-262
A	aaa	aaa
V-memory	V	All (See page 3-57)
Pointer	P	All V-memory (See page 3-57)
Constant	K	0-FFFF

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



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

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



Subtract Binary Double (SUBBD)

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

SUBBD
A aaa

- ✗

230
- ✗

240
- ✗

250-1
- ✓

260
- ✓

262

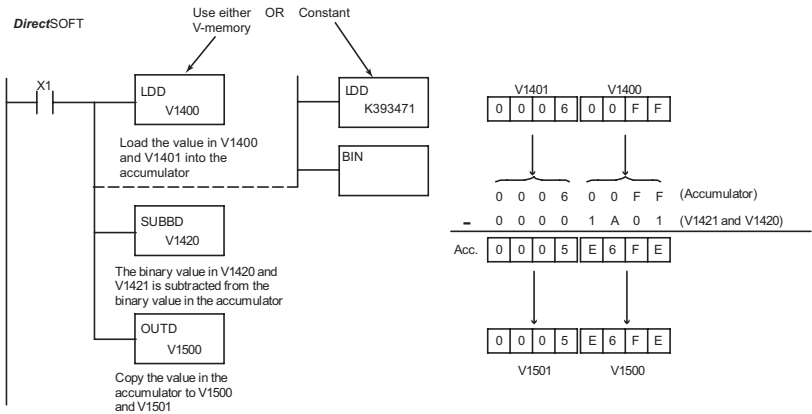
DS	Used
HPP	Used

Operand Data Type	Range D2-260/D2-262
A	aaa
V-memory	V All (See page 3-57)
Pointer	P All V-memory (See page 3-57)
Constant	K 0-FFFFFF

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

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

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



Handheld Programmer Keystrokes

STR	→	1	ENT
SHFT	L	D	D → 1 4 0 0 ENT
SHFT	S	SHFT U	B B D →
1	4	2	0 ENT
OUT	SHFT D	→	1 5 0 0 ENT

Multiply Binary (MULB)

230

240

250-1

260

262

DS	Used
HPP	Used

The Multiply Binary instruction multiplies a 16-bit number A(aaa) by the value stored in the accumulator. The number in the accumulator can be up to 32 bits long. The source of the 16-bit operand can be a constant or a data value located in V-memory. Multiply Binary performs the multiplication operation on the full binary representation of the operands, which distinguishes it from the Multiply instruction (see page 5-94), which treats the operands as BCD numbers. Although the multiplication operation is performed on the underlying binary values, the native display format is hexadecimal. For that reason, you will need to load constants in hex.

MULB

A aaa

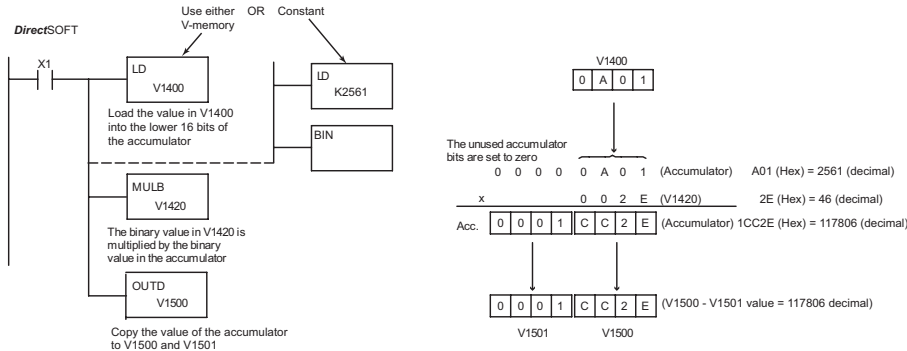
The product of the Multiply Binary operation occupies the full 32-bit accumulator and requires an Out Double to move the product to V-memory. If the value in the accumulator occupies fewer than 32 bits, leading zeros are loaded in the left-most empty bit positions.

Operand Data Type	Range	
	D2-250-1	D2-260/D2-262
A	aaa	aaa
V-memory	V	All (See page 3-56)
Pointer	P	All V-memory (See page 3-56)
Constant	K	0-FFFF

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

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

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



Handheld Programmer Keystrokes

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

Divide Binary (DIVB)

- 230
- 240
- 250-1
- 260
- 262

DS	Used
HPP	Used

The Divide Binary instruction divides a 16-bit number (Aaaa) into the value stored in the accumulator. The number in the accumulator can be up to 32 bits long. The source of the 16-bit divisor can be a constant or a data value located in V-memory. Divide Binary performs the division operation on the full binary representation of the operands, which distinguishes it from the Divide instruction (see page 5-97), which treats the operands as BCD numbers. Although the division operation is performed on the underlying binary values, the native display format is hexadecimal. For that reason you will need to load constants in hex.

DIVB
A aaa

At the completion of the division operation, the quotient resides in the accumulator and the remainder resides in the first stack location.

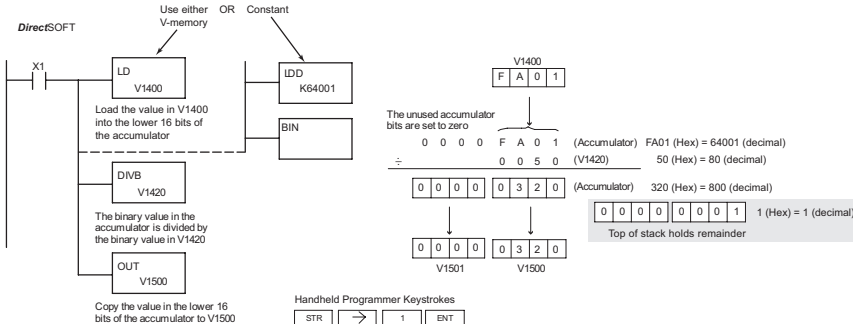
The quotient occupies the full 32-bit accumulator and requires an Out Double to move the quotient to V-memory. If the value in the accumulator occupies fewer than 32 bits, leading zeros are loaded in the left-most empty bit positions.

Operand Data Type		Range	
		D2-250-1	D2-260/D2-262
	A	aaa	aaa
V-memory	V	All (See page 3-56)	All (See page 3-57)
Pointer	P	All V-memory (See page 3-56)	All V-memory (See page 3-57)
Constant	K	0-FFFF	0-FFFF

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

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

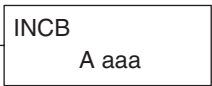
In the following example, when X1 is on, the value in V1400 will be loaded into the accumulator using the Load instruction. The binary value in the accumulator is divided by the binary value in V1420 using the Divide Binary instruction. The value in the accumulator is copied to V1500 using the Out Double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



Increment Binary (INCB)

- 230
- 240
- 250-1
- 260
- 262

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



DS	Used
HPP	Used

Operand Data Type	Range			
	D2-230	D2-240	D2-250-1	D2-260/D2-262
A	aaa	aaa	aaa	aaa
V-memory V	All (See page 3-54)	All (See page 3-55)	All (See page 3-56)	All (See page 3-57)
Pointer P	-	All V-memory (See page 3-55)	All V-memory (See page 3-56)	All V-memory (See page 3-57)

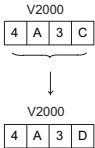
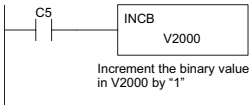
Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero



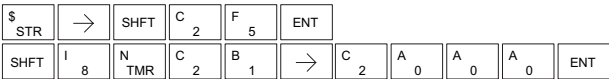
NOTE: The status flags are only valid until another instruction that uses the same flags is executed.

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

DirectSOFT



Handheld Programmer Keystrokes



Decrement Binary (DECB)

- 230
- 240
- 250-1
- 260
- 262

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

DECB
A aaa

DS	Used
HPP	Used

Operand Data Type		Range			
		D2-230	D2-240	D2-250-1	D2-260/D2-262
A		aaa	aaa	aaa	aaa
V-memory	V	All (See page 3-54)	All (See page 3-55)	All (See page 3-56)	All (See page 3-57)
Pointer	P	-	All V-memory (See page 3-55)	All V-memory (See page 3-56)	All V-memory (See page 3-57)

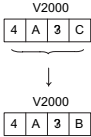
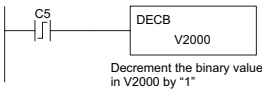
Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero



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.

DirectSOFT



Handheld Programmer Keystrokes

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

Add Formatted (ADDF)

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



- ☐ 230
- ☐ 240
- ☐ 250-1
- ☒ 260
- ☒ 262

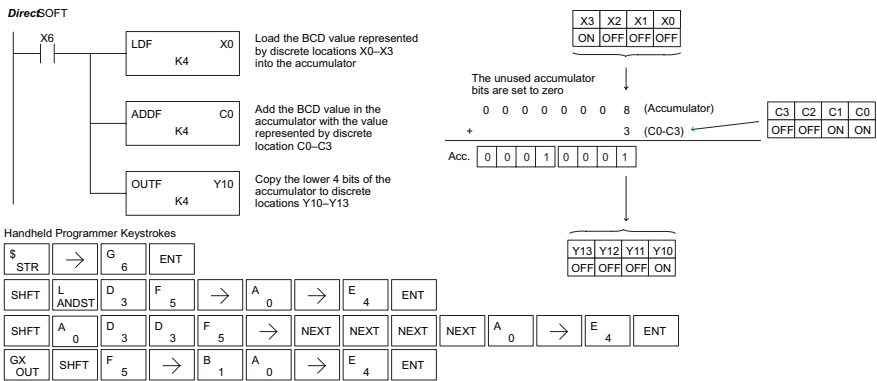
DS	Used
HPP	Used

Operand Data Type		D2-260/D2-262 Range	
	A	aaa	bbb
Inputs	X	0-1777	-
Outputs	Y	0-1777	-
Control Relays	C	0-3777	-
Stage bits	S	0-1777	-
Timer bits	T	0-377	-
Counter bits	CT	0-377	-
Special Relay	SP	0-777	-
Global I/O	GX/ GY	0-3777	-
Constant	K		1-32

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

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

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



Subtract Formatted (SUBF)

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

SUBF A aaa
 K bbb

- 230
- 240
- 250-1
- 260
- 262

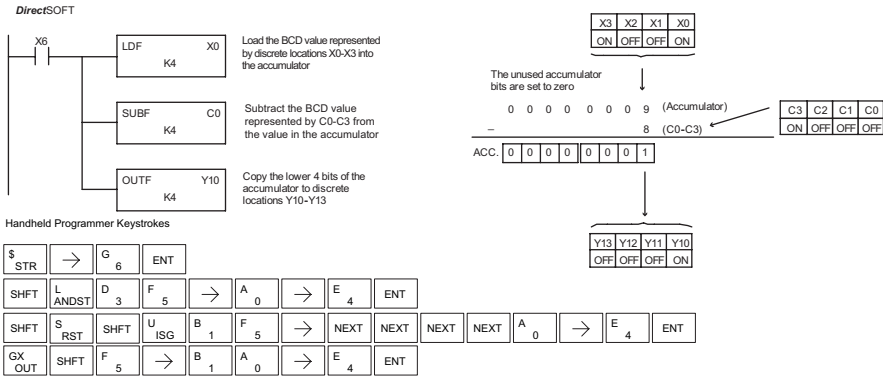
DS	Used
HPP	Used

Operand Data Type	D2-260/D2-262 Range	
	A	aaa bbb
Inputs	X	0-1777 -
Outputs	Y	0-1777 -
Control Relays	C	0-3777 -
Stage bits	S	0-1777 -
Timer bits	T	0-377 -
Counter bits	CT	0-377 -
Special Relay	SP	0-777 -
Global I/O	GX/ GY	0-3777 -
Constant	K	1-32

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

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

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



Multiply Formatted (MULF)

- 230
- 240
- 250-1
- 260
- 262

DS	Used
HPP	Used

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

MULF A aaa
 K bbb

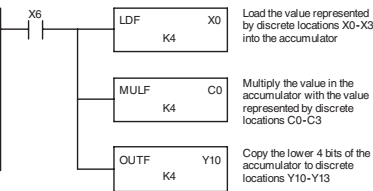
Operand Data Type	D2-260/D2-262 Range	
	A aaa	bbb
Inputs	X 0-1777	-
Outputs	Y 0-1777	-
Control Relays	C 0-3777	-
Stage bits	S 0-1777	-
Timer bits	T 0-377	-
Counter bits	CT 0-377	-
Special Relay	SP 0-777	-
Global I/O	GX/ GY 0-3777	-
Constant	K	1-16

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

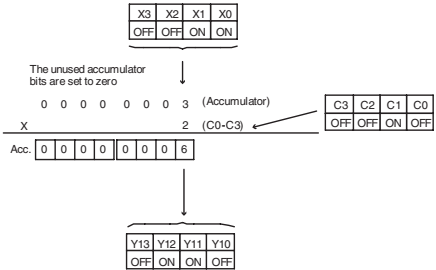
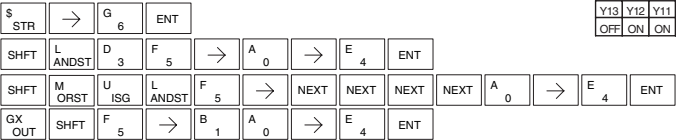
NOTE: Status flags are valid only until another instruction uses the same flag.

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

DirectSOFT



Handheld Programmer Keystrokes



Divide Formatted (DIVF)

- 230
- 240
- 250-1
- 260
- 262

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

DIVF A aaa
 K bbb

DS	Used
HPP	Used

Operand Data Type		D2-260/D2-262 Range	
	A	aaa	bbb
Inputs	X	0-1777	-
Outputs	Y	0-1777	-
Control Relays	C	0-3777	-
Stage bits	S	0-1777	-
Timer bits	T	0-377	-
Counter bits	CT	0-377	-
Special Relay	SP	0-777	-
Global I/O	GX/ GY	0-3777	-
Constant	K		1-16

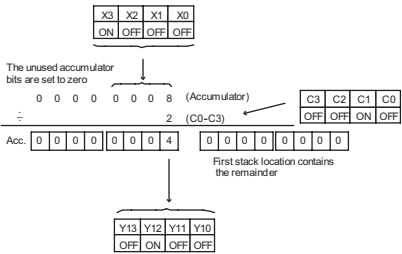
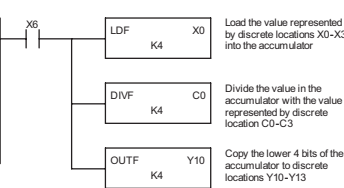
Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero
SP70	On anytime the value in the accumulator is negative
SP75	On when a BCD instruction is executed and a NON-BCD number was encountered



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

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

DirectSOFT



Handheld Programmer Keystrokes

\$	STR	→	G	6	ENT														OFF	ON			
SHFT	L	ANDST	D	3	F	5	→	A	0	→	E	4	ENT										
SHFT	D	3	I	8	V	AND	F	5	→	NEXT	NEXT	NEXT	NEXT	A	0	→	E	4	ENT				
GX	OUT	SHFT	F	5	→	B	1	A	0	→	E	4	ENT										

Add Top of Stack (ADDS)

- ✗

230
- ✗

240
- ✗

250-1
- ✓

260
- ✓

262
- Add Top of Stack is a 32-bit instruction that adds the BCD value in the accumulator with the BCD value in the first level of the accumulator stack. The result resides in the accumulator. The value in the first level of the accumulator stack is removed and all stack values are moved up one level.

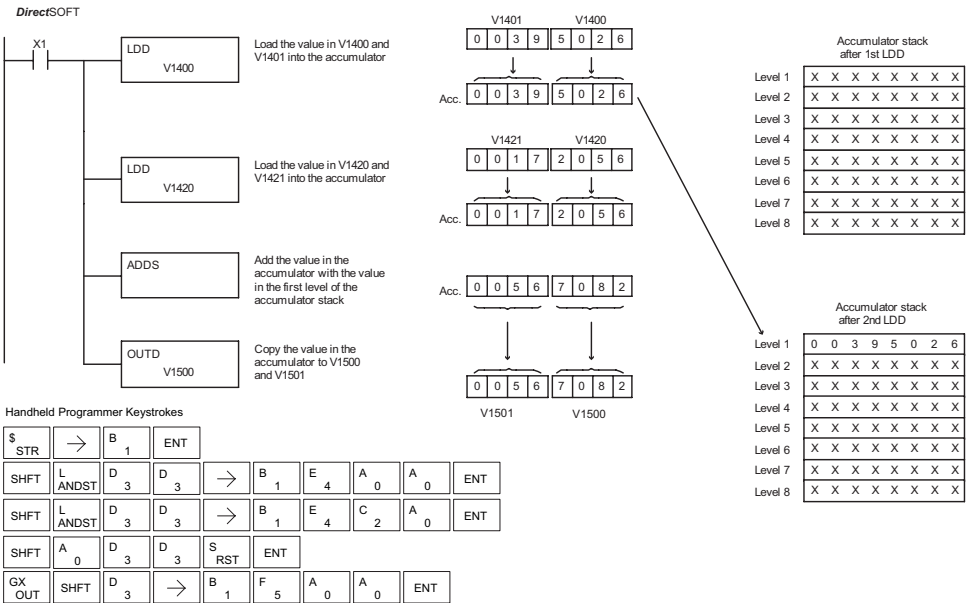
ADDS

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



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

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



Subtract Top of Stack (SUBS)

-  230
-  240
-  250-1
-  260
-  262

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

SUBS

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

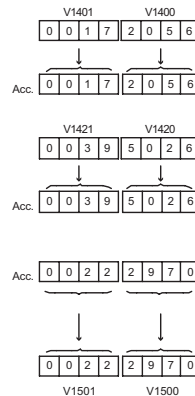
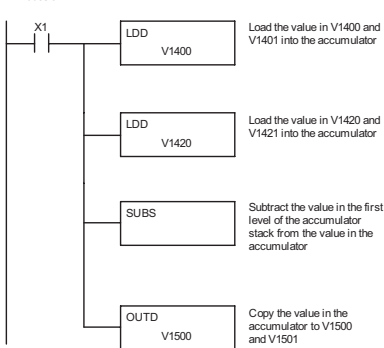


NOTE: Status flags are valid only until another instruction uses the same flag.

DS	Used
HPP	Used

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

DirectSOFT



Accumulator stack after 1st LDD

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

Accumulator stack after 2nd LDD

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

Handheld Programmer Keystrokes

\$ STR	→	B 1	ENT										
SHFT	L ANDST	D 3	D 3	→	B 1	E 4	A 0	A 0	ENT				
SHFT	L ANDST	D 3	D 3	→	B 1	E 4	C 2	A 0	ENT				
SHFT	S RST	SHFT	U ISG	B 1	S RST	ENT							
GX OUT	SHFT	D 3	→	B 1	F 5	A 0	A 0	ENT					

Multiply Top of Stack (MULS)

- 230
- 240
- 250-1
- 260
- 262

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

MULS

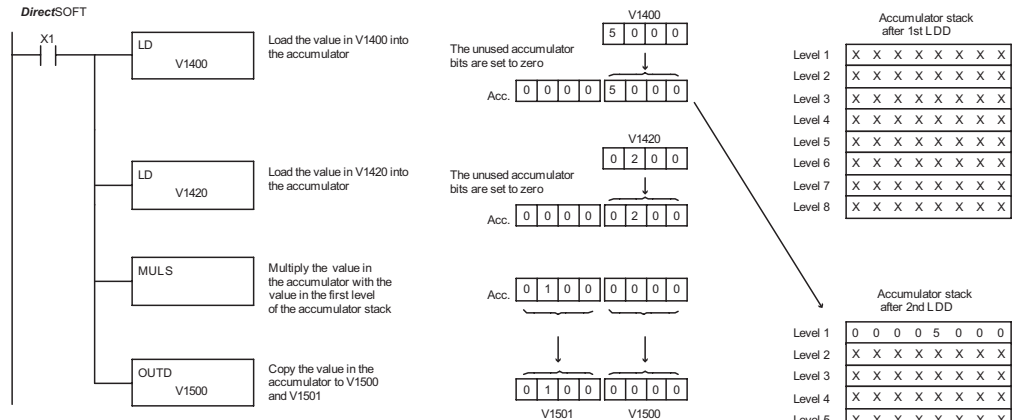
Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero
SP70	On anytime the value in the accumulator is negative
SP75	On when a BCD instruction is executed and a NON-BCD number was encountered



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

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






DS	Used
HPP	Used



Handheld Programmer Keystrokes

\$	STR	→	B	1	ENT														
SHFT	L	ANDST	D	3	→	B	1	E	4	A	0	A	0	ENT					
SHFT	L	ANDST	D	3	→	B	1	E	4	C	2	A	0	ENT					
SHFT	M	ORST	U	ISG	L	ANDST	S	RST	ENT										
GX	OUT	SHFT	D	3	→	B	1	F	5	A	0	A	0	ENT					

Divide by Top of Stack (DIVS)

-  230 Divide Top of Stack is a 32-bit instruction that divides the
-  240 8-digit BCD value in the accumulator by a 4-digit BCD
-  250-1 value in the first level of the accumulator stack. The result
-  260 resides in the accumulator and the remainder resides in
-  262 the first level of the accumulator stack.

DIVS

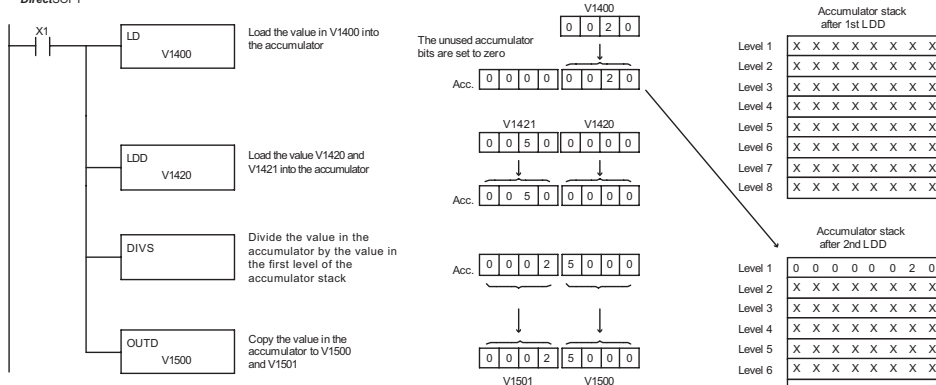
Discrete Bit Flags	Description
SP53	On when the value of the operand is larger than the accumulator can work with
SP63	On when the result of the instruction causes the value in the accumulator to be zero
SP70	On anytime the value in the accumulator is negative.
SP75	On when a BCD instruction is executed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



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

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

DirectSOFT



Handheld Programmer Keystrokes

\$	STR	→	B	1	ENT
SHFT	L	ANDST	D	3	→
			B	1	E 4 A 0 A 0 ENT
SHFT	L	ANDST	D	3	D 3 →
			B	1	E 4 C 2 A 0 ENT
SHFT	D	3	I	8	V AND S RST ENT
GX	OUT	SHFT	D	3	→
			B	1	F 5 A 0 A 0 ENT

Add Binary Top of Stack (ADDBS)

- 230
- 240
- 250-1
- 260
- 262

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

ADDBS

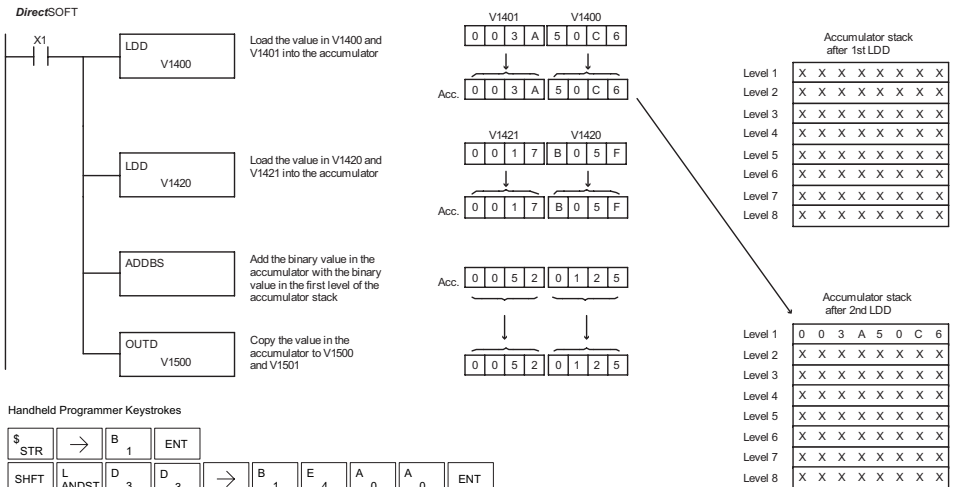
Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero
SP66	On when the 16-bit addition instruction results in a carry
SP67	On when the 32-bit addition instruction results in a carry
SP70	On anytime the value in the accumulator is negative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



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

DS	Used
HPP	Used

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



Subtract Binary Top of Stack (SUBBS)

- ✗

230
- ✗

240
- ✗

250-1
- ✓

260
- ✓

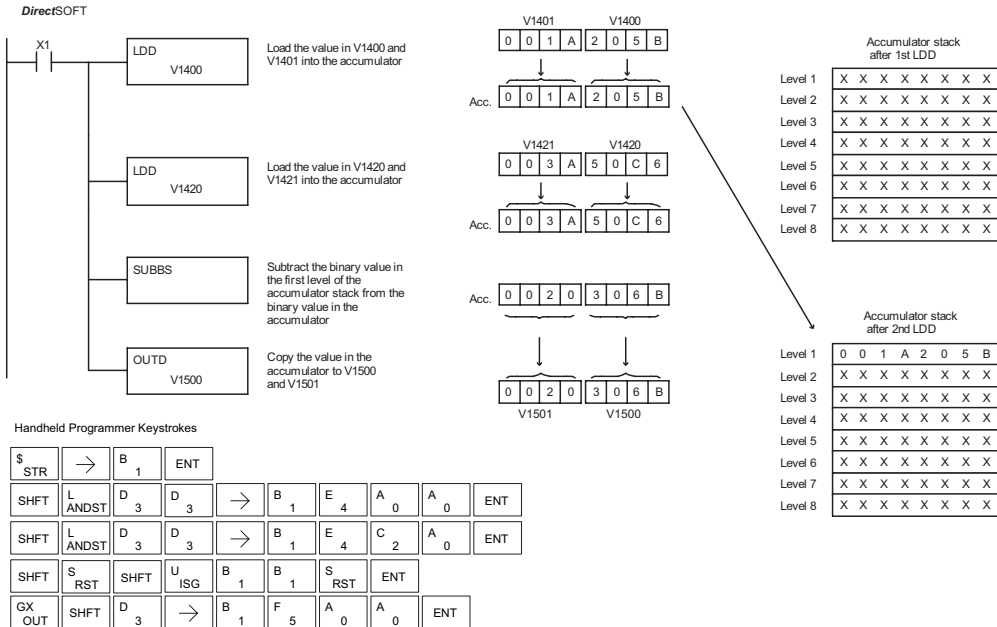
262
- Subtract Binary Top of Stack is a 32-bit instruction that subtracts the binary value in the first level of the accumulator stack from the binary value in the accumulator. The result resides in the accumulator. The value in the first level of the accumulator stack is removed, and all stack locations are moved up one level.

SUBBS

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

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

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



Multiply Binary Top of Stack (MULBS)

- 230
- 240
- 250-1
- 260
- 262

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

MULBS

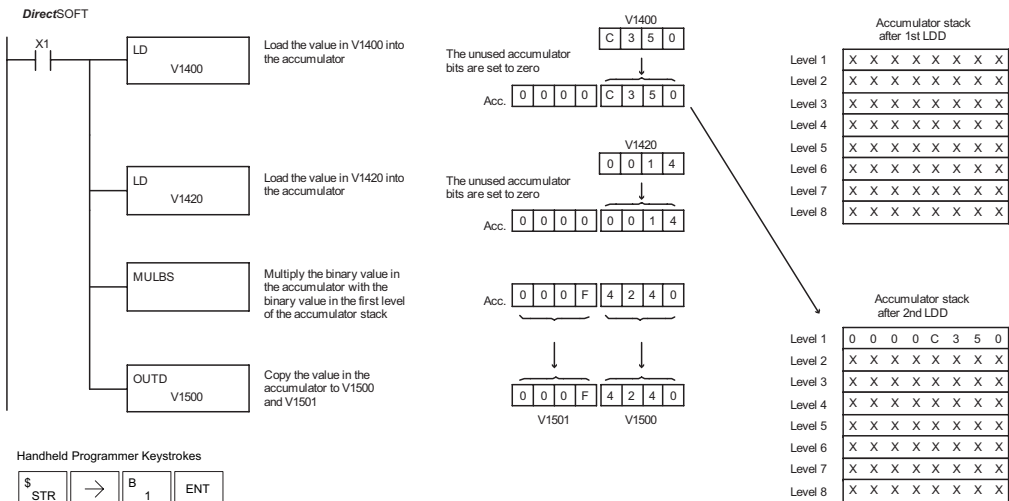
Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero
SP70	On anytime the value in the accumulator is negative



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

DS	Used
HPP	Used

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



Handheld Programmer Keystrokes

\$	STR	→	B	1	ENT									
SHFT	L	ANDST	D	3	→	B	1	E	4	A	0	A	0	ENT
SHFT	L	ANDST	D	3	→	B	1	E	4	C	2	A	0	ENT
SHFT	M	ORST	U	ISG	L	ANDST	B	1	S	RST	ENT			
GX	OUT	SHFT	D	3	→	B	1	F	5	A	0	A	0	ENT

Divide Binary by Top of Stack (DIVBS)

- ✗

230
- ✗

240
- ✗

250-1
- ✓

260
- ✓

262

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

DIVBS

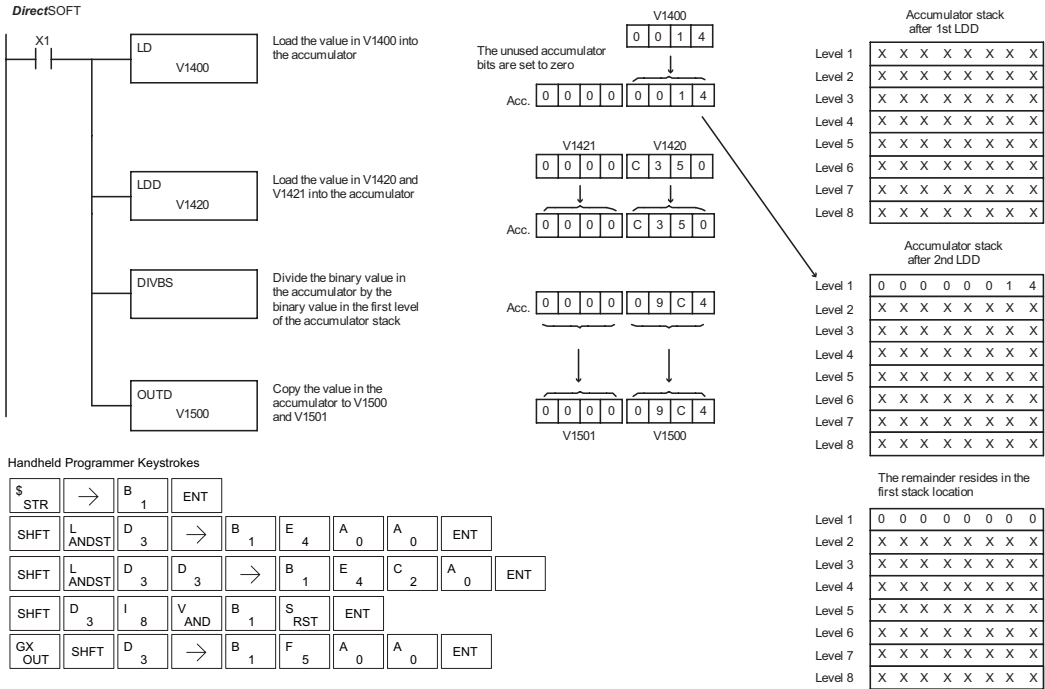
Discrete Bit Flags	Description
SP53	On when the value of the operand is larger than the accumulator can work with
SP63	On when the result of the instruction causes the value in the accumulator to be zero
SP70	On anytime the value in the accumulator is negative




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

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

DS	Used
HPP	Used



Transcendental Functions (D2-260 and D2-262 only)

 230 The D2-260 and D2-262 CPUs feature special numerical functions to complement the real number capability. The transcendental functions include the trigonometric sine, cosine, and tangent, and also their inverses (arc sine, arc cosine, and arc tangent). The square root function is also grouped with these other functions.

 240

 250-1

 260

 262

DS	Used
HPP	N/A

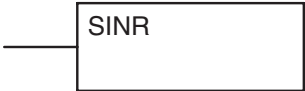
The transcendental math instructions operate on a real number in the accumulator (it cannot be BCD or binary). The real number result resides in the accumulator. The square root function operates on the full range of positive real numbers. The sine, cosine and tangent functions require numbers expressed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

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

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

Sine Real (SINR)

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



Cosine Real (COSR)

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



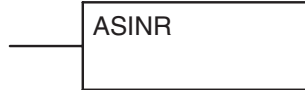
Tangent Real (TANR)

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



Arc Sine Real (ASINR)

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



Arc Cosine Real (ACOSR)

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

ACOSR

Arc Tangent Real (ATANR)

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

ATANR

Square Root Real (SQRTR)

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

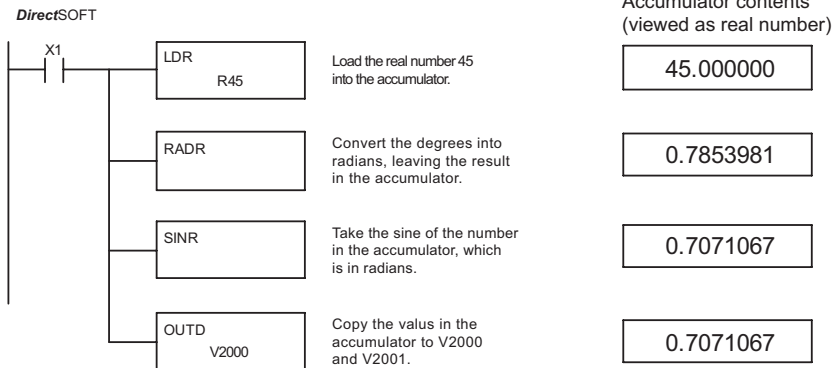
SQRTR



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

DS	Used
HPP	N/A

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



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

The `Count` instruction counts the number of bits that are

[illegible]

Math Function	Range of Argument
\sin^{-1}	$[-\frac{\pi}{2}, \frac{\pi}{2}]$
\cos^{-1}	$[0, \pi]$
\tan^{-1}	$(-\frac{\pi}{2}, \frac{\pi}{2})$
\cot^{-1}	$(0, \pi)$
\sec^{-1}	$[0, \pi] \setminus \{\frac{\pi}{2}\}$
\csc^{-1}	$(-\frac{\pi}{2}, \frac{\pi}{2}) \setminus \{0\}$

SP63	On when the result of the instruction causes the value in the accumulator to be zero
------	--

--	--

--	--

[illegible]

Shift Left (SHFL)

- 230
- 240
- 250-1
- 260
- 262

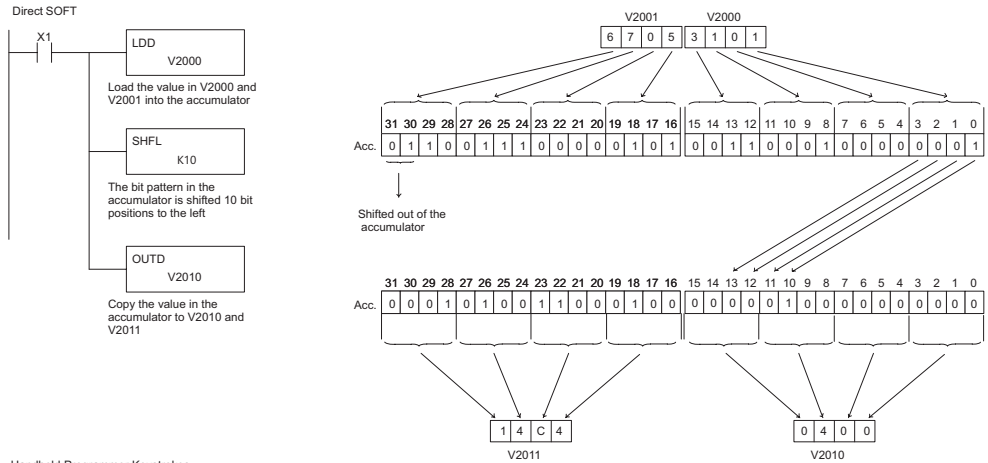
Shift Left is a 32-bit instruction that shifts the bits in the accumulator a specified number (Aaaa) of places to the left. The vacant positions are filled with zeros, and the bits shifted out of the accumulator are lost.

SHFL
A aaa

Operand Data Type	Range			
	D2-230	D2-240	D2-250-1	D2-260/D2-262
A	aaa	aaa	aaa	aaa
V-memory	V	All (See page 3-54)	All (See page 3-55)	All (See page 3-57)
Constant	K	1-32	1-32	1-32

In the following example, when X1 is on, the value in V2000 and V2001 will be loaded into the accumulator using the Load Double instruction. The bit pattern in the accumulator is shifted 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.

DS	Used
HPP	Used



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		

Rotate Left (ROTL)

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

ROTL

A aaa

☐ 230

☐ 240

☒ 250-1

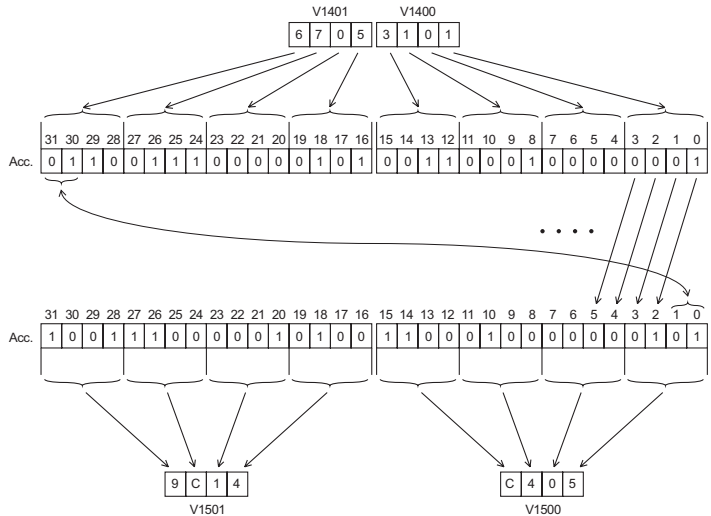
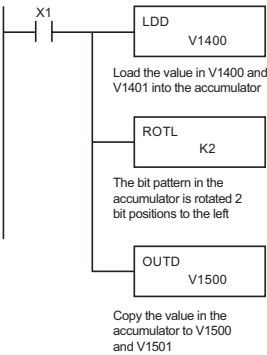
☒ 260

☒ 262

Operand Data Type		Range	
		D2-250-1	D2-260/D2-262
A		aaa	aaa
V-memory	V	All (See page 3-56)	All (See page 3-57)
Constant	K	1-32	1-32

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

DS	Used
HPP	Used



Handheld Programmer Keystrokes

\$ STR	→	B ₁	ENT							
SHFT	L ANDST	D ₃	D ₃	→	B ₁	E ₄	A ₀	A ₀	ENT	
SHFT	R ORN	O INST#	T MLR	L ANDST	→	C ₂	ENT			
GX OUT	SHFT	D ₃	→	B ₁	F ₅	A ₀	A ₀	ENT		

Rotate Right (ROTR)

- 230
- 240
- 250-1
- 260
- 262

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

ROTR

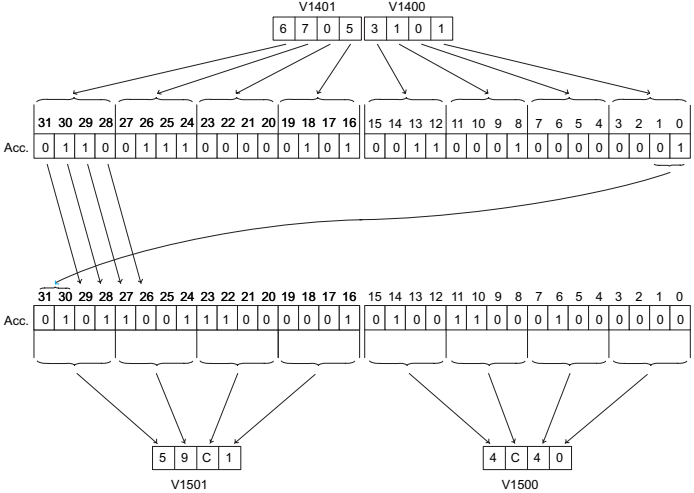
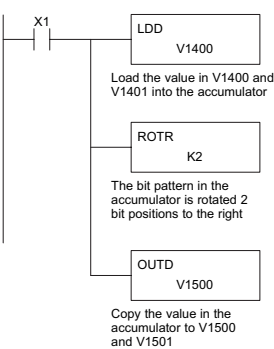
A aaa

Operand Data Type	Range	
	D2-250-1	D2-260/D2-262
A	aaa	aaa
V-memory	V	All (See page 3-56)
Constant	K	1-32

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

DS	Used
HPP	Used

DirectSOFT



Handheld Programmer Keystrokes

\$	STR	→	B	1	ENT
SHFT	L	ANDST	D	3	→
SHFT	R	ORN	O	INST#	T
GX	OUT	SHFT	D	3	→
			B	1	F
			A	0	A
			A	0	ENT

Encode (ENCO)

- 230
- 240
- 250-1
- 260
- 262

DS	Used
HPP	Used

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

ENCO

Discrete Bit Flags

Description

SP53

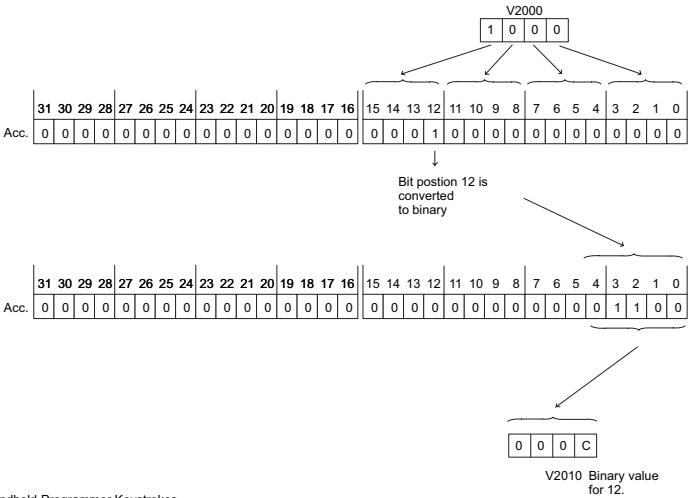
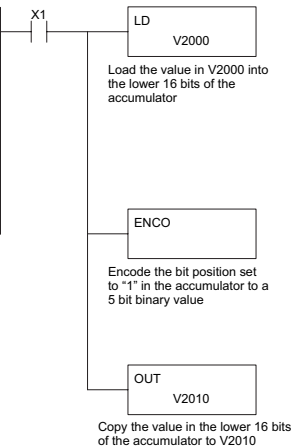
On when the value of the operand is larger than the accumulator can work with



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.

DirectSOFT



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)

- 230
- 240
- 250-1
- 260
- 262

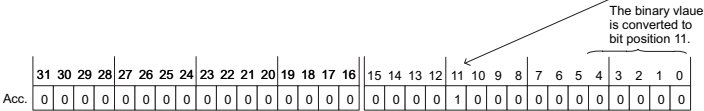
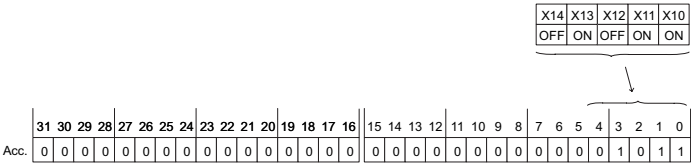
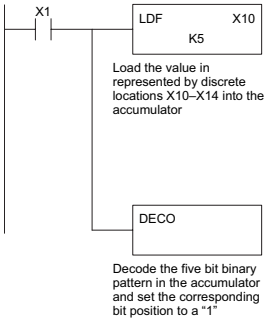
The Decode instruction decodes a 5-bit binary value of 0 to 31 (0 to 1F HEX) in the accumulator by setting the appropriate bit position to a 1. If the accumulator contains the value F (HEX), bit 15 will be set in the accumulator. If the value to be decoded is greater than 31, the number is divided by 32 until the value is less than 32 and then the value is decoded.

DECO

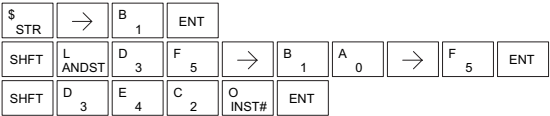
DS	Used
HPP	Used

In the following example, when X1 is on, the value formed by discrete locations X10–X14 is loaded into the accumulator using the Load Formatted instruction. The 5-bit binary pattern in the accumulator is decoded by setting the corresponding bit position to a “1” using the Decode instruction.

DirectSOFT



Handheld Programmer Keystrokes



Number Conversion Instructions (Accumulator)

Binary (BIN)

- ✓ 230
- ✓ 240
- ✓ 250-1
- ✓ 260
- ✓ 262

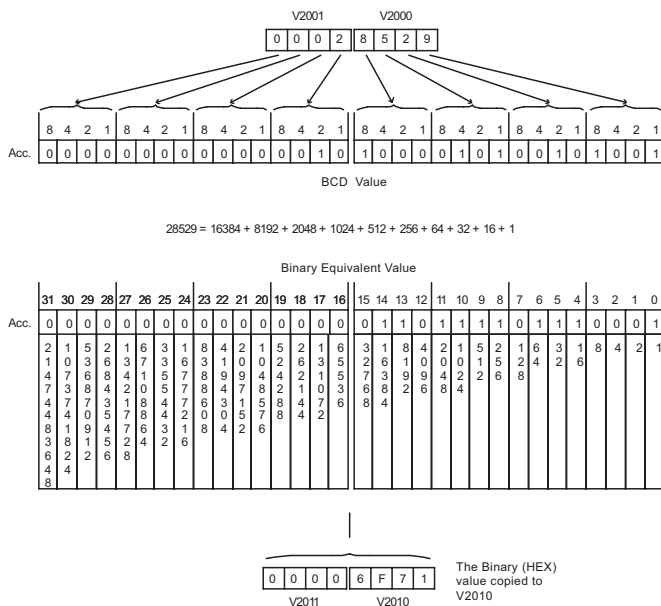
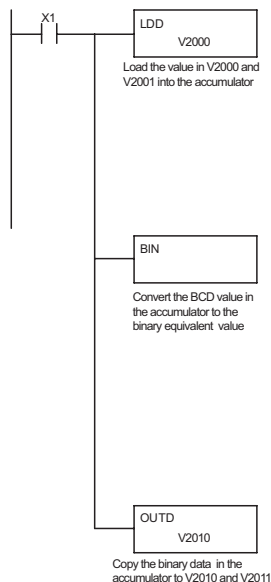
DS	Used
HPP	Used

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

BIN

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

DirectSOFT



Handheld Programmer Keystrokes

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

Binary Coded Decimal (BCD)

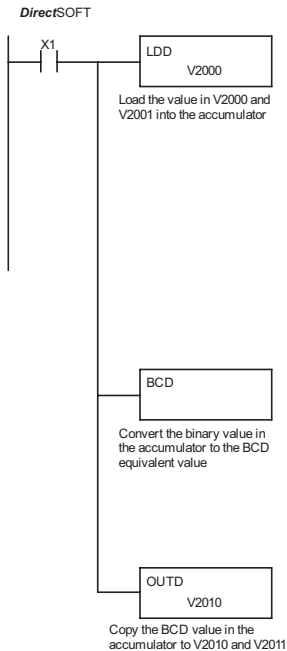
- 230
- 240
- 250-1
- 260
- 262

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

BCD

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.

DS	Used
HPP	Used



V2001																V2000															
0 0 0 0																6 F 7 1															
Binary Value																															
<div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><</div>																															

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

BCD Equivalent Value

Acc.																															
8	4	2	1	8	4	2	1	8	4	2	1	8	4	2	1	8	4	2	1	8	4	2	1	8	4	2	1	8	4	2	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	1	0	1	1	0	0	1	0	1	0	0	1

V2011				V2010				The BCD value copied to V2010 and V2011			
0 0 0 2				8 5 2 9							

Handheld Programmer Keystrokes

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

Invert (INV)

- 230
- 240
- 250-1
- 260
- 262

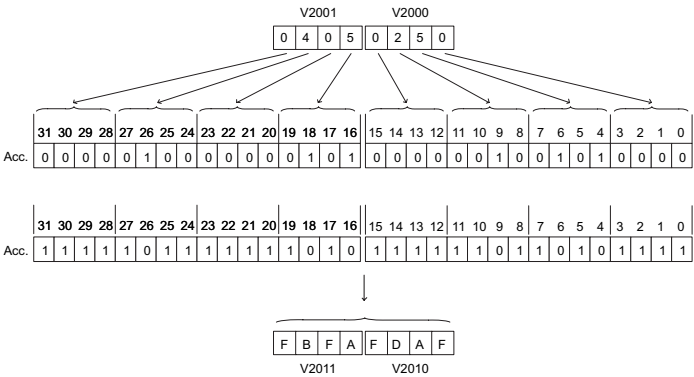
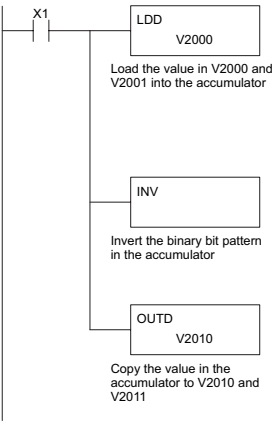
DS	Used
HPP	Used

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

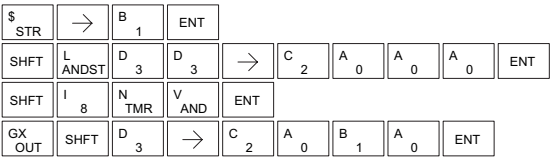
INV

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.

DirectSOFT



Handheld Programmer Keystrokes



Ten's Complement (BCDCPL)

- 230
- 240
- 250-1
- 260
- 262

The Ten's Complement instruction takes the 10's complement (BCD) of the 8-digit accumulator. The result resides in the accumulator. The calculation for this instruction is :

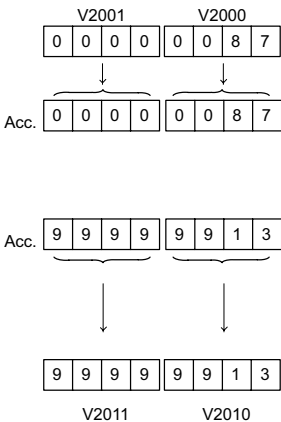
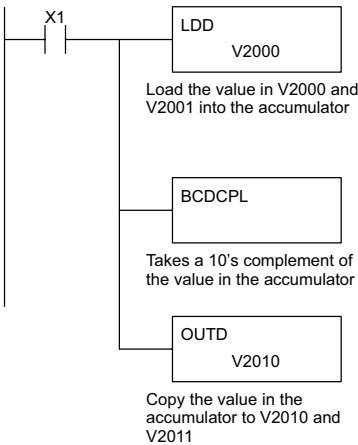
BCDCPL

100000000
— accumulator value
10's complement value

DS	Used
HPP	Used

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

DirectSOFT



Handheld Programmer Keystrokes

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

Binary to Real Conversion (BTOR)

- 230
- 240
- 250-1
- 260
- 262

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

BTOR

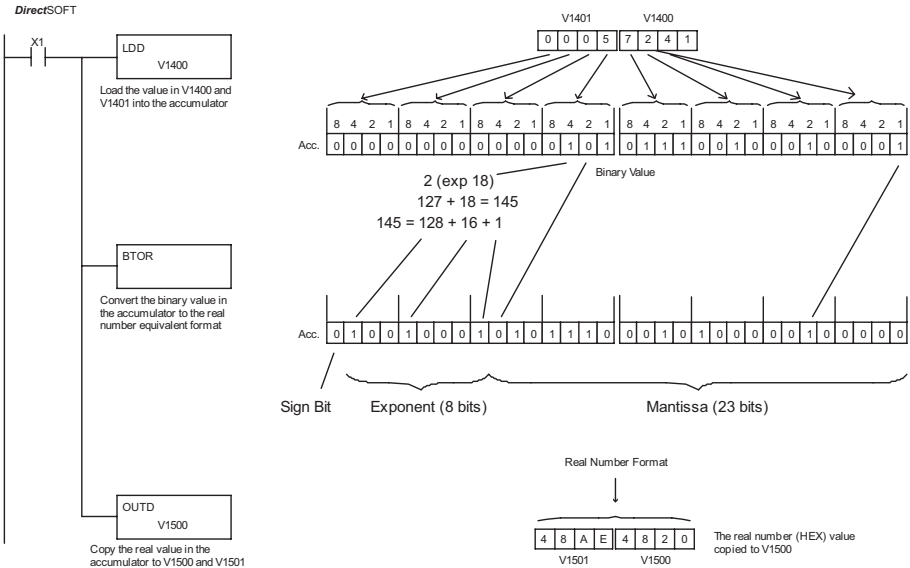


NOTE: This instruction only works with unsigned binary, or decimal values. It will not work with signed decimal values.

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

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

DS	Used
HPP	Used



Handheld Programmer Keystrokes

\$	STR	→	B	1	ENT									
SHFT	L	ANDST	D	3	→	B	1	E	4	A	0	A	0	ENT
SHFT	B	1	T	MLR	O	INST#	R	ORN	ENT					
GX	OUT	SHFT	D	3	→	B	1	F	5	A	0	A	0	ENT

Real to Binary Conversion (RTOB)

- ✗

230
- ✗

240
- ✓

250-1
- ✓

260
- ✓

262

RTOB

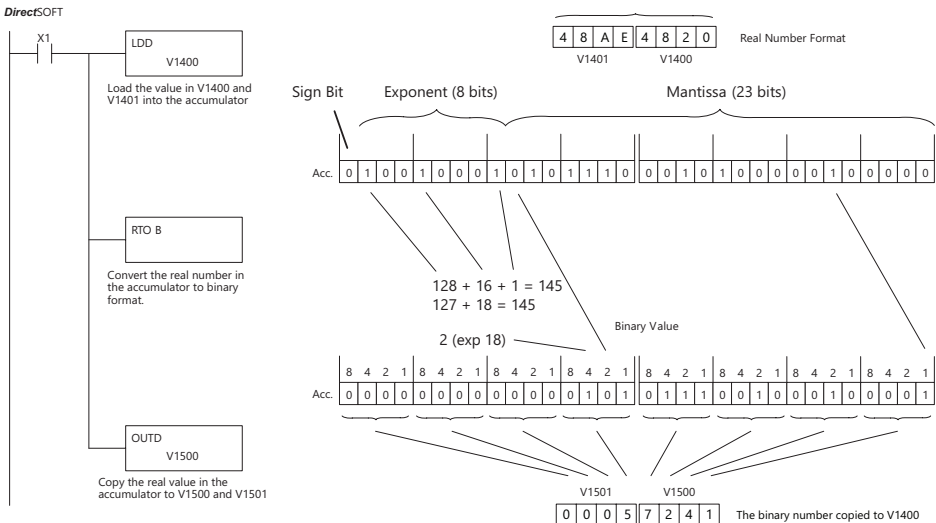


NOTE 1: The decimal portion of the result will be rounded down (14.1 to 14 or -14.1 to -15).
NOTE 2: If the real number is negative, it becomes a signed decimal value.

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero
SP70	On anytime the value in the accumulator is negative
SP72	On anytime the value in the accumulator is a valid floating point number
SP73	On when a signed addition or subtraction results in an incorrect sign bit.
SP75	On when a number cannot be converted to binary

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

DS	Used
HPP	Used



Handheld Programmer Keystrokes

\$ STR	→	B 1	ENT
SHFT	L ANDST	D 3	D 3
SHFT	R ORN	T MLR	O INST#
GX OUT	SHFT	D 3	→
		B 1	F 5
		A 0	A 0
			ENT

Radian Real Conversion (RADR)

- 230 The Radian Real Conversion instruction converts the real degree value stored in the accumulator to the equivalent real number in radians. The result resides in the accumulator.
- 240
- 250-1

RADR

Degree Real Conversion (DEGR)

- 260 The Degree Real instruction converts the degree real radian value stored in the accumulator to the equivalent real number in degrees. The result resides in the accumulator.
- 262
- 230 The two instructions described above convert real numbers in the accumulator from degree format to radian format, and visa-versa. In degree format, a circle contains 360 degrees. In radian format, a circle contains 2 P. These convert between both positive and negative real numbers, and for angles greater than a full circle. These functions are very useful when combined with the transcendental trigonometric functions (see the section on math instructions).
- 240
- 250-1
- 260
- 262

DEGR

DS	Used
HPP	N/A

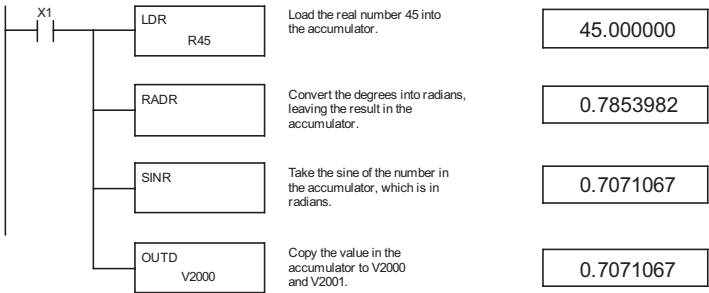
Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero
SP70	On anytime the value in the accumulator is negative
SP71	On anytime the V-memory specified by a pointer (P) is not valid
SP72	On anytime the value in the accumulator is a valid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



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

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

DirectSOFT



ASCII to HEX (ATH)

230

240

250-1

260

262

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

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

ATH

Vaaa

DS	Used
HPP	N/A

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

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

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

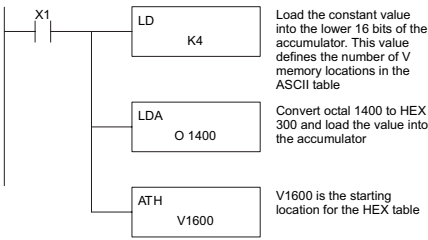
Helpful hint: — For parameters that require HEX values when referencing memory locations, the LDA instruction can be used to convert an octal address to the HEX equivalent and load the value into the accumulator.

Operand Data Type		Range	
		D2-250-1	D2-260/D2-262
		aaa	aaa
V-memory	V	All (See page 3-56)	All (See page 3-57)

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

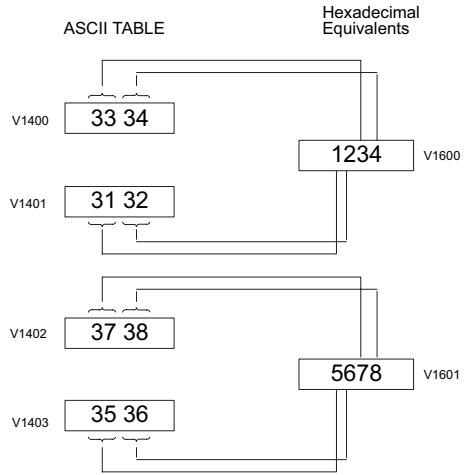
ASCII Values Valid for ATH Conversion			
ASCII	Hex Value	ASCII Value	Hex Value
30	0	38	8
31	1	39	9
32	2	41	A
33	3	42	B
34	4	43	C
35	5	44	D
36	6	45	E
37	7	46	F

DirectSOFT



Handheld Programmer Keystrokes

\$	STR	→	B	1	ENT														
SHFT	L	ANDST	D	3	→	PREV	E	4	ENT										
SHFT	L	ANDST	D	3	A	0	→	B	1	E	4	A	0	A	0	ENT			
SHFT	A	0	T	MLR	H	7	→	B	1	G	6	A	0	A	0	ENT			



HEX to ASCII (HTA)

- 230
- 240
- 250-1
- 260
- 262

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

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



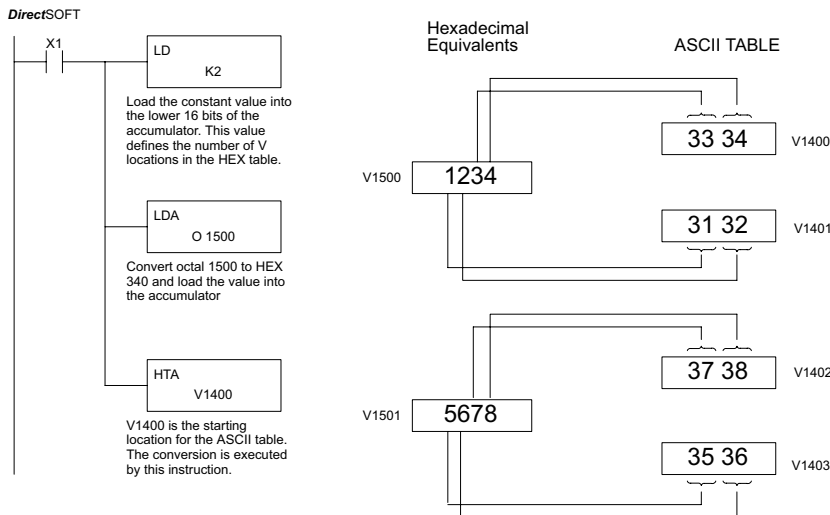
DS	Used
HPP	N/A

- Step 1: Load the number of V-memory locations in the HEX table into the first level of the accumulator stack.
- Step 2: Load the starting V-memory location for the HEX table into the accumulator. This parameter must be a HEX value.
- Step 3: Specify the starting V-memory location (Vaaa) for the ASCII table in the HTA instruction.

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

Operand Data Type	Range	
	D2-250-1	D2-260/D2-262
	aaa	aaa
V-memory	All (See page 3-56)	All (See page 3-57)

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



Handheld Programmer Keystrokes

\$ STR	→	B 1	ENT
SHFT L ANDST	D 3	→	PREV C 2 ENT
SHFT L ANDST	D 3	A 0	→ B 1 F 5 A 0 A 0 ENT
SHFT H 7	T MLR	A 0	→ B 1 E 4 A 0 A 0 ENT

The table below lists valid ASCII values for HTA conversion.

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

Segment (SEG)

Gray Code (GRAY)

- 230
- 240
- 250-1
- 260
- 262

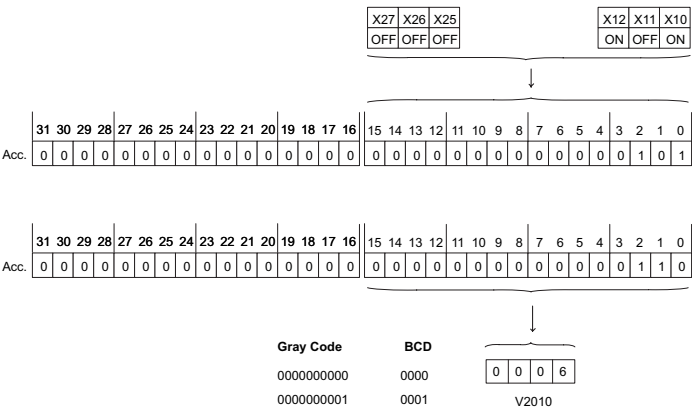
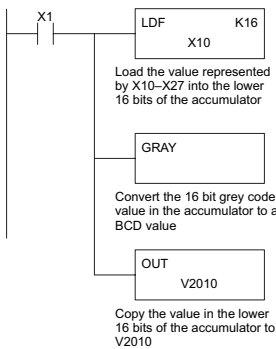
DS	Used
HPP	Used

The Gray code instruction converts a 16-bit gray code value to a BCD value. The BCD conversion requires 10 bits of the accumulator. The upper 22 bits are set to "0." This instruction is designed for use with devices (typically encoders) that use the gray code numbering scheme. The Gray Code instruction will directly convert a gray code number to a BCD number for devices having a resolution of 512 or 1024 counts per revolution. If a device having a resolution of 360 counts per revolution is to be used, you must subtract a BCD value of 76 from the converted value to obtain the proper result. For a device having a resolution of 720 counts per revolution, you must subtract a BCD value of 152.

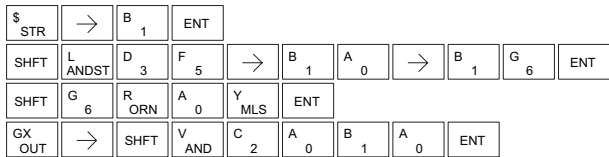
In the following example, when X1 is ON the binary value represented by X10–X27 is loaded into the accumulator using the Load Formatted instruction. The gray code value in the accumulator is converted to BCD using the Gray Code instruction. The value in the lower 16 bits of the accumulator is copied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

GRAY

DirectSOFT



Handheld Programmer Keystrokes



Shuffle Digits (SFLDGT)

- 230
- 240
- 250-1
- 260
- 262

DS	Used
HPP	Used

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

SFLDGT

- Step 1: Load the value (digits) to be shuffled into the first level of the accumulator stack.
- Step 2: Load the order that the digits will be shuffled to into the accumulator.
- Step 3: Insert the SFLDGT instruction.



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

See example on the next page.



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

Shuffle Digits Block Diagram

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

Digits to be shuffled (first stack location)

9	A	B	C	D	E	F	0
---	---	---	---	---	---	---	---



1	2	8	7	3	6	5	4
---	---	---	---	---	---	---	---

Specified order (accumulator)

Bit Positions 8 7 6 5 4 3 2 1

B	C	E	F	0	D	A	9
---	---	---	---	---	---	---	---

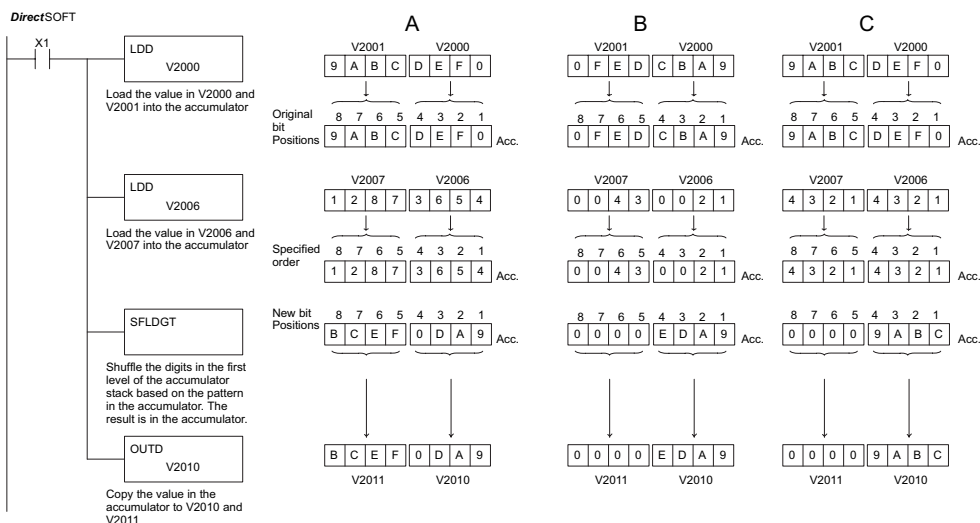
Result (accumulator)

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

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

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

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



Handheld Programmer Keystrokes

[illegible]

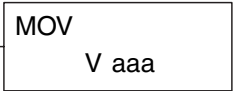
Table Instructions

Move (MOV)

- 230
- 240
- 250-1
- 260
- 262

DS	Used
HPP	Used

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.

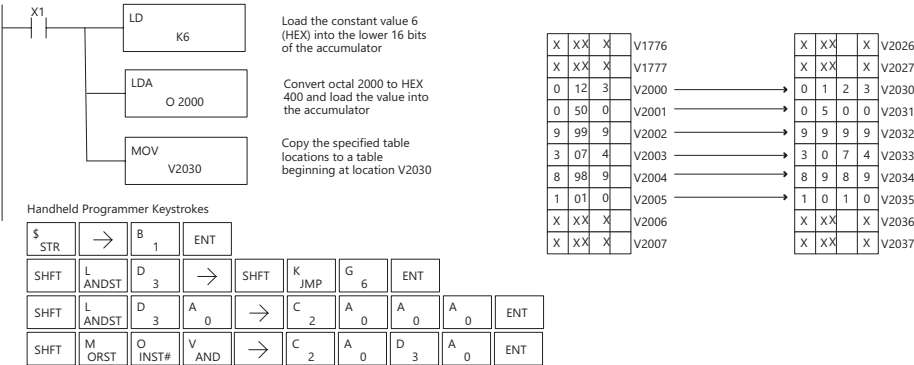


- Step 1: Load the number of V-memory locations to be moved into the first level of the accumulator stack. This parameter is a HEX value (KFFF max, 7777 octal).
- Step 2: Load the starting V-memory location for the locations to be moved into the accumulator. This parameter must be a HEX value.
- Step 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	Range			
	D2-230	D2-240	D2-250-1	D2-260/D2-262
A	aaa	aaa	aaa	aaa
V-memory V	All (See page 3-54)	All (See page 3-55)	All (See page 3-56)	All (See page 3-57)
Pointer P	All (See page 3-54)	All (See page 3-55)	All (See page 3-56)	All (See page 3-57)

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)

- ✓ 230
- ✓ 240
- ✓ 250-1
- ✓ 260
- ✓ 262

DS	Used
HPP	Used

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

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

MOVMC
V aaa

LDLBL
K aaa

Operand Data Type		Range			
		D2-230	D2-240	D2-250-1	D2-260/D2-262
	A	aaa	aaa	aaa	aaa
V-memory	V	All (See page 3-54)	All (See page 3-55)	All (See page 3-56)	All (See page 3-57)
Constant	K	K1-KFFFF	K1-KFFFF	K1-KFFFF	K1-KFFFF



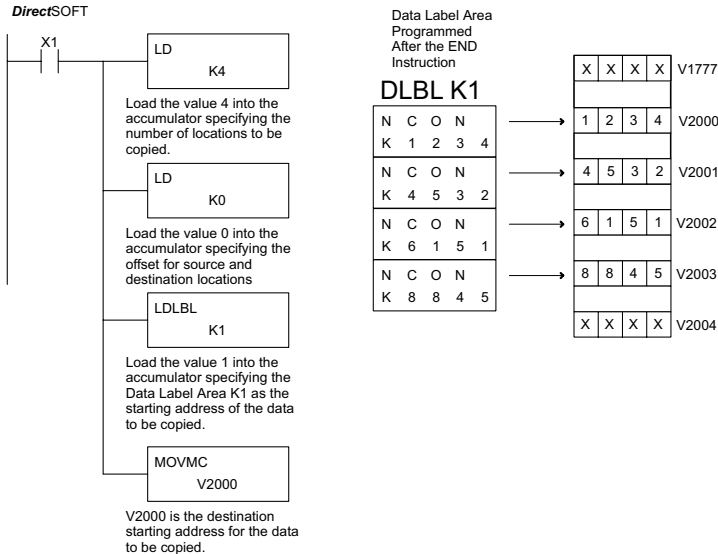
WARNING: The offset for this usage of the instruction starts at 0, but may be any number that does not result in data outside of the source data area being copied into the destination table. When an offset is outside of the source information boundaries, then unknown data values will be transferred into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

Copy Data From a Data Label Area to V-Memory

- 230
- 240
- 250-1
- 260
- 262

DS	Used
HPP	Used

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



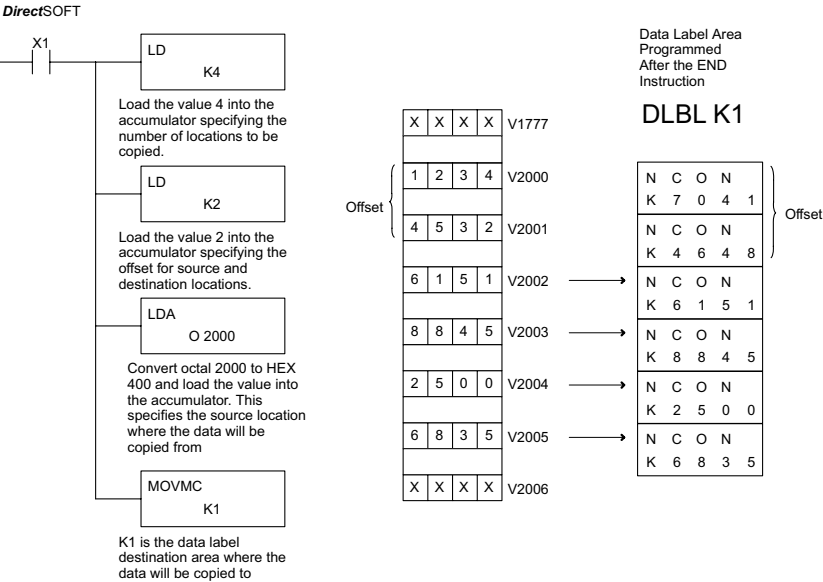
WARNING: The offset for this usage of the instruction starts at 0, but may be any number that does not result in data outside of the source data area being copied into the destination table. When an offset is outside of the source information boundaries, then unknown data values will be transferred into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

Copy Data From V-Memory to a Data Label Area

- 230
- 240
- 250-1
- 260

DS	Used
HPP	Used

In the following example, data is copied from V-memory to a data label area. When X1 is on, the constant value (K4) is loaded into the accumulator using the Load instruction. This value specifies the length of the table and is placed in the second stack location after the next Load and Load Address instructions are executed. The constant value (K2) is loaded into the accumulator using the Load instruction. This value specifies the offset for the source and destination data, and is placed in the first stack location after the Load Address instruction is executed. The source address where data is being copied from is loaded into the accumulator using the Load Address instruction. The MOVMC instruction specifies the destination starting location and executes the copying of data from V-memory to the data label area.




Handheld Programmer Keystrokes

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



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


Set Bit (SETBIT)

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

 240

 250-1

Reset Bit (RSTBIT)

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

 262


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

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

 230

 240

 250-1

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

 262

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

DS	Used
HPP	Used

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

Operand Data Type		Range D2-260/D2-262
		aaa
V-memory	V	All (See page 3-57)

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



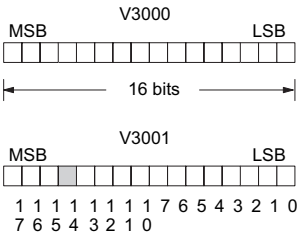
NOTE: Status flags are only valid until the end of the scan or another instruction that uses the same flag is executed.

SETBIT
V aaa

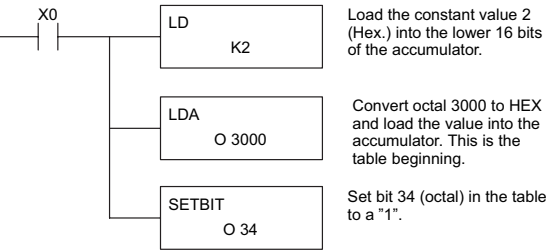
RSTBIT
V aaa

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

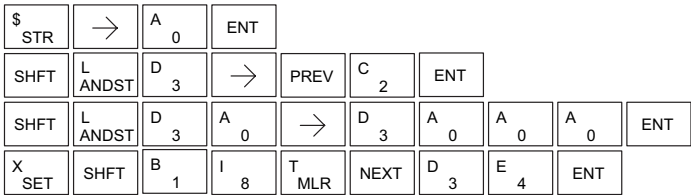
In this ladder example, we will use input X0 to trigger the Set Bit operation. First, we will load the table length (two words) into the accumulator stack. Next, we load the starting address into the accumulator. Since V3000 is an octal number we have to convert it to hex by using the LDA command. Finally,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



DirectSOFT



Handheld Programmer Keystrokes

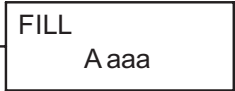


Fill (FILL)

- 230
- 240
- 250-1
- 260
- 262

DS	Used
HPP	Used

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



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

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

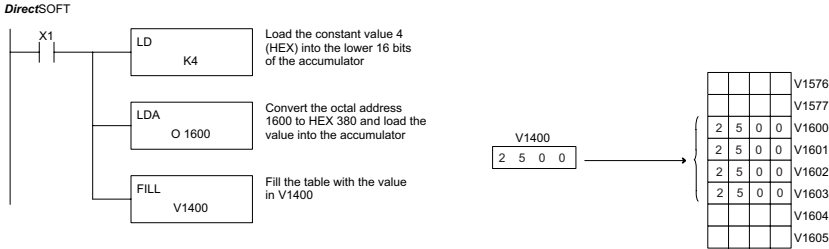
Step 3: Insert the Fill instructions which specifies the value to fill the table with.

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

Operand Data Type		Range D2-260/D2-262
	A	aaa
V-memory	V	All (See page 3-57)
Pointer	P	All V mem (See page 3-57)
Constant	K	0-FFFF

Discrete Bit Flag	Description
SP53	On if V-memory address is out of range

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



Handheld Programmer Keystrokes

\$	STR	→	B ₁	ENT														
SHFT	L _{ANDST}	D ₃	→	PREV	E ₄	ENT												
SHFT	L _{ANDST}	D ₃	A ₀	→	B ₁	G ₆	A ₀	A ₀	ENT									
SHFT	F ₅	I ₈	L _{ANDST}	L _{ANDST}	→	B ₁	E ₄	A ₀	A ₀	ENT								

Find (FIND)

 230

 240

 250-1

 260

 262

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

FIND
A aaa

DS	Used
HPP	Used

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

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

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

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

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

Operand Data Type	Range D2-260/D2-262	
	A	aaa
V-memory	V	All (See page 3-57)
Constant	K	0-FFFF

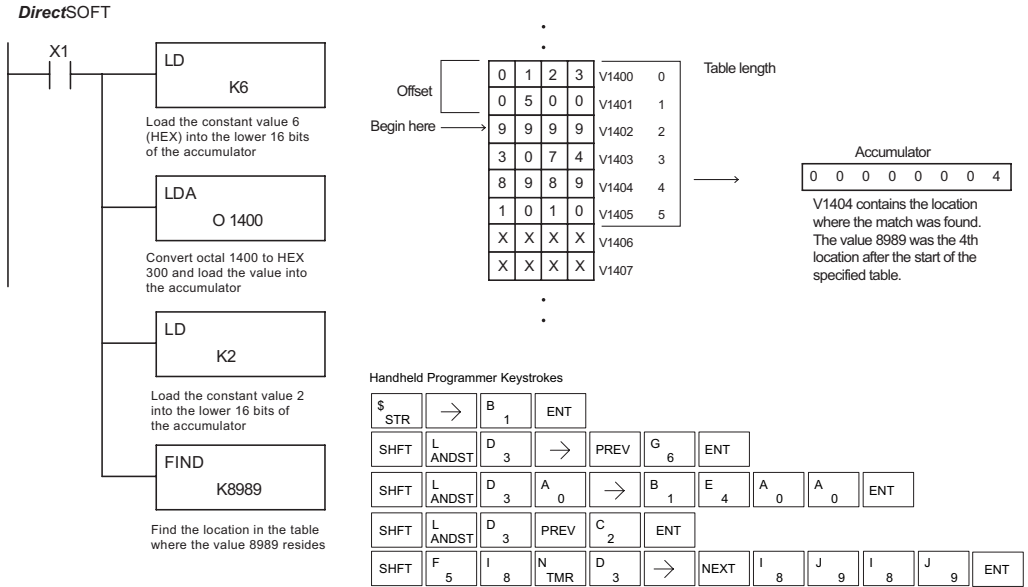
Helpful hint: For parameters that require HEX values when referencing memory locations, the LDA instruction can be used to convert an octal address to the HEX equivalent and load the value into the accumulator.

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



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

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



Find Greater Than (FDGT)

- 230
- 240
- 250-1
- 260
- 262

The Find Greater Than instruction is used to search for the first occurrence of a value in a V-memory table that is greater than the specified value (Aaaa), which can be either a V-memory location or a 4-digit constant. The function parameters are loaded into the first level of the accumulator stack and the accumulator by two additional instructions. Listed below are the steps necessary to program the Find Greater Than function.

FDGT
A aaa



NOTE: This instruction does not have an offset, such as the one required for the FIND instruction.

DS	Used
HPP	Used

- Step 1: Load the length of the table (up to 255 locations) into the first level of the accumulator stack. This parameter must be a HEX value, 0 to FFFF.
- Step 2: Load the starting V-memory location for the table into the accumulator. This parameter must be a HEX value.
- Step 3: Insert the FDGT instruction which specifies the greater than search value.

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

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

Operand Data Type		Range D2-260/D2-262
	A	aaa
V-memory	V	All (See page 3-57)
Constant	K	0-FFFF

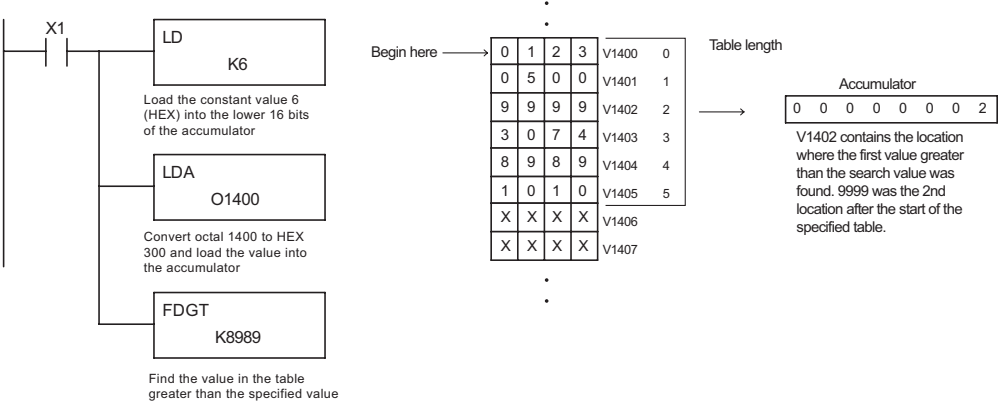
Discrete Bit Flags	Description
SP53	On if there is no value in the table that is equal to the search value.



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

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

DirectSOFT



Handheld Programmer Keystrokes

\$	→	B ₁	ENT																	
SHFT	L ANDST	D ₃	→	PREV	G ₆	ENT														
SHFT	L ANDST	D ₃	A ₀	→	B ₁	E ₄	A ₀	A ₀	ENT											
SHFT	F ₅	D ₃	G ₆	T MLR	→	NEXT	I ₈	J ₉	I ₈	J ₉	ENT									

Table to Destination (TTD)

- 230
- 240
- 250-1
- 260
- 262

DS	Used
HPP	Used

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

TTD
Vaaa

- Step 1: Load the length of the data table (number of V-memory locations) into the first level of the accumulator stack. This parameter must be a HEX value, 0 to FF.
- Step 2: Load the starting V-memory location for the table into the accumulator. (Remember, the starting location of the table is used as the table pointer.) This parameter must be a HEX value.
- Step 3: Insert the TTD instruction that specifies the destination V-memory location (Vaaa).
- Helpful hint: For parameters that require HEX values when referencing memory locations, the LDA instruction can be used to convert an octal address to the HEX equivalent and load the value into the accumulator.
- Helpful hint: The instruction will be executed every scan if the input logic is on. If you do not want the instruction to execute for more than one scan, a one shot (PD) should be used in the input logic.
- Helpful hint: The pointer location should be set to the value where the table operation will begin. The special relay SP0 or a one shot (PD) should be used so the value will only be set in one scan and will not affect the instruction operation.

Operand Data Type		Range D2-260/D2-262
		aaa
V-memory	V	All (See page 3-56)

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

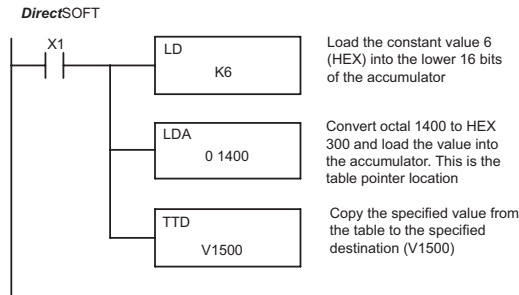
NOTE: Status flags (SPs) are only valid until:

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

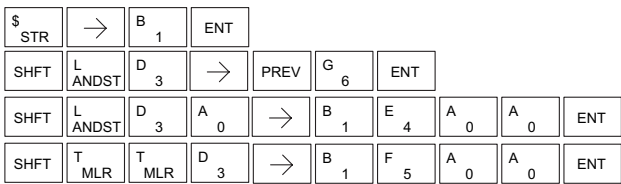


The pointer for this instruction starts at 0 and resets when the table length is reached. At first glance it may appear that the pointer should reset to 0. However, it resets to 1, not 0.

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



Handheld Programmer Keystrokes



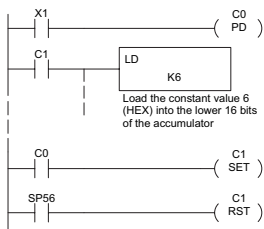
It is important to understand how the table locations are numbered. If you examine the example table, you'll notice that the first data location, V1401, will be used when the pointer is equal to 0, and again when the pointer is equal to 6. Why? Because the pointer is only equal to 0 before the very first execution. From then on, it increments from 1 to 6, and then resets to 1.

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

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

Destination					
X	X	X	X	X	V1500

DirectSOFT Display (optional latch example using SP56)

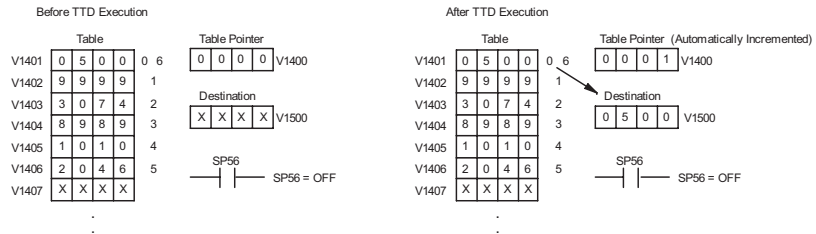


Since Special Relays are reset at the end of the scan, this latch must follow the TTD instruction in the program.

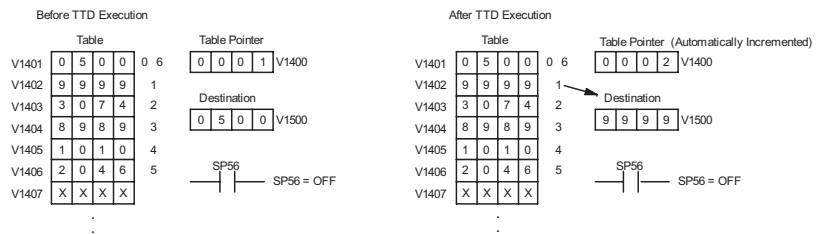
Chapter 5: Standard RLL Instructions

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

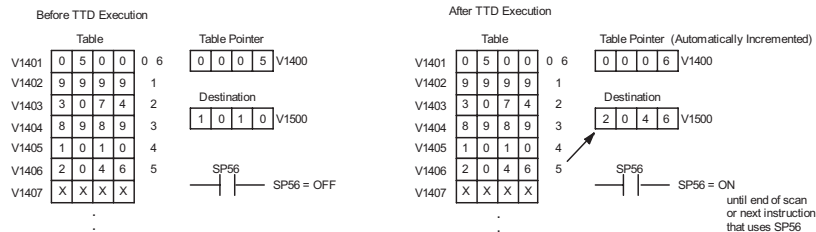
Scan N



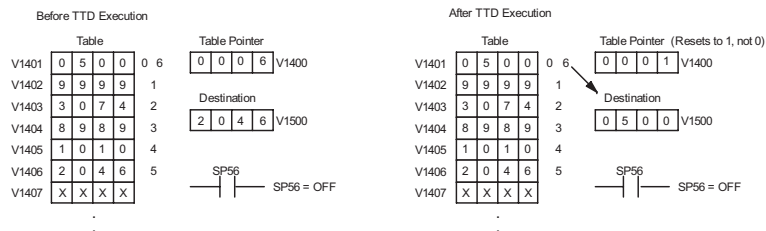
Scan N+1



Scan N+5



Scan N+6



Remove from Bottom (RFB)

	230
	240
	250-1
	260
	262

DS	Used
HPP	Used

The Remove From Bottom instruction moves a value from the bottom of a V-memory table to a V-memory location and decrements a table pointer by 1. The first V-memory location in the table contains the table pointer which indicates the next location in the table to be moved. The instruction will be executed once per scan provided the input remains on. The instruction will stop operation when the pointer equals 0. The function parameters are loaded into the first level of the accumulator stack and the accumulator by 2 additional instructions. Listed below are the steps necessary to program the Remove From Bottom function.

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

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

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

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

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

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

Operand Data Type		Range D2-260/D2-262
		aaa
V-memory	V	All (See page 3-57)

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

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

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



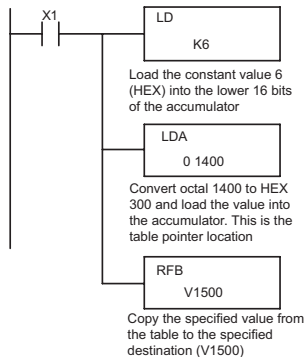
The pointer for this instruction can be set to start anywhere in the table. It is not set automatically. You have to load a value into the pointer somewhere in your program.



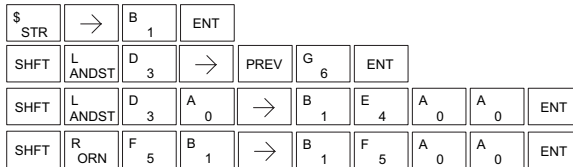
Chapter 5: Standard RLL Instructions

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

DirectSOFT



Handheld Programmer Keystrokes



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

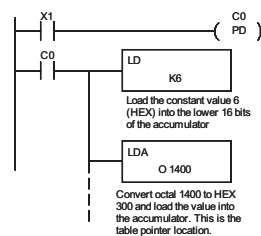
Also, our example uses a normal input contact (X1) to control the execution. Since the CPU scan is extremely fast, and the pointer decrements automatically, the table would cycle through the locations very quickly. If this is a problem for your application, you have an option of using a one-shot (PD)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

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

Table Pointer			
0	0	0	0

Destination			
X	X	X	X

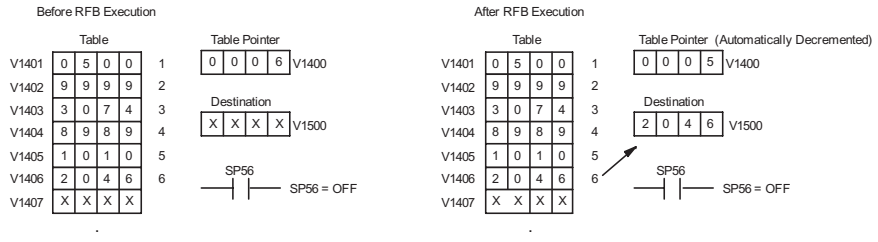
DirectSOFT (optional one-shot method)



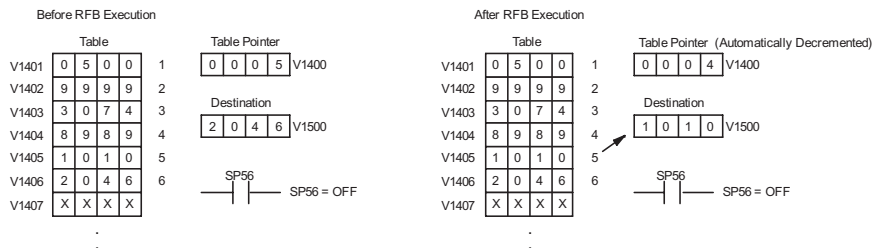
The following diagram shows the scan-by-scan results of the execution for our example program. Notice how the pointer automatically decrements from 6 to 0. Also, notice how SP56 is only on until the end of the scan.

Example of Execution

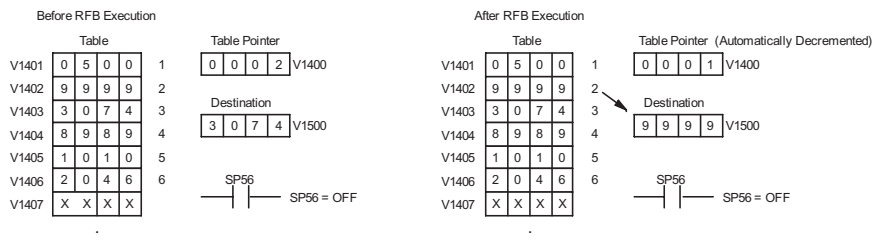
Scan N



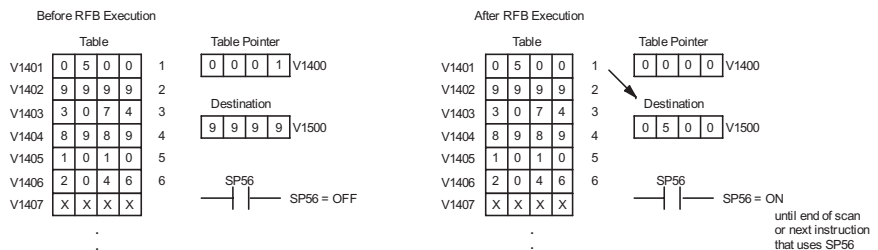
Scan N+1



Scan N+4



Scan N+5



Source to Table (STT)

	230
	240
	250-1
	260
	262

DS	Used
HPP	Used

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

STT
Vaaa

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

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

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

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

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

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

Operand Data Type		Range D2-260/D2-262
		aaa
V-memory	V	All (See page 3-57)

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

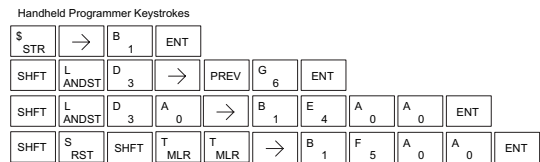
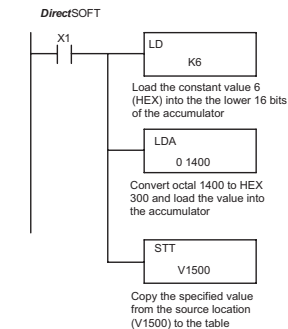
NOTE: Status flags (SPs) are only valid until:

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

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



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

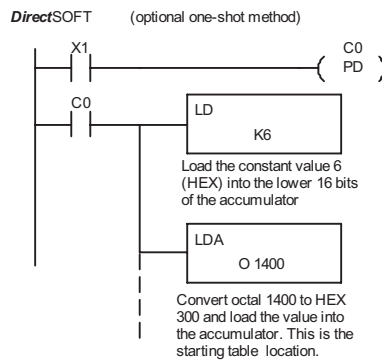


It is important to understand how the table locations are numbered. If you examine the example table, you'll notice that the first data storage location, V1401, will be used when the pointer is equal to 0, and again when the pointer is equal to 6. Why? Because the pointer is only equal to 0 before the very first execution. From then on, it increments from 1 to 6, and then resets to 1.

Also, our example uses a normal input contact (X1) to control the execution. Since the CPU scan is extremely fast, and the source data would be moved into all the table locations very quickly. If this is a problem for your application, you have an option of using a one-shot (PD) to move 1 value each time the input contact transitions from low to high.

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

Data Source				
0	5	0	0	V1500

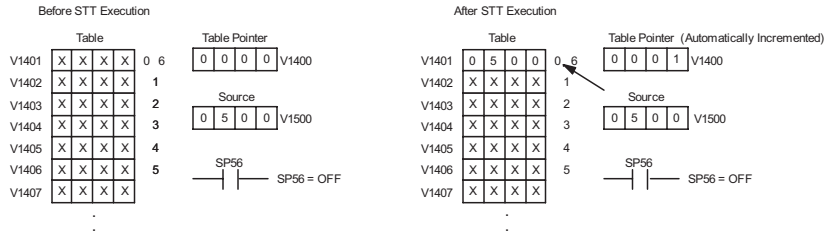


Chapter 5: Standard RLL Instructions

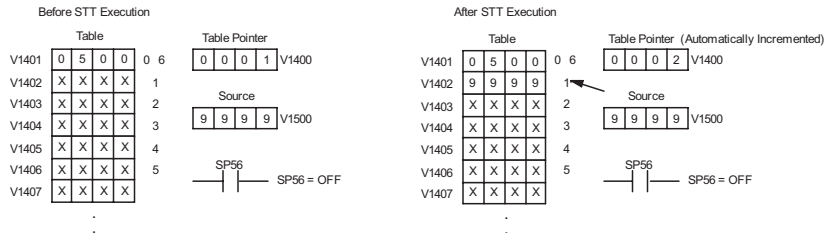
The following diagram shows the scan-by-scan results of the execution for our example program. Notice how the pointer automatically cycles from 0 to 6, and then starts over at 1 instead of 0. Also, notice how SP56 is affected by the execution. Although our example does not show it, we are assuming that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

Example of Execution

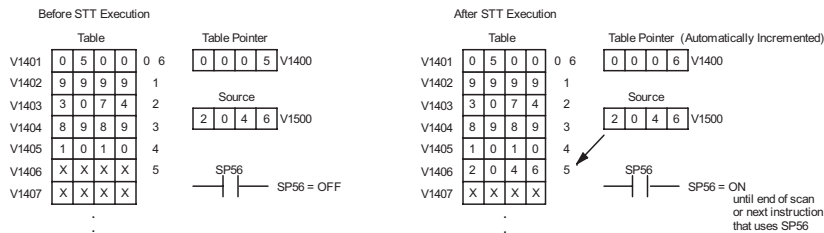
Scan N



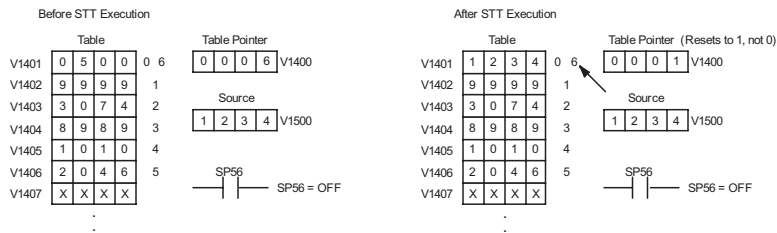
Scan N+1



Scan N+5



Scan N+6



Remove from Table (RFT)

- 230
- 240
- 250-1
- 260
- 262

DS	Used
HPP	Used

The Remove From Table instruction pops a value off of a table and stores it in a V-memory location. When a value is removed from the table all other values are shifted up 1 location. The first V-memory location in the table contains the table length counter. The table counter decrements by 1 each time the instruction is executed. If the length counter is 0 or greater than the maximum table length (specified in the first level of the accumulator stack), the instruction will not execute and SP56 will be on.

RFT
Vaaa

The instruction will be executed once per scan provided the input remains on. The function parameters are loaded into the first level of the accumulator stack and the accumulator by 2 additional instructions. Listed below are the steps necessary to program the Remove From Table function.

- Step 1: Load the length of the table (number of V-memory locations) into the first level of the accumulator stack. This parameter must be a HEX value, 0 to FF.
- Step 2: Load the starting V-memory location for the table into the accumulator. (Remember, the starting location of the table is used as the table length counter.) This parameter must be a HEX value.
- Step 3: Insert the RFT instructions which specifies destination V-memory location (Vaaa). This is where the value will be moved to.

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

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

Helpful hint: The table counter value should be set to indicate the starting point for the operation. Also, it must be set to a value that is within the length of the table. For example, if the table is 6 words long, then the allowable range of values that could be in the table counter should be between 1 and 6. If the value is outside of this range or 0, the data will not be moved from the table. Also, a one shot (PD) should be used so the value will only be set in one scan and will not affect the instruction operation.

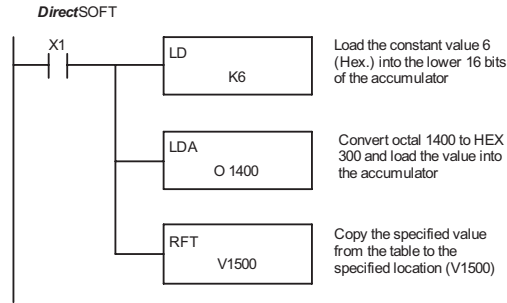
Operand Data Type		Range D2-260/D2-262
		aaa
V-memory	V	All (See page 3-57)

Discrete Bit Flags	Description
SP56	On when the table counter equals 0.

NOTE: Status flags (SPs) are only valid until:
— another instruction that uses the same flag is executed, or
— the end of the scan
The pointer for this instruction can be set to start anywhere in the table. It is not set automatically. You have to load a value into the pointer somewhere in your program.



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



Handheld Programmer Keystrokes

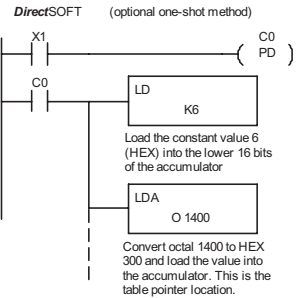
\$ STR	→	B 1	ENT							
SHFT	L ANDST	D 3	→	PREV	G 6	ENT				
SHFT	L ANDST	D 3	A 0	→	B 1	E 4	A 0	A 0	ENT	
SHFT	R ORN	F 5	T MLR	→	B 1	F 5	A 0	A 0	ENT	

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

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

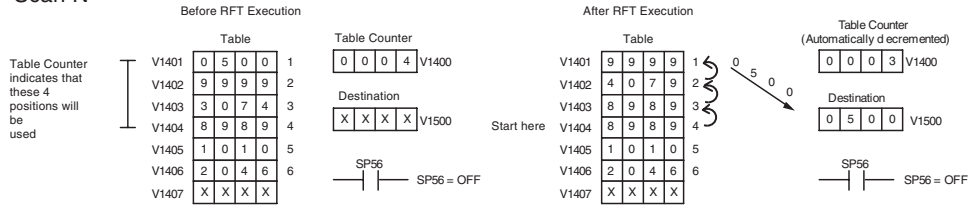
Table					
V1401	0	5	0	0	1
V1402	9	9	9	9	2
V1403	3	0	7	4	3
V1404	8	9	8	9	4
V1405	1	0	1	0	5
V1406	2	0	4	6	6
V1407	X	X	X	X	

Table Counter				
0	0	0	6	V1400
Destination				
X	X	X	X	V1500

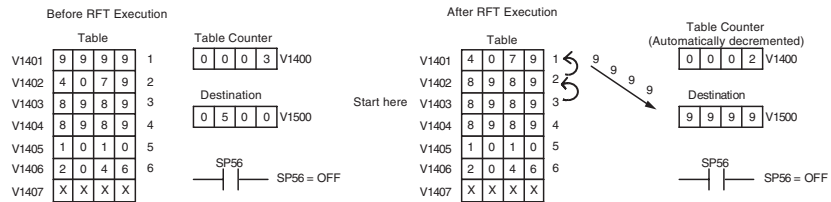


The following diagram shows the scan-by-scan results of the execution for our example program. In our example we're showing the table counter set to 4 initially (Remember, you can set the table counter to any value that is within the range of the table). The table counter automatically decrements from 4 to 0 as the instruction is executed. Notice how the last 2 table positions, 5 and 6, are not moved up through the table. Also, notice how SP56, which comes on when the table counter is 0, is only on until the end of the scan.

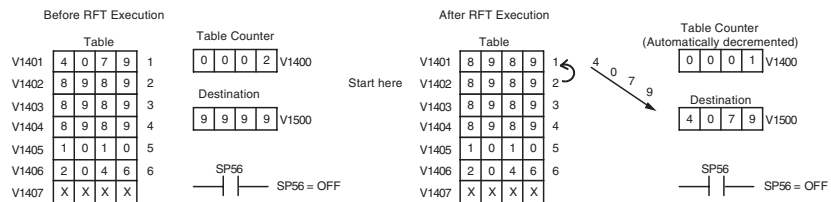
Scan N



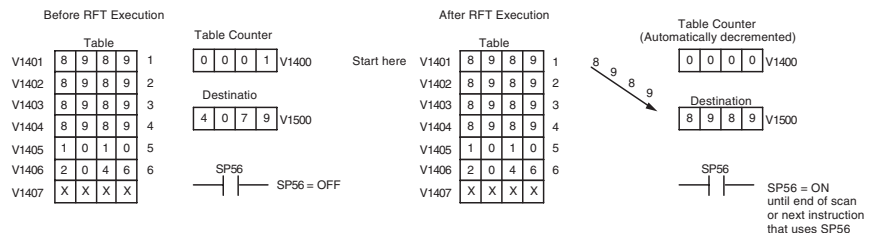
Scan N+1



Scan N+2



Scan N+3



Add to Top (ATT)

-  230
-  240
-  250-1
-  260
-  262

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

The instruction will be executed once per scan provided the input remains on. The function parameters are loaded into the first level of the accumulator stack and the accumulator by 2 additional instructions. Listed below are the steps necessary to program the Add To Top function.

ATT

V aaa

DS	Used
HPP	Used

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

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

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

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

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

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

Operand Data Type		Range D2-260/D2-262
		aaa
V-memory	V	All (See page 3-57)

Discrete Bit Flags	Description
SP56	On when the table counter equals 0.

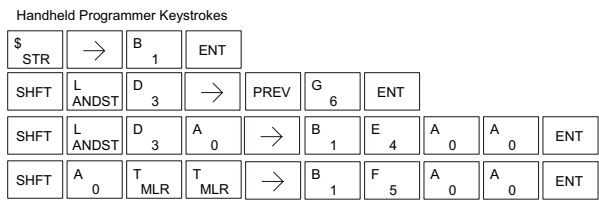
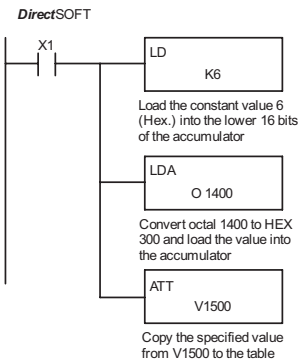
NOTE: Status flags (SPs) are only valid until:

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



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

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



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

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

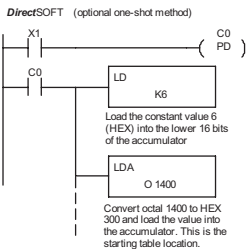
Table length – table counter = number of executions

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

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

Data Source				
X	X	X	X	V1500

(e.g.: 6 - 2 = 4)

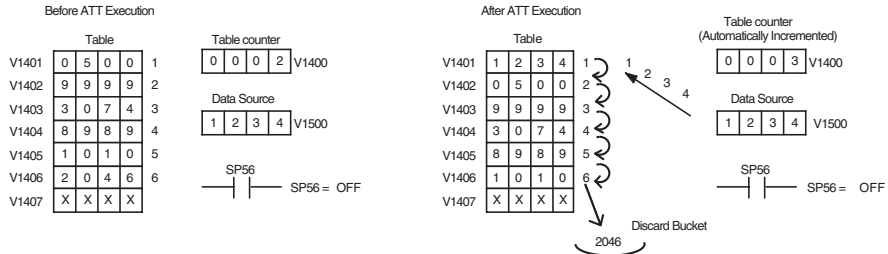


Chapter 5: Standard RLL Instructions

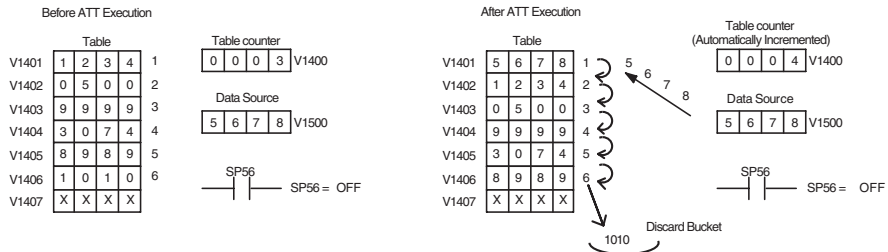
The following diagram shows the scan-by-scan results of the execution for our example program. The table counter is set to 2 initially, and it will automatically increment from 2 to 6 as the instruction is executed. Notice how SP56 comes on when the table counter is 6, which is equal to the table length.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

Example of Execution

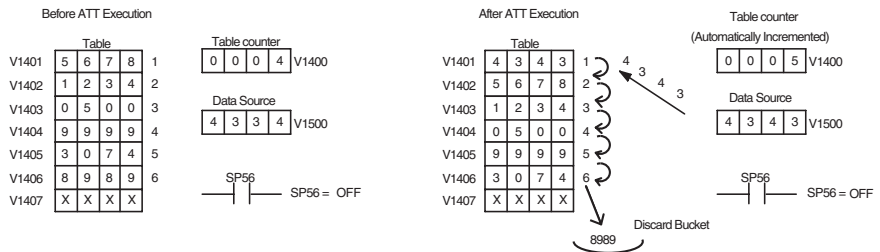
Scan N



Scan N+1



Scan N+2



Scan N+3

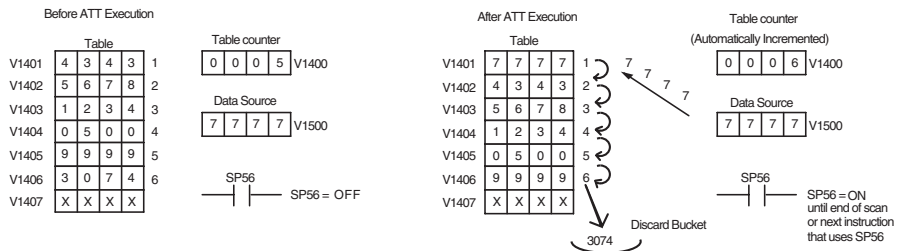


Table Shift Left (TSHFL)

230

240

250-1

260

262

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



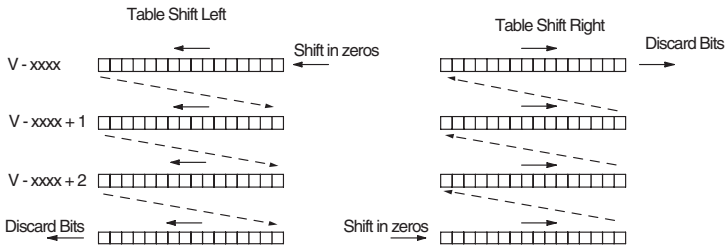
Table Shift Right (TSHFR)

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



DS	Used
HPP	Used

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



- Step 1: Load the length of the table (number of V-memory locations) into the first level of the accumulator stack. This parameter must be a HEX value, 0 to FF.
- Step 2: Load the starting V-memory location for the table into the accumulator. This parameter must be a HEX value. You can use the LDA instruction to convert an octal address to hex.
- Step 3: Insert the Table Shift Left or Table Shift Right instruction. This specifies the number of bit positions you wish to shift the entire table. The number of bit positions must be in octal.

Helpful hint: Remember that each V-memory location contains 16 bits. The bits of the first word of the table are numbered from 0 to 17 octal. If you want to shift the entire table by 20 bits, that is 24 octal. Flag 53 will be set if the number of bits to be shifted is larger than the total bits contained within the table. Flag 67 will be set if the last bit shifted (just before it is discarded) is a "1".

Operand Data Type		Range D2-260/D2-262
		aaa
V-memory	V	All (See page 3-57)

Chapter 5: Standard RLL Instructions

Discrete Bit Flags	Description
SP53	On when the number of bits to be shifted is larger than the total bits contained within the table
SP67	On when the last bit shifted (just before it is discarded) is a "1"



NOTE: Status flags are only valid until:

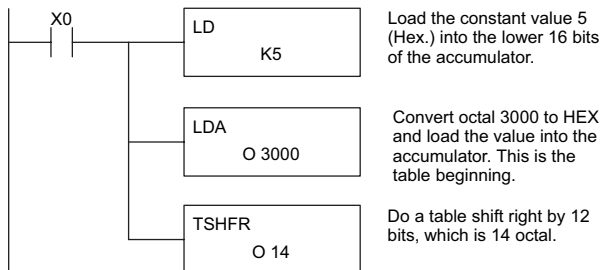
- the end of the scan
- or another instruction that uses the same flag is executed.

The example table to the right contains BCD data as shown (for demonstration purposes). Suppose we want to do a table shift right by 3 BCD digits (12 bits). Converting to octal, 12 bits is 14 octal. Using the Table Shift Right instruction and specifying a shift by octal 14, we have the resulting table shown at the far right. Notice that the 2–3–4 sequence has been discarded, and the 0–0–0 sequence has been shifted in at the bottom.

V 3000	V 3000
1 2 3 4	6 7 8 1
5 6 7 8	1 2 2 5
1 1 2 2	3 4 4 1
3 3 4 4	5 6 6 3
5 5 6 6	0 0 0 5

The following ladder example assumes the data at V3000 to V3004 already exists as shown above. We will use input X0 to trigger the Table Shift Right operation. First, we will load the table length (5 words) into the accumulator stack. Next, we load the starting address into the accumulator. Since V3000 is an octal number we have to convert it to hex by using the LDA command. Finally, we use the Table Shift Right instruction and specify the number of bits to be shifted (12 decimal), which is 14 octal.

DirectSOFT



Handheld Programmer Keystrokes

\$	→	A	ENT
STR	0		
SHFT	L	D	→
ANDST	3	PREV	F
			5
SHFT	L	D	→
ANDST	3	A	D
		0	A
			0
			A
			0
			ENT
SHFT	T	S	H
MLR	RST	7	F
			5
			R
			ORN
			→
			NEXT
			B
			1
			E
			4
			ENT

AND Move (ANDMOV)

- ✗ 230 The AND Move instruction copies data from a table to the specified memory location, ANDing each word with the accumulator data as it is written.

✗ 240

OR Move (ORMOV)

- ✓ 260 The Or Move instruction copies data from a table to the specified memory location, ORing each word with the accumulator contents as it is written.

✓ 262

Exclusive OR Move (XORMOV)

The Exclusive OR Move instruction copies data from a table to the specified memory location, XORing each word with the accumulator value as it is written.

The following description applies to the AND Move, OR

✗ 230

✗ 240

✗ 250-1

✓ 260

✓ 262

Move, and Exclusive OR Move instructions. A table is just a range of V-memory locations. These instructions copy the data of a table to another specified location, performing a logical operation on each word with the accumulator contents as the new table is written.

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

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

Step 3: Load the BCD/hex bit pattern into the accumulator which will be logically combined with the table contents as they are copied.

Step 4: Insert the AND Move, OR Move, or XOR Move instruction. This specifies the starting location of the copy of the original table. This new table will automatically be the same length as the original table.

DS	Used
HPP	Used

Operand Data Type	Range D2-260/D2-262
	aaa
V-memory	V All (See page 3-57)

The example table to the right contains BCD data as shown (for demonstration purposes). Suppose we want to move a table of two words at V3000 and AND it with K6666. The copy of the table at V3100 shows the result of the AND operation for each word.

V 3000	ANDMOV K 6666	V 3100
3 3 3 3	→	2 2 2 2
F F F F		6 6 6 6

The program on the next page performs the ANDMOV operation example above. It assumes that the data in the table at V3000 – V3001 already exists. First we load the table length (two words) into the accumulator. Next we load the starting address of the source table, using the LDA instruction. Then we load the data into the accumulator to be ANDed with the table. In the ANDMOV command, we specify the table destination, V3100.

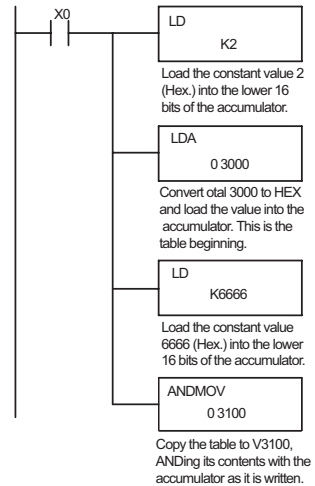
Handheld Programmer Keystrokes

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

The example to the right shows a table of two words at V3000 and logically ORs it with K8888. The copy of the table at V3100 shows the result of the OR operation for each word.

The program to the right performs the ORMOV example above. It assumes that the data in the table at V3000 – V3001 already exists. First we load the table length (two words) into the accumulator. Next we load the starting address of the source table, using the LDA instruction. Then we load the data into the accumulator to be ORed with the table. In the ORMOV command, we specify the table destination, V3100.

DirectSOFT



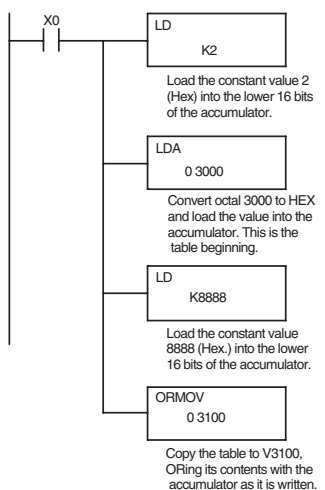
Handheld Programmer Keystrokes

\$	STR	→	A	0	ENT																
SHFT	L	ANDST	D	3	→	PREV	C	2	ENT												
SHFT	L	ANDST	D	3	→	D	3	A	0	→	D	3	A	0	A	0	A	0	ENT		
SHFT	L	ANDST	D	3	→	PREV	I	8	I	8	I	8	I	8	ENT						
Q	OR	SHFT	M	ORST	O	INST#	V	AND	→	D	3	B	1	A	0	A	0	ENT			

The example to the right shows a table of two words at V3000 and logical XORs it with K3333. The copy of the table at V3100 shows the result of the XOR operation for each word.

The ladder program example for the XORMOV is similar to the one above for the ORMOV. Just use the XORMOV instruction. On the Handheld Programmer, you must use the SHFT key and spell "XORMOV" explicitly.

DirectSOFT 32



Find Block (FINDB)

- 230
- 240
- 250-1
- 260
- 262

The Find Block instruction searches for an occurrence of a specified block of values in a V-memory table. The function parameters are loaded into the first and second levels of the accumulator stack and the accumulator by three additional instructions. If the block is found, its starting address will be stored in the accumulator. If the block is not found, flag SP53 will be set.

FINDB
Aaaa

DS	Used
HPP	N/A

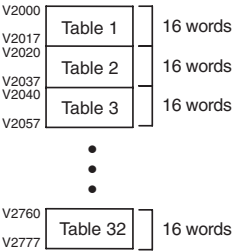
Operand Data Type		Range D2-260/D2-262
	A	aaa
V-memory	V	All (See page 3-56)
V-memory	P	All (See page 3-56)

Discrete Bit Flags	Description
SP53	On when the Find Block instruction was executed but did not find the block of data in table specified.

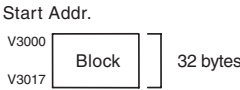
The steps below are necessary to program the Find Block function.

- Step 1: Load the number of bytes in the block to be located. This parameter must be a decimal value from 1 to 256.
- Step 2: Load the length of a table (number of words) to be searched. The Find Block will search multiple tables that are adjacent in V-memory. This parameter must be a decimal value from 1 to 128.
- Step 3: Load the ending location for all the tables into the accumulator. This parameter must be a HEX value. You can use the LDA instruction to convert an octal address to hex.
- Step 4: Load the table starting location for all the tables into the accumulator. This parameter must be a HEX value. You can use the LDA instruction to convert an octal address to hex.
- Step 5: Insert the Find Block instruction. This specifies the starting location of the block of data you are trying to locate.

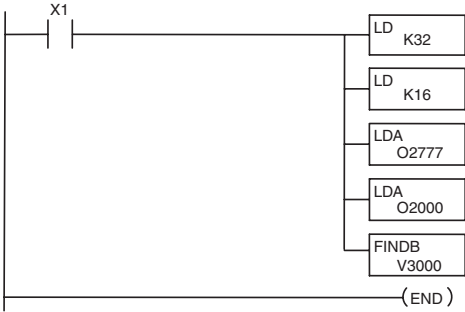
Start Addr.



End Addr.



Sample Program of FINDB



Swap (SWAP)

230

The Swap instruction exchanges the data in two tables of equal length.

240

The following steps apply to both the Set Bit and Reset Bit table instructions.

250-1

260

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

262

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

DS	Used
HPP	Used

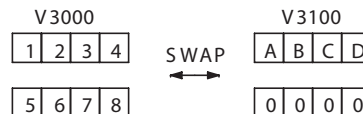
SWAP
V aaa

Step 3: Insert the Swap instruction. This specifies the starting address of the second table.

Helpful hint: The data swap occurs within a single scan. If the instruction executes on multiple consecutive scans, it will be difficult to know the actual contents of either table at any particular time. So, remember to swap just on a single scan.

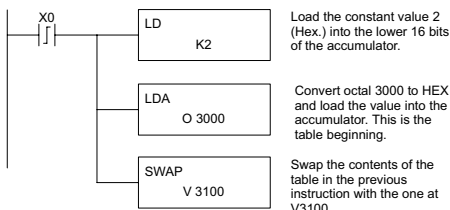
Operand Data Type	Range D2-260/D2-262
	aaa
V-memory	V All (See page 3-57)

The example to the right shows a table of two words at V3000. We will swap its contents with another table of two words at V3100 by using the Swap instruction.



The example program below uses a PD contact (triggers for one scan for off-to-on transition). First, we load the length of the tables (two words) into the accumulator. Then we load the address of the first table (V3000) into the accumulator using the LDA instruction, converting the octal address to hex. Note that it does not matter which table we declare "first," because the swap results will be the same.

DirectSOFT



Handheld Programmer Keystrokes

\$	STR	SHFT	P	D	→	A	ENT				
		SHFT	L	D	→	C	ENT				
		SHFT	L	D	A	D	A	A	A	ENT	
		SHFT	S	SHFT	W	A	P	D	B	A	A
			R	ANDN	A	CV	→	3	1	0	0
			ST		0						ENT

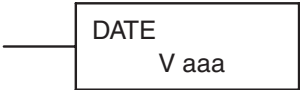
Clock/Calendar Instructions

Date (DATE)

- 230
- 240
- 250-1
- 260
- 262

DS	Used
HPP	Used

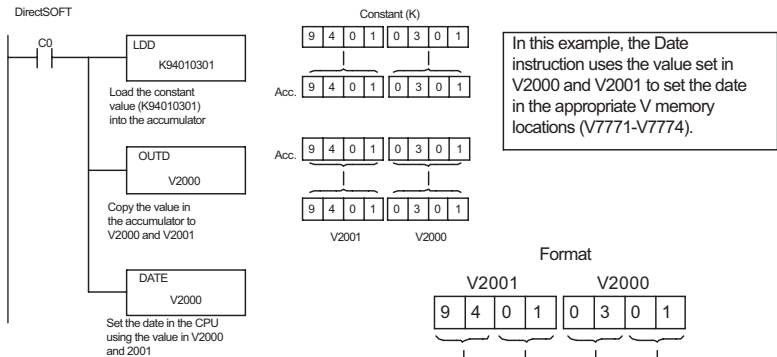
The Date instruction can be used to set the date in the CPU. The instruction requires two consecutive V-memory locations (Vaaa) to set the date. If the values in the specified locations are not valid, the date will not be set. The current date can be read from 4 consecutive V-memory locations (V7771–V7774).



Date	Range	V-memory Location (BCD) (READ Only)
Year	0-99	V7774
Month	1-12	V7773
Day	1-31	V7772
Day of Week	0-06	V7771
The values entered for the day of week are: 0=Sunday, 1=Monday, 2=Tuesday, 3=Wednesday, 4=Thursday, 5=Friday, 6=Saturday.		

Operand Data Type	Range	
	D2-250-1	D2-260/D2-262
	aaa	aaa
V-memory	V	All (See page 3-56)
		All (See page 3-57)

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



Handheld Programmer Keystrokes

Year Month

\$	STR	→	NEXT	NEXT	NEXT	NEXT	A ₀	ENT										
SHIFT	L	ANDST	D ₃	D ₃	→	PREV	J ₉	E ₄	A ₀	B ₁	ENT							
A ₀	D ₃	A ₀	B ₁	ENT														
GX	OUT	SHIFT	D ₃	→	C ₂	A ₀	A ₀	A ₀	ENT									
SHIFT	D ₃	A ₀	T	MLR	E ₄	→	C ₂	A ₀	A ₀	A ₀	A ₀	ENT						

Time (TIME)

- 230
- 240
- 250-1
- 260
- 262

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

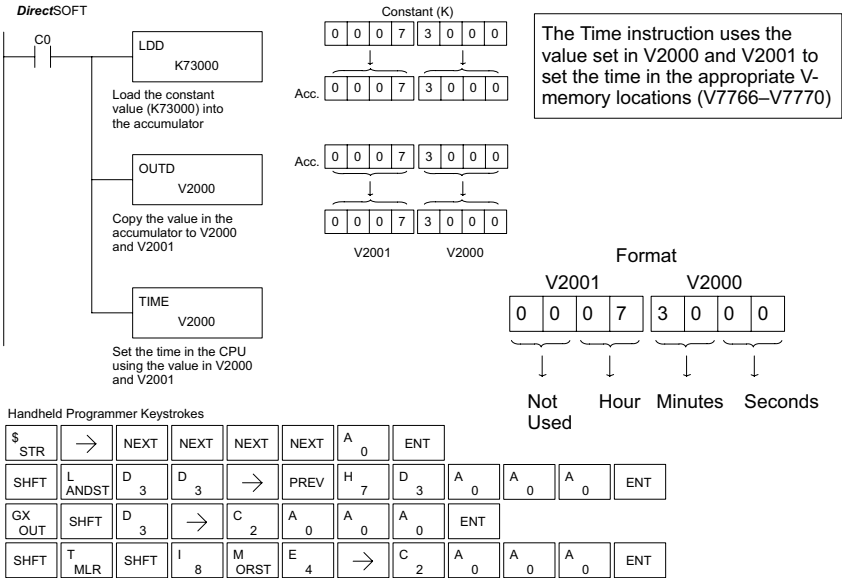
TIME
V aaa

DS	Used
HPP	Used

Date	Range	V-memory Location (BCD) (READ Only)
1/100 seconds (10ms)	0-99	V7747
Seconds	0-59	V7766
Minutes	0-59	V7767
Hour	0-23	V7770

Operand Data Type	Range	
	D2-250-1	D2-260/D2-262
	aaa	aaa
V-memory	V	All (See page 3-56)

In the following example, when C0 is on, the constant value (K73000) is loaded into the accumulator using the Load Double instruction (C0 should be a contact from a one shot (PD) instruction). The value in the accumulator is output to V2000 using the Out Double instruction. The Time instruction uses the value in V2000 to set the time in the CPU.



CPU Control Instructions

No Operation (NOP)

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

—(NOP)

- ✓ 230
- ✓ 240
- ✓ 250-1
- ✓ 260
- ✓ 262

DS	Used
HPP	Used

DirectSOFT



Handheld Programmer Keystrokes

SHFT	N	O	P	ENT
	TMR	INST#	CV	

End (END)

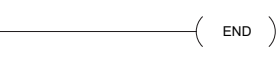
The End instruction marks the termination point of the normal program scan. An End instruction is required at the end of the main program body. If the End instruction is omitted, an error will occur and the CPU will not enter the Run Mode. Data labels, subroutines and interrupt routines are placed after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

—(END)

- ✓ 230
- ✓ 240
- ✓ 250-1
- ✓ 260
- ✓ 262

DS	Used
HPP	Used

DirectSOFT



Handheld Programmer Keystrokes

SHFT	E	N	D	ENT
	4	TMR	3	

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.

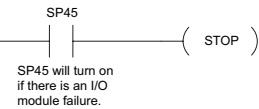
—(STOP)

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.

- ✓ 230
- ✓ 240
- ✓ 250-1
- ✓ 260
- ✓ 262

DS	Used
HPP	Used

DirectSOFT



Handheld Programmer Keystrokes

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

Reset Watch Dog Timer (RSTWT)

- 230
- 240
- 250-1
- 260
- 262

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

(RSTWT)

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

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

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

DS	Used
HPP	Used

DirectSOFT



Handheld Programmer Keystrokes

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

Program Control Instructions

Goto Label (GOTO) (LBL)

- 230
- 240
- 250-1
- 260
- 262

The Goto / Label skips all instructions between the Goto and the corresponding LBL instruction. The operand value for the Goto and the corresponding LBL instruction is the same. The logic between Goto and LBL instruction is not executed when the Goto instruction is enabled. Up to 128 Goto instructions and 64 LBL instructions can be used in the program.

(K aaa
GOTO)

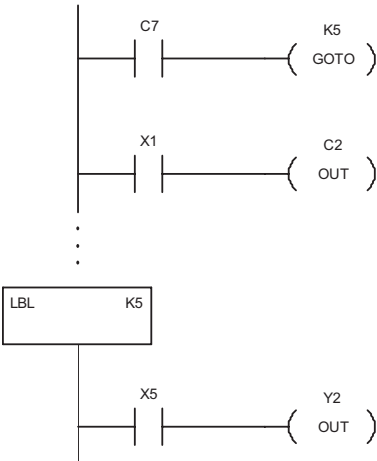
LBL K aaa

Operand Data Type	K	Range		
		D2-240	D2-250-1	D2-260/D2-262
		aaa	aaa	aaa
Constant	K	1-FFFF	1-FFFF	1-FFFF

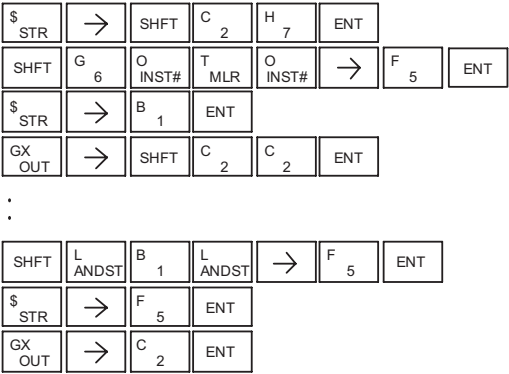
In the following example, when C7 is on, all the program logic between the GOTO and the corresponding LBL instruction (designated with the same constant Kaaa value) will be skipped. The instructions being skipped will not be executed by the CPU.

DS	Used
HPP	Used

DirectSOFT



Handheld Programmer Keystrokes

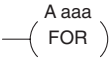


For/Next (FOR) (NEXT)

- 230
- 240
- 250-1
- 260
- 262

DS	Used
HPP	Used

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



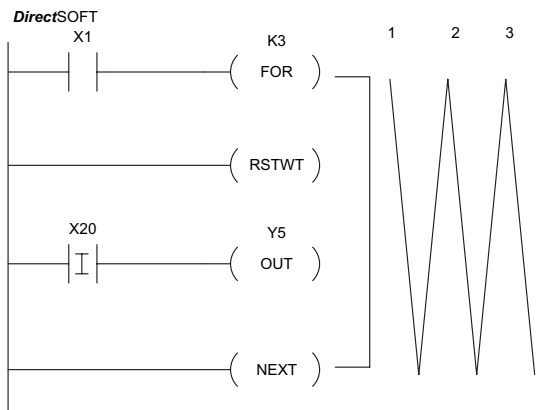
For/Next instructions cannot be nested. Up to 64 For/Next loops may be used in a program. If the maximum number of For/Next loops is exceeded, error E413 will occur.



The normal I/O update and CPU housekeeping is suspended while executing the For/Next loop. The program scan time can increase significantly, depending on the number of times the logic between the For and Next instruction is executed. With the exception of immediate I/O instructions, I/O will not be updated until the program execution is completed for that scan. Depending on the length of time required to complete the program execution, it may be necessary to reset the watchdog timer inside of the For/Next loop using the RSTWT instruction.

Operand Data Type		Range		
		D2-240	D2-250-1	D2-260/D2-262
	A	aaa	aaa	aaa
V-memory	V	All (See page 3-54)	All (See page 3-55)	All (See page 3-57)
Constant	K	1-9999	1-9999	1-9999

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



Handheld Programmer Keystrokes

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

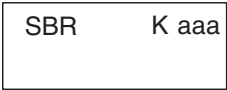
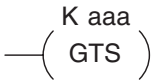
Goto Subroutine (GTS) (SBR)

- 230
- 240
- 250-1
- 260
- 262

DS	Used
HPP	Used

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

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



Operand Data Type	Range		
	D2-240	D2-250-1	D2-260/D2-262
	aaa	aaa	aaa
Constant	K 1-FFFF	1-FFFF	1-FFFF

Subroutine Return (RT)

- 230
- 240
- 250-1
- 260
- 262

When a Subroutine Return is executed in the subroutine, the CPU will return to the point in the main body of the program from which it was called. The Subroutine Return is used as termination of the subroutine, which must be the last instruction in the subroutine and is a stand-alone instruction (no input contact on the rung).



Subroutine Return Conditional (RTC)

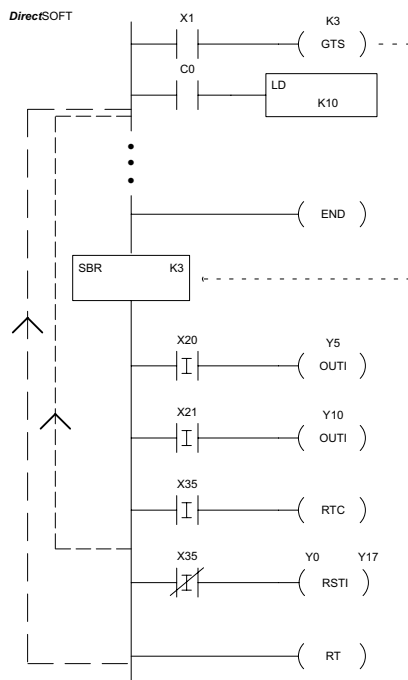
- 230
- 240
- 250-1
- 260
- 262

DS	Used
HPP	Used

The Subroutine Return Conditional instruction is an optional instruction used with an input contact to implement a conditional return from the subroutine. The Subroutine Return (RT) is still required for termination of the Subroutine.



In the following example, when X1 is on, Subroutine K3 will be called. The CPU will jump to the Subroutine Label K3, and the ladder logic in the subprogram will be executed. If X35 is on, the CPU will return to the main program at the RTC instruction. If X35 is not on, Y0–Y17 will be reset to off and then the CPU will return to the main body of the program.

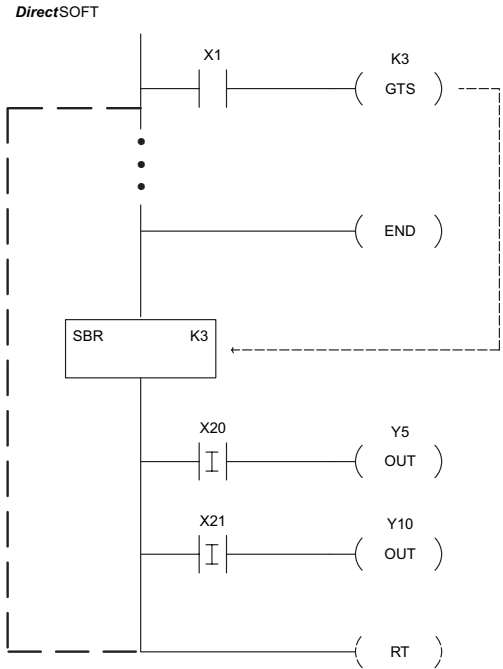


Handheld Programmer Keystrokes

STR	→	1	ENT
SHFT	G	T	S
→	K	3	ENT
⋮			
SHFT	E	N	D
SHFT	S	SHFT	B
→	K	3	ENT
STR	SHFT	I	→
OUT	SHFT	I	→
STR	SHFT	I	→
OUT	SHFT	I	→
STR	SHFT	I	→
SHFT	R	T	C
STRN	SHFT	I	→
RST	SHFT	I	→
SHFT	R	T	ENT

Chapter 5: Standard RLL Instructions

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



Handheld Programmer Keystrokes

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

Master Line Set (MLS)

- ✓ 230

✓ 240

✓ 250-1

✓ 260

✓ 262
- 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 RLLPLUS, the logic within the master control relays is still scanned and updated even though it will not function if the MLS is off.
- (K aaa)

(MLS)

Operand Data Type		Range			
		D2-230	D2-240	D2-250-1	D2-260/D2-262
		aaa	aaa	aaa	aaa
Constant	K	1-7	1-7	1-7	1-7

- ✓ 230

✓ 240

✓ 250-1

✓ 260

✓ 262
- 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.
- (K aaa)

(MLR)

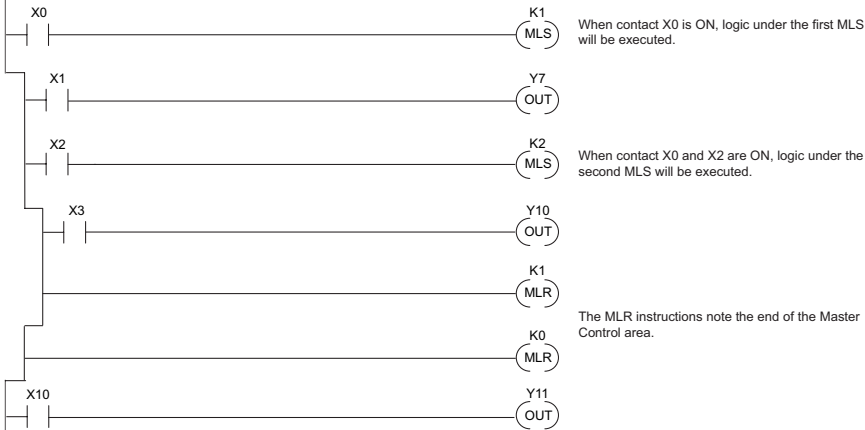
Operand Data Type		Range			
		D2-230	D2-240	D2-250-1	D2-260/D2-262
		aaa	aaa	aaa	aaa
Constant	K	1-6	1-6	1-6	1-6

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.

DS	Used
HPP	Used

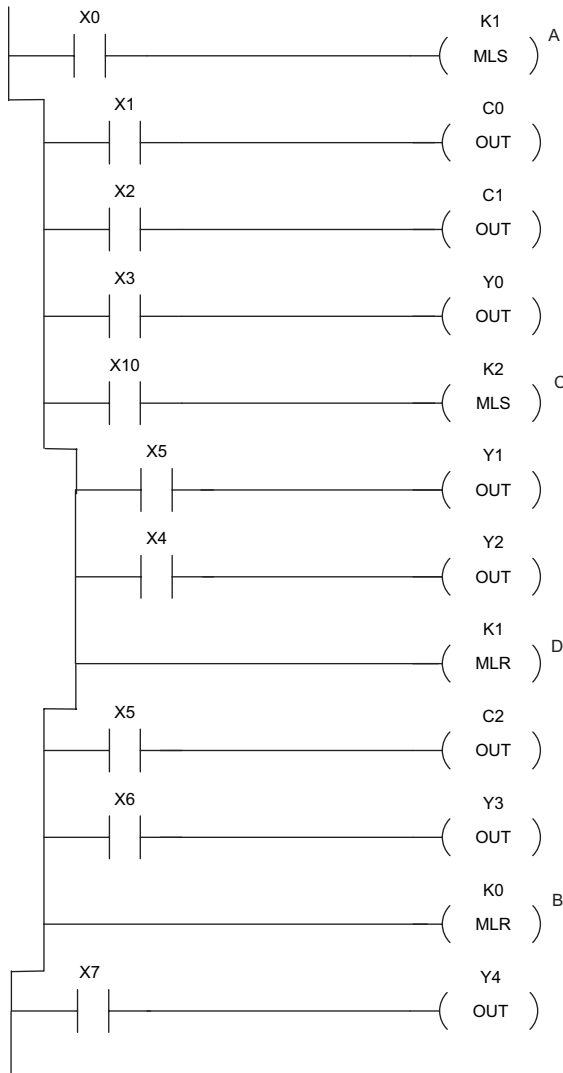
DirectSOFT



MLS/MLR Example

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

DirectSOFT



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)

 230

 240

 250-1

 260

 262

DS	Used
HPP	Used

The Interrupt instruction allows a section of ladder logic to be placed outside the main body of the program and executed when needed. Interrupts can be called from the program or by external interrupts via the counter interface module (D2-CTRINT), which provides 4 interrupts.

The software interrupt uses interrupt #00 which means the hardware interrupt #0 and the software interrupt cannot be used together.

Typically, interrupts will be used in an application where a fast response to an input is needed or a program section needs to execute faster than the normal CPU scan. The interrupt label and all associated logic must be placed after the End statement in the program. When the interrupt routine is called from the interrupt module or software interrupt, the CPU will complete execution of the instruction it is currently processing in ladder logic, then execute the designated interrupt routine. Interrupt module interrupts are labeled in octal to correspond with the hardware input signal (X1 will initiate interrupt INT1). There is only one software interrupt, and it is labeled INT 0. The program execution will continue from the point it was before the interrupt occurred once the interrupt is serviced.

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

INT	O aaa
-----	-------



NOTE: See the example program of a software interrupt.

Operand Data Type		Range		
		D2-240	D2-250-1	D2-260/D2-262*
		aaa	aaa	aaa
Constant	0	0-3	0-3	0-3

D2-240/D2-250-1/D2-260/D2-262*			
Software		Hardware	
Interrupt Input	Interrupt Routine	Interrupt Input	Interrupt Routine
V7634 sets interrupt time	INT 0	X0 (cannot be used along with s/w interrupt)	INT 0
-	-	X1	INT 1
-	-	X2	INT 2
-	-	X3	INT 3



*** NOTE:** D2-262 supports the Software Interrupt, INT 0, only.

Interrupt Return (IRT)

- ☒ 230 When an Interrupt Return is executed in the interrupt routine, the
☒ 240 CPU will return to the point in the main body of the program from
☒ 250-1 which it was called. The Interrupt Return is programmed as the last
☒ 260 instruction in an interrupt routine and is a stand alone instruction
☒ 262 (no input contact on the rung). —(IRT)

Interrupt Return Conditional (IRTC)

- ☒ 230 The Interrupt Return Conditional instruction is a optional instruction
☒ 240 used with an input contact to implement a conditional return from
☒ 250-1 the interrupt routine. The Interrupt Return is required to terminate
☒ 260 the interrupt routine. —(IRTC)

Enable Interrupts (ENI)

- ☒ 230 The Enable Interrupt instruction is programmed in the main body
☒ 240 of the application program (before the End instruction) to enable
☒ 250-1 hardware or software interrupts. Once the coil has been energized,
☒ 260 interrupts will be enabled until they are disabled by the Disable
☒ 262 Interrupt instruction. —(ENI)

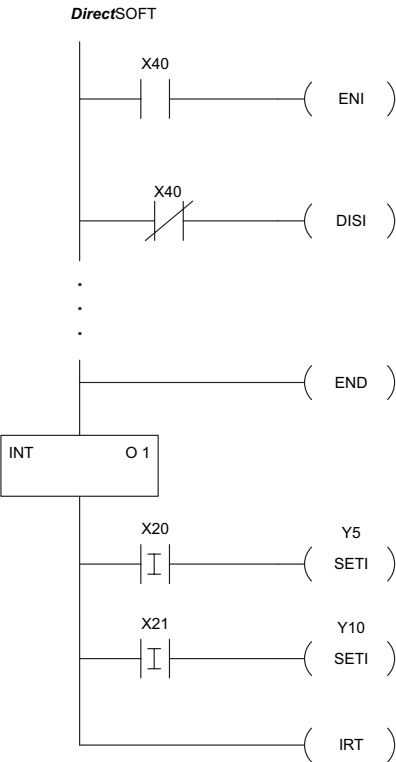
Disable Interrupts (DISI)

- ☒ 230 The Disable Interrupt instruction is programmed in the main body of
☒ 240 the application program (before the End instruction) to disable both
☒ 250-1 hardware or software interrupts. Once the coil has been energized,
☒ 260 interrupts will be disabled until they are enabled by the Enable
☒ 262 Interrupt instruction. —(DISI)

Interrupt Example for Interrupt Module

In the following example, when X40 is on, the interrupts will be enabled. When X40 is off, the interrupts will be disabled. When an interrupt signal X1 is received, the CPU will jump to the interrupt label INT O 1. The application ladder logic in the interrupt routine will be performed. The CPU will return to the main body of the program after the IRT instruction is executed.

DS	Used
HPP	Used



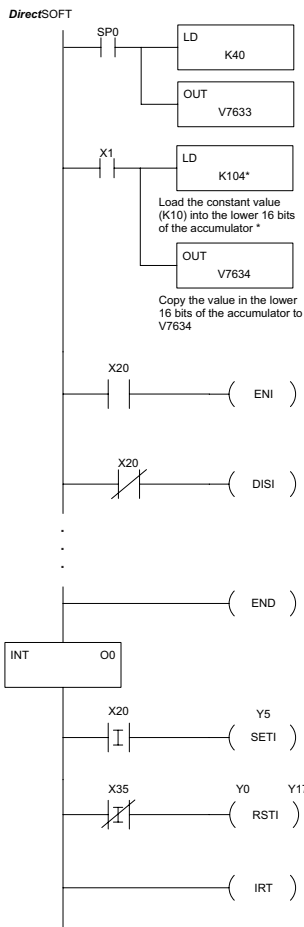
Handheld Programmer Keystrokes

\$STR	→	E ₄	A ₀	ENT	
SHFT	E ₄	N _{TMR}	I ₈	ENT	
SPSTRN	→	E ₄	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	→	B 1	ENT
\$ STR	SHFT	I 8	→	C 2	A 0	ENT
X SET	SHFT	I 8	→	F 5	ENT	
\$ STR	SHFT	I 8	→	C 2	B 1	ENT
X SET	SHFT	I 8	→	B 1	A 0	ENT
SHFT	I 8	R ORN	T MLR	ENT		

Interrupt Example for Software Interrupt

In the following example, when X1 is on, the value 10 is copied to V7634. This value sets the software interrupt to 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.



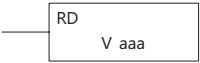
NOTE: Only one software interrupt is allowed and it must be Int0.

Intelligent I/O Instructions

Read from Intelligent Module (RD)

- ✓ 230
- ✓ 240
- ✓ 250-1
- ✓ 260
- ✓ 262

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



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

DS	Used
HPP	Used

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

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

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

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

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

Operand Data Type	Range			
	D2-230	D2-240	D2-250-1	D2-260/D2-262
	aaa	aaa	aaa	aaa
V-memory	V	All (See page 3-54)	All (See page 3-55)	All (See page 3-56)

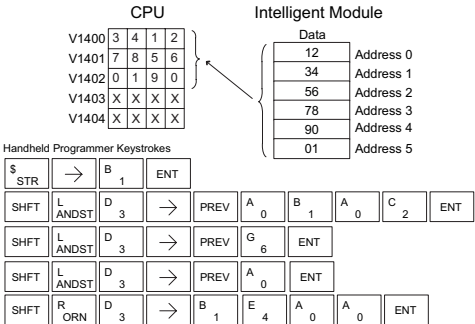
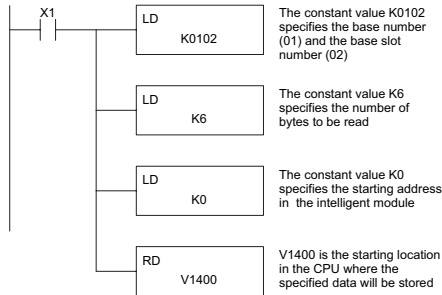
Discrete Bit Flags	Description
SP54	On when RX, WX, RD, WT instructions are executed with the wrong parameters.



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

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

DirectSOFT

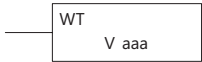


Write to Intelligent Module (WT)

- 230
- 240
- 250-1
- 260
- 262

DS	Used
HPP	Used

The Write to Intelligent Module instruction writes a block of data (1 to 128 bytes maximum) to an intelligent I/O module from a block of V-memory in the CPU. The function parameters are loaded into the first and second level of the accumulator stack and the accumulator by three additional instructions. Listed below are the steps necessary to program the Read from Intelligent module function.



- Step 1: Load the base number (0 to 3) into the first byte and the slot number (0 to 7) into the second byte of the second level of the accumulator stack.
- Step 2: Load the number of bytes to be transferred into the first level of the accumulator stack (maximum of 128 bytes).
- Step 3: Load the intelligent module address which will receive the data into the accumulator. This parameter must be a HEX value.
- Step 4: Insert the WT instruction which specifies the starting V-memory location (Vaaa) where the data will be written from in the CPU.

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

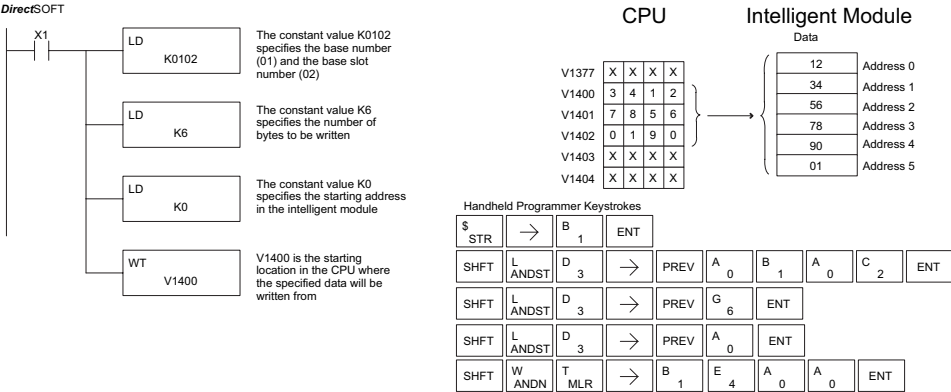
Operand Data Type		Range			
		D2-230	D2-240	D2-250-1	D2-260/D2-262
		aaa	aaa	aaa	aaa
V-memory	V	All (See page 3-54)	All (See page 3-55)	All (See page 3-56)	All (See page 3-57)

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



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

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



Network Instructions

Read from Network (RX)

☒ 230

☒ 240

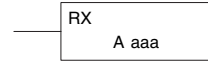
☒ 250-1

☒ 260

☒ 262

DS	Used
HPP	Used

The Read from Network instruction is used by the master device on a network to read a block of data from another CPU. The function parameters are loaded into the first and second levels of the accumulator stack and the accumulator by three additional instructions. Listed below are the steps necessary to program the Read from Network function.



Step 1: Load the slave address (0 to 90 BCD) into the first byte, and load the PLC internal port (KF1) or slot number of the master DCM or ECOM (0 to 7) into the second byte of the second level of the accumulator stack.

Step 2: Load the number of bytes (0 to 128 BCD, multiple of 2) to be transferred into the first level of the accumulator stack.

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

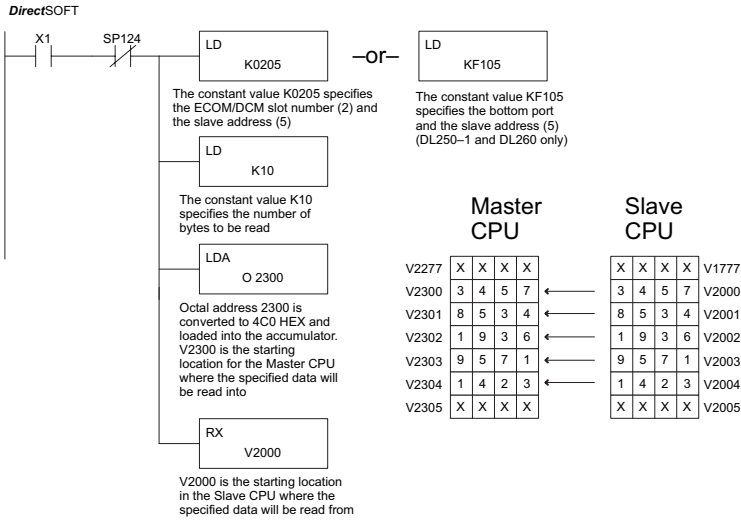
Step 4: Insert the RX instruction which specifies the starting V-memory location (Aaaa) where the data will be read from in the slave.

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

Operand Data Type		D2-240 Range	D2-250-1 Range	D2-260/D2-262 Range
	A	aaa	aaa	aaa
V-memory	V	All (See page 3-54)	All (See page 3-55)	All (See page 3-56)
Pointer	P	All V-memory (See page 3-54)	All V-memory (See page 3-55)	All V-memory (See page 3-56)
Inputs	X	0-477	0-777	0-1777
Outputs	Y	0-477	0-777	0-1777
Control Relays	C	0-377	0-1777	0-3777
Stage	S	0-777	0-1777	0-1777
Timer	T	0-177	0-377	0-377
Counter	CT	0-177	0-177	0-377
Global I/O	GX/ GY	-	-	0-3777
Special Relay	SP	0-137 540-617	0-777	0-777

Chapter 5: Standard RLL Instructions

In the following example, when X1 is on and the module busy relay SP124 (see special relays) is not on, the RX instruction will access an ECOM or DCM operating as a master in slot 2. Ten consecutive bytes of data (V2000 – V2004) will be read from a CPU at station address 5 and copied into V-memory locations V2300–V2304 in the CPU with the master DCM or ECOM.



Handheld Programmer Keystrokes

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

Write to Network (WX)

☐ 230

☒ 240

☒ 250-1

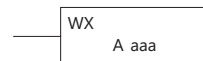
☒ 260

☒ 262

DS	Used
HPP	Used

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

The function parameters are loaded into the first and second levels of the accumulator stack and the accumulator by three additional instructions. Listed below are the steps necessary to program the Write to Network function.



Step 1: Load the slave address (0 to 90 BCD) into the first byte and the PLC internal port (KF1) or slot number of the master DCM or ECOM (0 to 7) into the second byte of the second level of the accumulator stack.

Step 2: Load the number of bytes (0 to 128 BCD, multiple of 2) to be transferred into the first level of the accumulator stack.

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

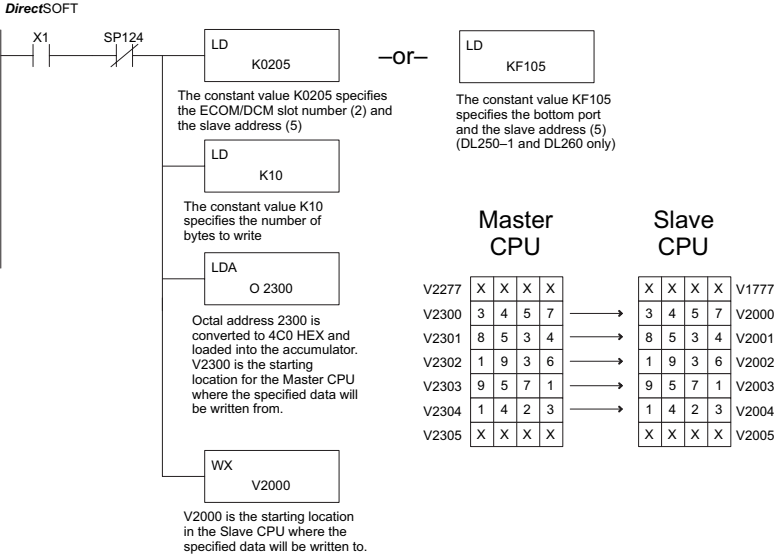
Step 4: Insert the WX instruction which specifies the starting V-memory location (Aaaa) where the data will be written to the slave.

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

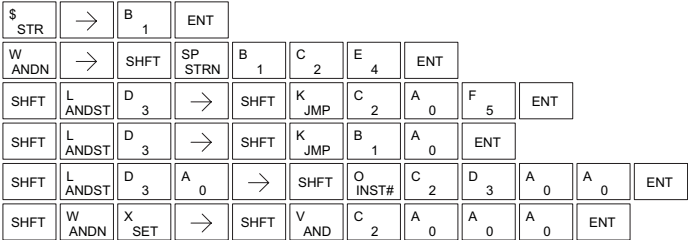
Operand Data Type		D2-240 Range	D2-250-1 Range	D2-260/D2-262 Range
	A	aaa	aaa	aaa
V-memory	V	All (See page 3-54)	All (See page 3-55)	All (See page 3-56)
Pointer	P	All V-memory (See page 3-54)	All V-memory (See page 3-55)	All V-memory (See page 3-56)
Inputs	X	0-477	0-777	0-1777
Outputs	Y	0-477	0-777	0-1777
Control Relays	C	0-377	0-1777	0-3777
Stage	S	0-777	0-1777	0-1777
Timer	T	0-177	0-377	0-377
Counter	CT	0-177	0-177	0-377
Global I/O	GX/ GY	-	-	0-3777
Special Relay	SP	0-137 540-617	0-777	0-777

Chapter 5: Standard RLL Instructions

In the following example when X1 is on and the module busy relay SP124 (see special relays) is not on, the WX instruction will access a DCM or ECOM operating as a master in slot 2. Ten consecutive bytes of data is read from the CPU at station address 5 and copied to V-memory locations V2000–V2004 in the slave CPU.



Handheld Programmer Keystrokes



Message Instructions

Fault (FAULT)

 230

 240

 250-1

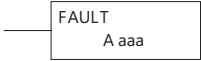
 260

 262

DS	Used
HPP	Used

The Fault instruction is used to display a message on the handheld programmer or DirectSOFT. 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.



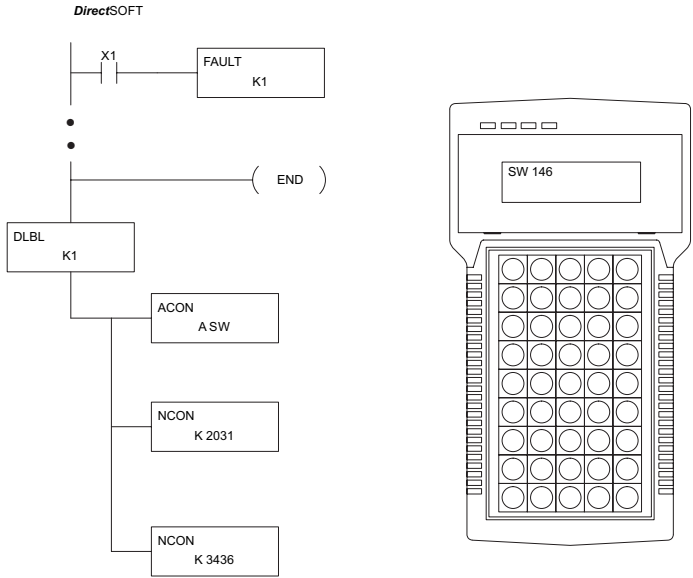
Operand Data Type	Range		
	D2-240	D2-250-1	D2-260/D2-262
A	aaa	aaa	aaa
V-memory V	All (See page 3-54)	All (See page 3-55)	All (See page 3-57)
Constant K	1-FFFF	1-FFFF	1-FFFF



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

\$	→	B	ENT											
SHFT	F	A	U	L	T	→	B	ENT						
	5	0	ISG	ANDST	MLR		1							
•														
•														
SHFT	E	N	D	ENT										
	4	TMR	3											
SHFT	D	L	B	L	→	B	ENT							
	3	ANDST	1	ANDST		1								
SHFT	A	C	O	N	→	S	W	ENT						
	0	2	INST#	TMR		RST	ANDN							
SHFT	N	C	O	N	→	C	A	D	B	ENT				
	TMR	2	INST#	TMR		2	0	3	1					
SHFT	N	C	O	N	→	D	E	D	G	ENT				
	TMR	2	INST#	TMR		3	4	3	6					

Data Label (DLBL)

- ☒ 230 The Data Label instruction marks the beginning of an ASCII/numeric data area. DLBLs are programmed after the End statement. A maximum of 64 (D2-240 and D2-250-1D2-260/262) or 32 (D2-230) DLBL instructions can be used in a program. Multiple NCONs and ACONs can be used in a DLBL area.

DLBL
K aaa

Operand Data Type		Range			
		D2-230	D2-240	D2-250-1	D2-260/D2-262
		aaa	aaa	aaa	aaa
Constant	K	1-FFFF	1-FFFF	1-FFFF	1-FFFF

ASCII Constant (ACON)

- ☒ 230 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.
- ☒ 240 Two ASCII characters can be stored in an ACON instruction.
- ☒ 250-1 If only one character is stored in an ACON, a leading space will be printed in the Fault message.
- ☒ 260
- ☒ 262

ACON
A aaa

Operand Data Type		D2-230 Range	D2-240 Range	D2-250-1 Range	D2-260/D2-262 Range
		aaa	aaa	aaa	aaa
ASCII	A	0-9 A-Z	0-9 A-Z	0-9 A-Z	0-9 A-Z

Numerical Constant (NCON)

- ☒ 230 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.

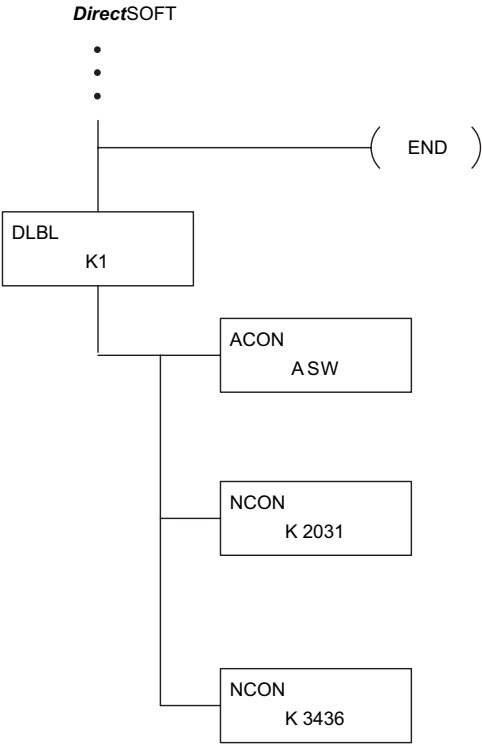
NCON
K aaa

<input checked="" type="checkbox"/> 260	DS	Used
<input checked="" type="checkbox"/> 262	HPP	Used

Operand Data Type		Range			
		D2-230	D2-240	D2-250-1	D2-260/D2-262
		aaa	aaa	aaa	aaa
Constant	K	0-FFFF	0-FFFF	0-FFFF	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.



Handheld Programmer Keystrokes

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

Print Message (PRINT)

- 230
- 240
- 250-1
- 260
- 262

The Print Message instruction prints the embedded text or text/data variable message to the specified communications port (port 2 on the D2-250-1, D2-260 and D2-262 CPUs), which must have the communications port configured.

PRINT A aaa

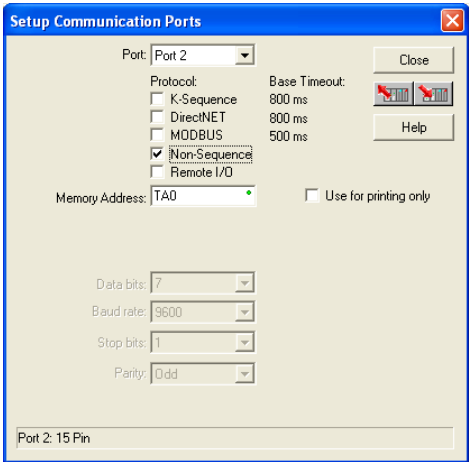
"Hello, this is a PLC message"

DS	Used
HPP	N/A

Data Type	Range	
	D2-250-1	D2-260/D2-262
A	aaa	aaa
Constant	K	2

You may recall from the CPU specifications in Chapter 3 that the D2-250–1, D2-260 and D2-262 ports are capable of several protocols. To configure a port using the Handheld Programmer, use AUX 56 and follow the prompts, making the same choices as indicated below on this page. To configure a port in DirectSOFT, choose the PLC menu, then Setup, then Setup Secondary Comm Port.

- Port: From the port number list box at the top, choose "Port 2."
- Protocol: Click the check box to the left of "Non-sequence." The Setup Communication Ports dialog box opens.



- Memory Address: Choose a V-memory address for DirectSOFT to use to store the port setup information. You will need to reserve 66 contiguous words in V-memory for this purpose. Select "Use for printing only" if it applies.
- Baud Rate: Choose the baud rate that matches your printer.
- Stop Bits, Parity: Choose number of stop bits and parity setting to match your printer.



Then click the button indicated to send the Port 2 configuration to the CPU, and click Close. See Chapter 3 for port wiring information to connect your printer to the D2-250-1, D2-260 and D2-262 CPUs.

Chapter 5: Standard RLL Instructions

Port 2 on the D2-250-1, D2-260 and D2-262 CPUs has standard RS232 levels, and should work with most printer serial input connections.

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

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

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

Example:

" " Length 0 without character

"A" Length 1 with character A

" " Length 1 with blank

" \$ " " Length 1 with double quotation mark

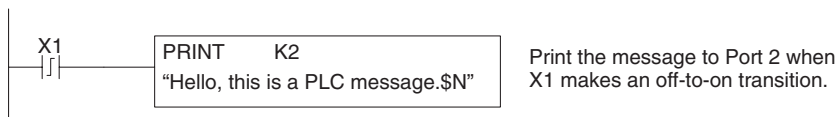
" \$ R \$ L " Length 2 with one CR and one LF

" \$ 0 D \$ 0 A " Length 2 with one CR and one LF

" \$ \$ " Length 1 with one \$ mark

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

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



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



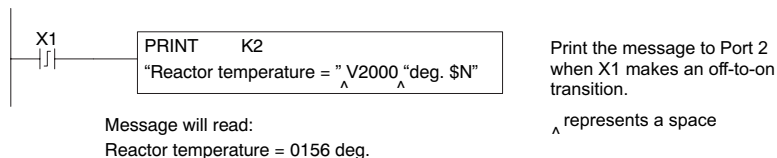
NOTE: There must be a space entered before and after the V-memory address to separate it from the text string. Failure to do this will result in an error code 499.

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

Example:

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

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



V-memory text element – used for printing text stored in V-memory. Use the % followed by the number of characters after V-memory number for representing the text. If you assign “0” as the number of characters, the print function will read the character count from the first location. Then it will start at the next V-memory location and read that number of ASCII codes for the text from memory.

Example:

V2000 % 16 16 characters in V2000 to V2007 are printed.
V2000 % 0 The characters in V2001 to Vxxxx (determined by the number in V2000) will be printed.

Chapter 5: Standard RLL Instructions

Bit element – used for printing the state of the designated bit in V-memory or a relay bit. The bit element can be assigned by the designating point (.) and bit number preceded by the V-memory number or relay number. The output type is described as shown in the table below.

#	Data format	Description
1	none	Print 1 for an ON state, and 0 for an OFF state
2	: BOOL	Print "TRUE" for an ON state, and "FALSE" for an OFF state
3	: ONOFF	Print "ON" for an ON state, and "OFF" for an OFF state

Example:

V2000.15 Prints the status of bit 15 in V2000, in 1/0 format

C100 Prints the status of C100 in 1/0 format

C100 : BOOL Prints the status of C100 in TRUE/FALSE format

C100 : ON/OFF Prints the status of C100 in ON/OFF format

V2000.15 : BOOL Prints the status of bit 15 in V2000 in TRUE/FALSE format

The maximum numbers of characters you can print is 128. The number of characters for each element is listed in the table below:

The Handheld Programmer's mnemonic is "PRINT," followed by the DEF field.

Element type	Maximum Characters
Text, 1 character	1
16-bit binary	6
32-bit binary	11
4-digit BCD	4
8-digit BCD	8
Floating point (real number)	13
Floating point (real with exponent)	13
V-memory/text	2
Bit (1/0 format)	1
Bit (TRUE/FALSE format)	5
Bit (ON/OFF format)	3

Special relay flags SP116 and SP117 indicate the status of the D2-250-1, D2-260 and D2-262 ports (busy, or communications error). See the appendix on special relays for a description.



NOTE: You must use the appropriate special relay in conjunction with the PRINT command to ensure the ladder program does not try to PRINT to a port that is still busy from a previous PRINT or WX or RX instruction.

Modbus RTU Instructions (D2-260/D2-262)

Modbus Read from Network (MRX)

	230
	240
	250-1
	260
	262

DS	Used
HPP	N/A

The Modbus Read from Network (MRX) instruction is used by the D2-260 and D2-262 network master to read a block of data from a connected slave device and to write the data into V-memory addresses within the master. The instruction allows the user to specify the Modbus Function Code, slave station address, starting master and slave memory addresses, number of elements to transfer, Modbus data format and the Exception Response Buffer.

- Port Number: must be D2-260/D2-262 Port 2 (K2)
- Slave Address: specify a slave station address (1 to 247)
- Function Code: The following Modbus function codes are supported by the MRX instruction:
 - 01 – Read a group of coils
 - 02 – Read a group of inputs
 - 03 – Read holding registers
 - 04 – Read input registers
 - 07 – Read Exception status
- Start Slave Memory Address: specifies the starting slave memory address of the data to be read. See the table on the following page.
- Start Master Memory Address: specifies the starting memory address in the master where the data will be placed. See the table on the following page.
- Number of Elements: specifies how many coils, inputs, holding registers or input registers will be read. See the table on the following page.
- Modbus Data Format: specifies Modbus 584/984 or 484 data format to be used.

- Exception Response Buffer: specifies the master memory address where the Exception Response will be placed (6 bytes in length). See the table on the following page. The exception response buffer uses 3 words. These bytes are swapped in the MRX/MWX exception response buffer V-memory so:
- V-Memory 1 Hi Byte = Function Code Byte (Most Significant Bit Set)
- V-Memory 1 Lo Byte = Address Byte
- V-Memory 2 Hi Byte = One of the CRC Bytes
- V-Memory 2 Lo Byte = Exception Code
- V-Memory 3 Hi Byte = 0
- V-Memory 3 Lo Byte = Other CRC Byte

MRX Slave Memory Address

MRX Slave Address Ranges		
Function Code	Modbus Data Format	Slave Address Range(s)
01-Read Coil	484 Mode	1-999
01-Read Coil	584/984 Mode	1-65535
02-Read Input Status	484 Mode	1001-1999
02-Read Input Status	584/984 Mode	10001-19999 (5 digit) or 100001-165535 (6 digit)
03-Read Holding Register	484 Mode	4001-4999
03-Read Holding Register	584/984	40001-49999 9 (5 digit) or 4000001-465535 (6 digit)
04-Read Input Register	484 Mode	3001-3999
04-Read Input Register	584/984 Mode	30001-39999 (5 digit) or 3000001-365535 (6 digit)
07-Read Exception Status	484 and 584/984 Mode	n/a

MRX Master Memory Address Ranges		
Operand Data Type		D2-260/D2-262 Range
Inputs	X	0-1777
Outputs	Y	0-1777
Control Relays	C	0-3777
Stage Bits	S	0-1777
Timer Bits	T	0-377
Counter Bits	CT	0-377
Special Relays	SP	0-777
V-memory	V	all (see page 3-57)
Global Inputs	GX	0-3777
Global Outputs	GY	0-3777

MRX Master Memory Addresses

MRX Number of Elements

Number of Elements		
Operand Data Type		D2-260/D2-262 Range
V-memory	V	All (see page 3-57)
Constant	K	Bits: 1-2000 Registers: 1-125

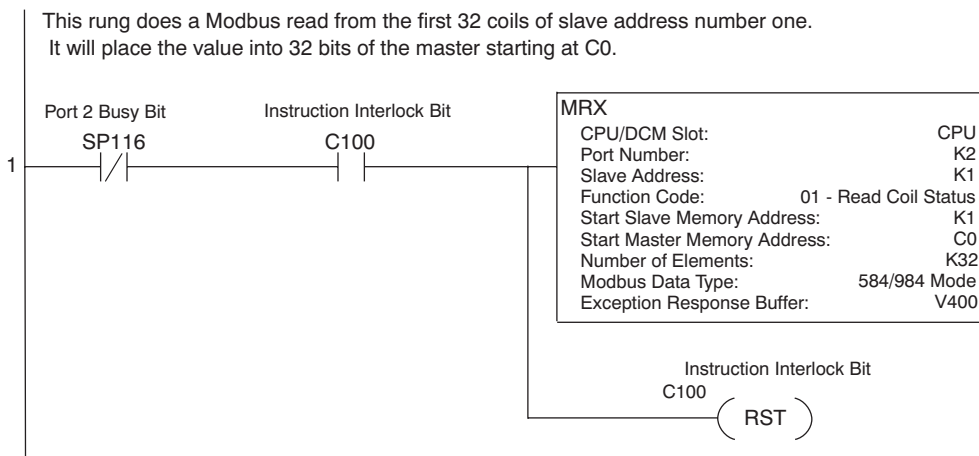
MRX Exception Response Buffer

Exception Response Buffer		
Operand Data Type		D2-260/D2-262 Range
V-memory	V	All (see page 3-57)

MRX Example

D2-260 and D2-262 port 2 has two Special Relay contacts associated with it (see Appendix D for comm port special relays). One indicates "Port busy" (SP116), and the other indicates "Port Communication Error" (SP117). The "Port Busy" bit is on while the PLC communicates with the slave. When the bit is off, the program can initiate the next network request. The "Port Communication Error" bit turns on when the PLC has detected an error. Use of this bit is optional. When used, it should be ahead of any network instruction boxes since the error bit is reset when an MRX or MWX instruction is executed.

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



Modbus Write to Network (MWX)

☒ 230

☒ 240

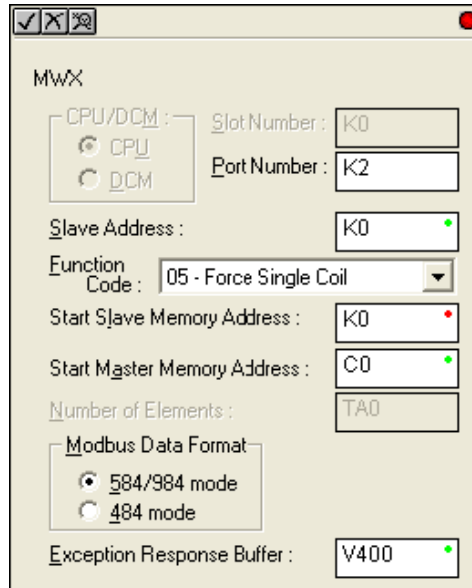
☒ 250-1

☒ 260

☒ 262

DS	Used
HPP	N/A

The Modbus Write to Network (MWX) instruction is used by a D2-262 or D2-262 network master to write a block of data from V-memory to Modbus memory addresses within a slave device on the network. The instruction allows the user to specify the Modbus Function Code, slave station address, starting master and slave memory addresses, number of elements to transfer, Modbus data format and the Exception Response Buffer.



The image shows a software window titled "MWX" for configuring the Modbus Write to Network instruction. It contains the following fields and controls:

- CPU/DCM:** Radio buttons for "CPU" (selected) and "DCM".
- Slot Number:** Text field with value "K0".
- Port Number:** Text field with value "K2".
- Slave Address:** Text field with value "K0" and a green status indicator.
- Function Code:** Dropdown menu showing "05 - Force Single Coil".
- Start Slave Memory Address:** Text field with value "K0" and a red status indicator.
- Start Master Memory Address:** Text field with value "C0" and a green status indicator.
- Number of Elements:** Text field with value "TA0".
- Modbus Data Format:** Radio buttons for "584/984 mode" (selected) and "484 mode".
- Exception Response Buffer:** Text field with value "V400" and a green status indicator.

- Port Number: must be D2-260 or D2-262 Port 2 (K2)
- Slave Address: specify a slave station address (0 to 247)
- Function Code: The following Modbus function codes are supported by the MWX instruction:
 - 05 – Force Single coil
 - 06 – Preset Single Register
 - 15 – Force Multiple Coils
 - 16 – Preset Multiple Registers
- Start Slave Memory Address: specifies the starting slave memory address where the data will be written
- Start Master Memory Address: specifies the starting address of the data in the master that is to be written to the slave
- Number of Elements: specifies how many consecutive coils or registers will be written to. This field is only active when either function code 15 or 16 is selected
- Modbus Data Format: specifies Modbus 584/984 or 484 data format to be used

- Exception Response Buffer: specifies the master memory address where the Exception Response will be placed (6-bytes in length). See the table on the following page. The exception response buffer uses 3 words. These bytes are swapped in the MRX/MWX exception response buffer V-memory so:
- V-Memory 1 Hi Byte = Function Code Byte (Most Significant Bit Set)
- V-Memory 1 Lo Byte = Address Byte
- V-Memory 2 Hi Byte = One of the CRC Bytes
- V-Memory 2 Lo Byte = Exception Code
- V-Memory 3 Hi Byte = 0
- V-Memory 3 Lo Byte = Other CRC Byte

MWX Slave Memory Address

MWX Slave Address Ranges		
Function Code	Modbus Data Format	Slave Address Range(s)
05-Force Single Coil	484 Mode	1-999
05-Force Single Coil	584/984 Mode	1-65535
06-Preset Single Register	484 Mode	4001-4999
06-Preset Single Register	584/984 Mode	40001-49999 (5 digit) or 400001-465535 (6 digit)
15-Force Multiple Coils	484 Mode	1-999
15-Force Multiple Coils	584/984 Mode	1-65535
16-Preset Multiple Registers	484 Mode	4001-4999
16-Preset Multiple Registers	584/984 Mode	40001-49999 (5 digit) or 400001-465535 (6 digit)

MWX Master Memory Addresses

MRX Master Memory Address Ranges		
Operand Data Type		D2-260/D2-262 Range
Inputs	X	0-1777
Outputs	Y	0-1777
Control Relays	C	0-3777
Stage Bits	S	0-1777
Timer Bits	T	0-377
Counter Bits	CT	0-377
Special Relays	SP	0-777
V-memory	V	all (see page 3-57)
Global Inputs	GX	0-3777
Global Outputs	GY	0-3777

MWX Number of Elements

Number of Elements		
Operand Data Type		D2-260/D2-262 Range
V-memory	V	all (see page 3-57)
Constant	K	Bits: 1-2000 Registers: 1-125

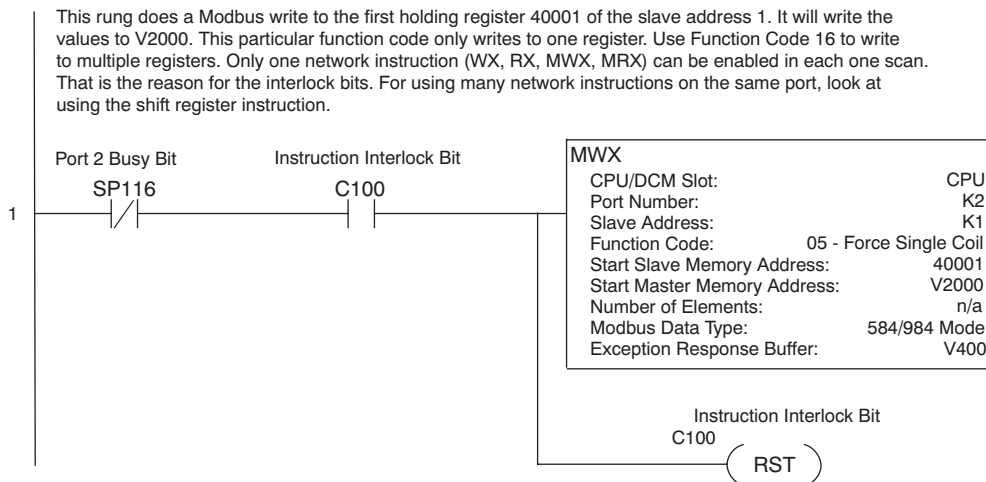
MWX Exception Response Buffer

Exception Response Buffer		
Operand Data Type		D2-260/D2-262 Range
V-memory	V	all (see page 3-57)

MWX Example

D2-260 and D2-262 port 2 has two Special Relay contacts associated with it (see Appendix D for comm port special relays). One indicates "Port busy" (SP116), and the other indicates "Port Communication Error" (SP117). The "Port Busy" bit is on while the PLC communicates with the slave. When the bit is off, the program can initiate the next network request. The "Port Communication Error" bit turns on when the PLC has detected an error. Use of this bit is optional. When used, it should be ahead of any network instruction boxes since the error bit is reset when an MRX or MWX instruction is executed.

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



ASCII Instructions (D2-260/D2-262)

 230

The D2-260 and D2-262 CPU supports several instructions and methods that allow ASCII strings to be read into and written from the PLC communications ports.

 240

 250-1

Specifically, port 2 on the D2-260 and D2-262 can be used for either reading or writing raw ASCII strings, but cannot be used for both on the same CPU.

 260

 262

The D2-260 and D2-262 can also decipher ASCII embedded within a supported protocol (K-Sequence, DirectNET, Modbus, Ethernet) via the CPU ports, H2-ECOM or D2-DCM module.

DS	Used
HPP	N/A

ASCII character tables and descriptions can be found at www.asciitable.com.

Reading ASCII Input Strings

There are several methods which the D2-260 and D2-262 can use to read ASCII input strings:

- 1) ASCII IN (AIN) – This instruction configures port 2 for raw ASCII input strings with parameters such as fixed and variable length ASCII strings, termination characters, byte swapping options, and instruction control bits. Use barcode scanners, weight scales, etc., to write raw ASCII input strings into port 2 based on the (AIN) instruction's parameters.
- 2) Write embedded ASCII strings directly to V-memory from an external HMI or similar master device via a supported communications protocol using the CPU ports, H2-ECOM or D2-DCM module. The AIN instruction is not used in this case.
- 3) If a D2-260 or a D2-262 PLC is a master on a network, the Network Read instruction (RX) can be used to read embedded ASCII data from a slave device via a supported communications protocol using port 2, H2-ECOM or D2-DCM module. The RX instruction places the data directly into V-memory.

Writing ASCII Output Strings

The following instructions can be used to write ASCII output strings:

- 1) Print from V-memory (PRINTV) – Use this instruction to write raw ASCII strings out of port 2 to a display panel or a serial printer, etc. The instruction features the starting V-memory address, string length, byte swapping options, etc. When the instruction's permissive bit is enabled, the string is written to port 2.
- 2) Print to V-memory (VPRINT) – Use this instruction to create pre-coded ASCII strings in the PLC (i.e. alarm messages). When the instruction's permissive bit is enabled, the message is loaded into a pre-defined V-memory address location. Then use the PRINTV instruction to write the pre-coded ASCII string out of port 2. American, European and Asian Time/Date stamps are supported.

Additionally, if a D2-260 or a D2-262 PLC is a master on a network, the Network Write instruction (WX) can be used to write embedded ASCII data to an HMI or slave device directly from V-memory via a supported communications protocol using port 2, H2-ECOM or D2-DCM module.

Managing the ASCII Strings

The following instructions can be helpful in managing the ASCII strings within the CPU's V-memory:

ASCII Find (AFIND) – Finds where a specific portion of the ASCII string is located in continuous V-memory addresses. Forward and reverse searches are supported.

ASCII Extract (AEX) – Extracts a specific portion (usually some data value) from the ASCII find location or other known ASCII data location.

Compare V-memory (CMPV) – This instruction is used to compare two blocks of V-memory addresses and is usually used to detect a change in an ASCII string. Compared data types must be of the same format (i.e., BCD, ASCII, etc.).

Swap Bytes (SWAPB) – usually used to swap V-memory bytes on ASCII data that was written directly to V-memory from an external HMI or similar master device via a communications protocol. The AIN and AEX instructions have a built-in byte swap feature.

ASCII Input (AIN)

The ASCII Input instruction allows the CPU to receive ASCII strings through the specified communications port and places the string into a series of specified V-memory registers. The ASCII data can be received as a fixed number of bytes or as a variable length string with a specified termination character(s). Other features include Byte Swap preferences, Character Timeout, and user-defined flag bits for Busy, Complete and Timeout Error.

✗ 230

✗ 240

✗ 250-1

✓ 260

✓ 262

DS	Used
HPP	N/A

AIN

Length Type: ☒ Fixed Length ☐ Variable Length

CPU/DCM: ☒ CPU ☐ DCM

Byte Swap: ☒ None ☐ All ☐ All but null

Termination Code Length: ☒ 1 Character ☐ 2 Characters

Slot Number: K0

Port Number: K2

Data Destination: V2000

* Data Destination = Byte count
* Data Destination + 1 = Start of data

Fixed Length: K32

Interchar. Timeout: None

First Char. Timeout: None

Overflow Error: C0

Busy: C0

Complete: C1

Interchar. T/O Error: K0

First Char. T/O Error: K0

AIN Fixed Length Configuration

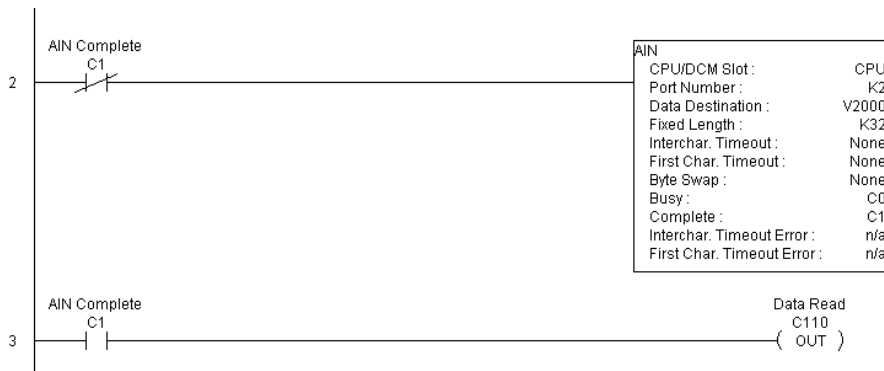
- Length Type: select fixed length based on the length of the ASCII string that will be sent to the CPU port.
- Port Number: must be D2-260/D2-262 port 2 (K2).
- Data Destination: specifies where the ASCII string will be placed in V-memory.
- Fixed Length: specifies the length, in bytes, of the fixed-length ASCII string the port will receive.
- Inter-character Timeout: if the amount of time between incoming ASCII characters exceeds the set time, the specified Timeout Error bit will be set. No data will be stored at the Data Destination V-memory location. The bit will reset when the AIN instruction permissive bits are disabled. None selection disables this feature.
- First Character Timeout: if the amount of time from when the AIN is enabled to the time the first character is received exceeds the set time, the specified First Character Timeout bit will be set. The bit will reset when the AIN instruction permissive bits are disabled. None selection disables this feature.
- Byte Swap: swaps the high-byte and low-byte within each V-memory register of the Fixed Length ASCII string. See the SWAPB instruction for details.
- Busy Bit: is ON while the AIN instruction is receiving ASCII data.
- Complete Bit: is set once the ASCII data has been received for the specified fixed length and reset when the AIN instruction permissive bits are disabled.
- Inter-character Timeout Error Bit: is set when the Character Timeout is exceed. See Character Timeout explanation above.
- First Character Timeout Error Bit: is set when the First Character Timeout is exceeded. See First Character Timeout explanation above.

Parameter	D2-260/D2-262 Range
Data Destination	All V-memory (See page 3 -56)
Fixed Length	K1-128
Bits: Busy, Complete, Timeout Error, Overflow	C0-3777

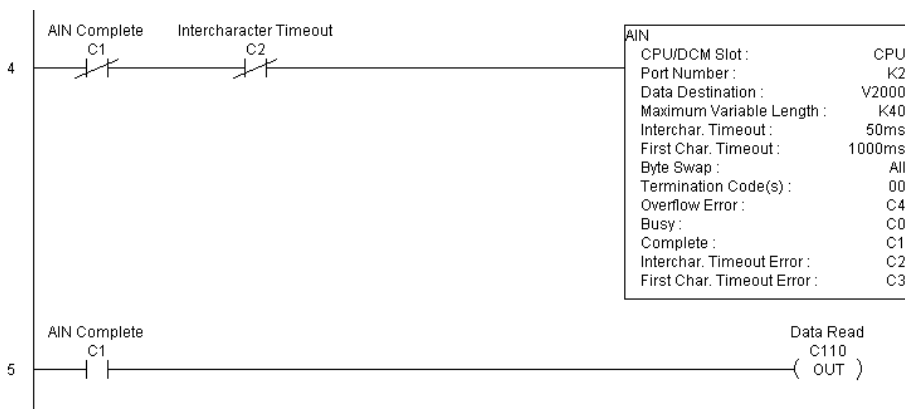
Discrete Bit Flags	Description
SP53	On if the CPU cannot execute the instruction
SP71	On when a value used by the instruction is invalid
SP116	On when CPU port 2 is communicating with another device
SP117	On when CPU port 2 has experienced a communication error

AIN Fixed Length Examples

Fixed Length example when the PLC is reading the port continuously and timing is not critical.



Fixed Length example when character to character timing is critical.



AIN

Length Type:
☐ Fixed Length
☒ Variable Length

CPU/DCM:
☒ CPU
☐ DCM

Byte Swap:
☐ None
☐ All
☒ All but null

Slot Number: K0

Port Number: K2

Data Destination: V2000
* Data Destination = Byte count
 * Data Destination + 1 = Start of data

Maximum Variable Length: K40

Interchar. Timeout: 100 ms

First Char. Timeout: 2000 ms

Termination Code Length:
☒ 1 Character
☐ 2 Characters

TermCode 1: 0D hexadecimal

TermCode 2: 0D hexadecimal

Overflow Error: C4

Busy: C0

Complete: C1

Interchar. T/O Error: C2

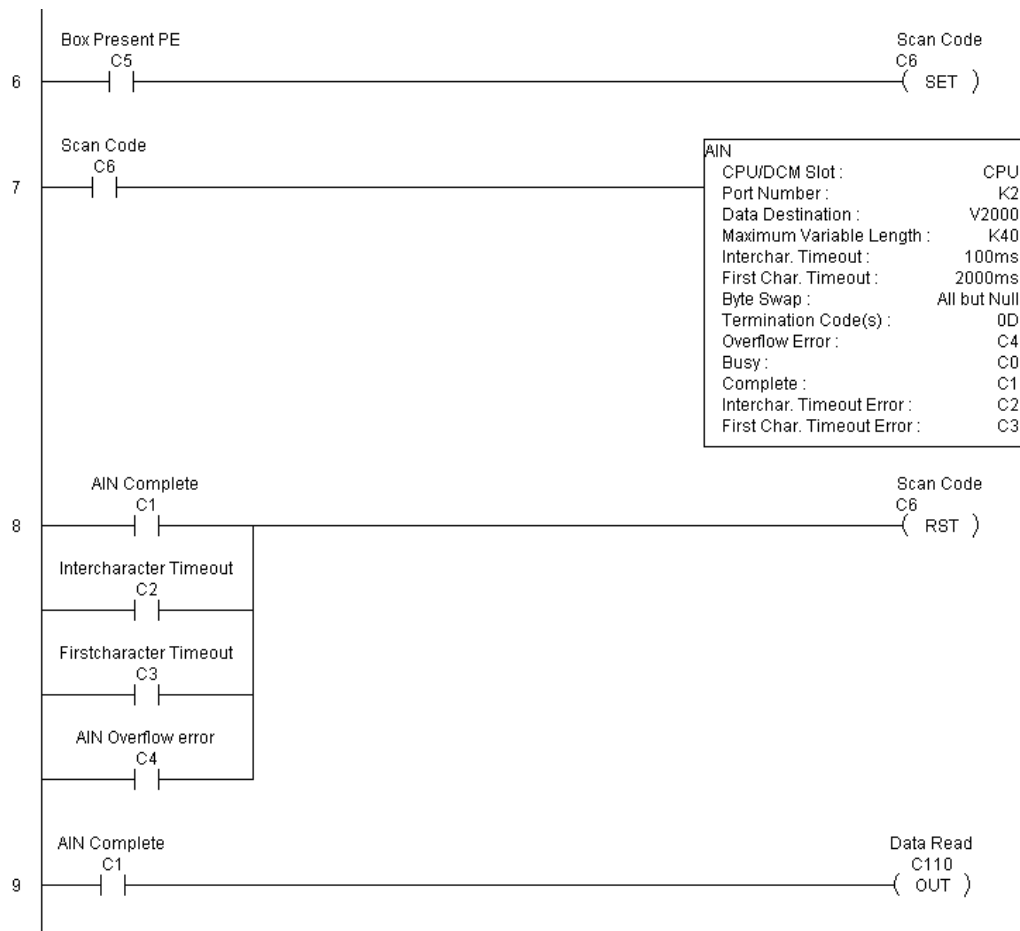
First Char. T/O Error: C3

- AIN Variable Length Configuration:
- Length Type: select Variable Length if the ASCII string length followed by termination characters will vary in length.
- Port Number: must be D2-260/D2-262 port 2 (K2).
- Data Destination: specifies where the ASCII string will be placed in V-memory.
- Maximum Variable Length: specifies, in bytes, the maximum length of a Variable Length ASCII string the port will receive.
- Inter-character Timeout: if the amount of time between incoming ASCII characters exceeds the set time, the Timeout Error bit will be set. No data will be stored at the Data Destination V-memory location. The Timeout Error bit will reset when the AIN instruction permissive bits are disabled. None selection disables this feature.
- First Character Timeout: if the amount of time from when the AIN is enabled to the time the first character is received exceeds the set time, the specified First Character Timeout bit will be set. The bit will reset when the AIN instruction permissive bits are disabled. None selection disables this feature.
- Byte Swap: swaps the high-byte and low-byte within each V-memory register of the Variable Length ASCII string. See the SWAPB instruction for details.
- Termination Code Length: consists of either 1 or 2 characters. Refer to the ASCII table in Appendix G.
- Overflow Error Bit: is set when the ASCII data received exceeds the Maximum Variable Length specified.
- Busy Bit: is ON while the AIN instruction is receiving ASCII data.
- Complete Bit: is set once the ASCII data has been received up to the termination code characters. It will be reset when the AIN instruction permissive bits are disabled.
- Inter-character Timeout Error Bit: is set when the Character Timeout is exceed. See Character Timeout explanation above.
- First Character Timeout Error Bit: is set when the First Character Timeout is exceeded. See First Character Timeout explanation above.

Parameter	D2-260/D2-262 Range
Data Destination	All V-memory (See page 3-57)
Max. Variable Length	K1-128
Bits: Busy, Complete, Timeout Error, Overflow	C0-3777

AIN Variable Length Example

AIN Variable Length example used to read barcodes on boxes (PE = photoelectric sensor).



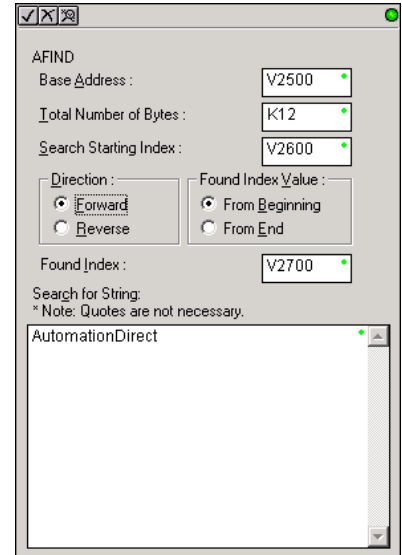
ASCII Find (AFIND)

	230
	240
	250-1
	260
	262

DS	Used
HPP	N/A

The ASCII Find instruction locates a specific ASCII string or portion of an ASCII string within a range of V-memory registers and places the string's Found Index number (byte number where desired string is found) in Hex, into a specified V-memory register. Other features include, Search Starting Index number for skipping over unnecessary bytes before beginning the FIND operation, Forward or Reverse direction search, and From Beginning and From End selections to reference the Found Index Value.

- **Base Address:** specifies the beginning V-memory register where the entire ASCII string is stored in memory.
- **Total Number of Bytes:** specifies the total number of bytes to search for the desired ASCII string.
- **Search Starting Index:** specifies which byte to skip to (with respect to the Base Address) before beginning the search.
- **Direction:** Forward begins the search from lower numbered V-memory registers to higher numbered V-memory registers. Reverse does the search from higher numbered V-memory registers to lower-numbered V-memory registers.
- **Found Index Value:** specifies whether the Beginning or the End byte of the ASCII string found will be loaded into the Found Index register.
- **Found Index:** specifies the V-memory register where the Found Index Value will be stored. A value of FFFF will result if the desired string is not located in the memory registers specified. A value of EEEE will result if there is a conflict in the AFIND search parameters specified.



AFIND

Base Address : V2500

Total Number of Bytes : K12

Search Starting Index : V2600

Direction : ☒ Forward ☐ Reverse

Found Index Value : ☒ From Beginning ☐ From End

Found Index : V2700

Search for String:
* Note: Quotes are not necessary.

AutomationDirect

Parameter	D2-260/D2-262 Range
Base Address	All V-memory (See page 3-57)
Total Number of Bytes	All V-memory (See page 3-57) or K1-128
Search Starting Index	All V-memory (See page 3-57) or K0-127
Found Index	All V-memory (See page 3-57)

Discrete Bit Flags	Description
SP53	On if the CPU cannot execute the instruction.
SP71	On when a value used by the instruction is invalid.

Search for String: up to 128 characters.

AFIND Search Example

In the following example, the AFIND instruction is used to search for the “day” portion of “Friday” in the ASCII string “Today is Friday,” which had previously been loaded into V-memory. Note that a Search Starting Index of constant (K) 5 combined with a Forward Direction Search is used to prevent finding the “day” portion of the word “Today.” The Found Index will be placed into V4000.

Find

Base Address: V3000

Total Number of Bytes: K16

Search Starting Index: K5

Direction:

- ☒ Forward
- ☐ Reverse

Found Index Value:

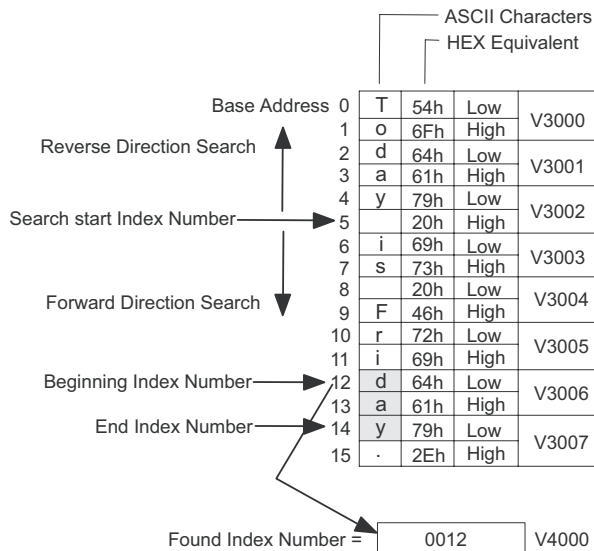
- ☒ From Beginning
- ☐ From End

Found Index: V4000

Search for String:

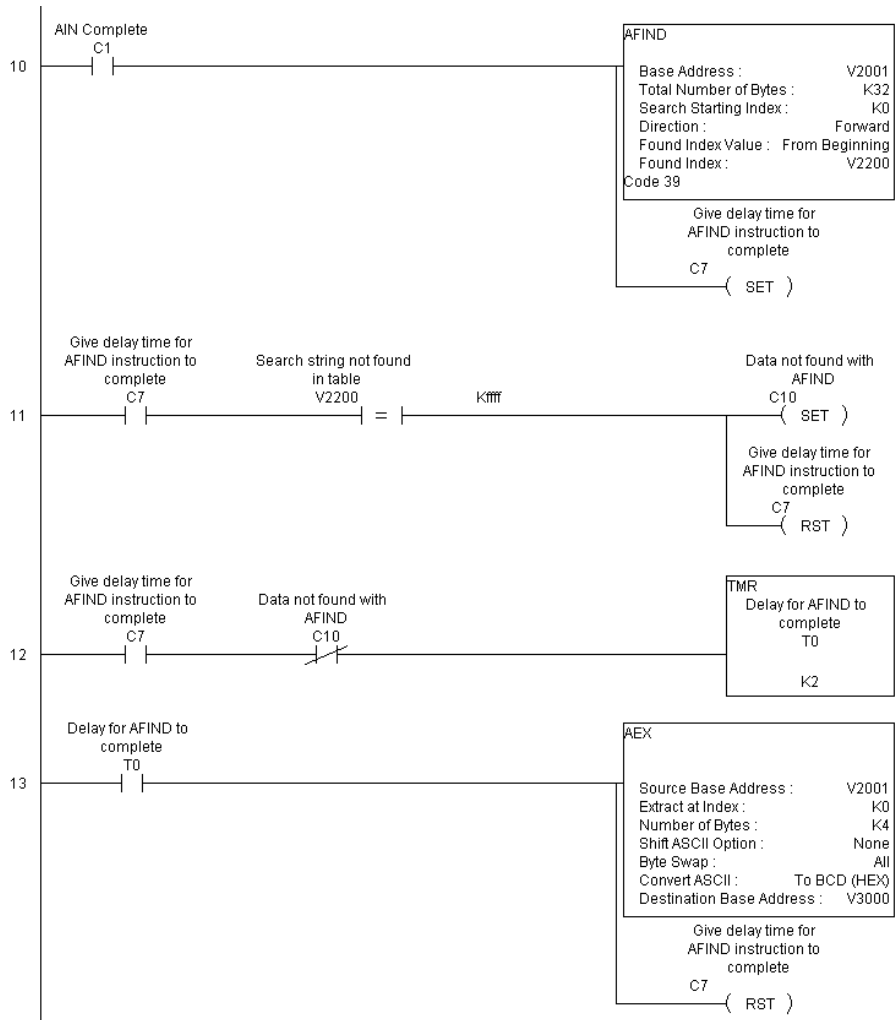
Note: Quotes are not necessary.

day



AFIND Example Combined with AEX Instruction

When an AIN instruction has executed, its Complete bit can be used to trigger an AFIND instruction to search for a desired portion of the ASCII string. Once the string is found, the AEX instruction can be used to extract the located string.



ASCII Extract (AEX)

230

240

250-1

260

262

DS	Used
HPP	N/A

The ASCII Extract instruction extracts a specified number of bytes of ASCII data from one series of V-memory registers and places them into another series of V-memory registers. Other features include Extract at Index for skipping over unnecessary bytes before beginning the Extract operation, Shift ASCII Option, for One Byte Left or One Byte Right, Byte Swap and Convert data to a BCD format number.

- Source Base Address: specifies the beginning V-memory register where the entire ASCII string is stored in memory.
- Extract at Index: specifies which byte to skip to (with respect to the Source Base Address) before extracting the data.
- Number of Bytes: specifies the number of bytes to be extracted.
- Shift ASCII Option: shifts all extracted data one byte left or one byte right to displace "unwanted" characters, if necessary.
- Byte Swap: swaps the high-byte and the low-byte within each V-memory register of the extracted data. See the SWAPB instruction for details.
- Convert BCD(Hex) ASCII to BCD(Hex): if enabled, this will convert ASCII numerical characters to Hexadecimal numerical values.
- Destination Base Address: specifies the V-memory register where the extracted data will be stored.

Parameter	D2-260/D2-262 Range
Source Base Address	All V-memory (See page 3-57)
Extract at Index	All V-memory (See page 3-57) or K0-127
Number of Bytes	K1-128
Destination Base Address	All V-memory (See page 3-57)

Discrete Bit Flags	Description
SP53	On if the CPU cannot execute the instruction.
SP71	On when a value used by the instruction is invalid.

See the previous page for an example using the AEX instruction.

ASCII Compare (CMPV)

230

240

250-1

260

262

The ASCII Compare instruction compares two groups of V-memory registers. The CMPV will compare any data type (ASCII to ASCII, BCD to BCD, etc) of one series (group) of V-memory registers to another series of V-memory registers for a specified byte length.

"Compare from" Starting Address:

specifies the beginning V-memory register of the first group of V-memory registers to be compared from.

"Compare to" Starting Address:

specifies the beginning V-memory register of the second group of V-memory registers to be compared to.

Number of Bytes: specifies the length of each V-memory group to be compared.

CMPV

"Compare from" Starting Address : V3400

"Compare to" Starting Address : V3500

Number of Bytes : K12

SP61 = 1 (ON), the result is equal

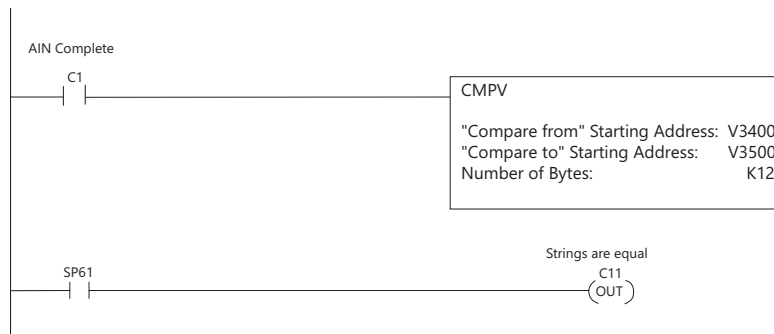
SP61 = 0 (OFF), the result is not equal

Parameter	D2-260/D2-262 Range
Compare from Starting Address	All V-memory (See page 3-57)
Compare to Starting Address	All V-memory (See page 3-57)
Number of Bytes	All V-memory (See page 3-57) or K0-127

Discrete Bit Flags	Description
SP53	On if the CPU cannot execute the instruction.
SP61	On when result is equal.
SP71	On when a value used by the instruction is invalid.

CMPV Example

The CMPV instruction executes when the AIN instruction is complete. If the compared V-memory tables are equal, SP61 will turn ON.



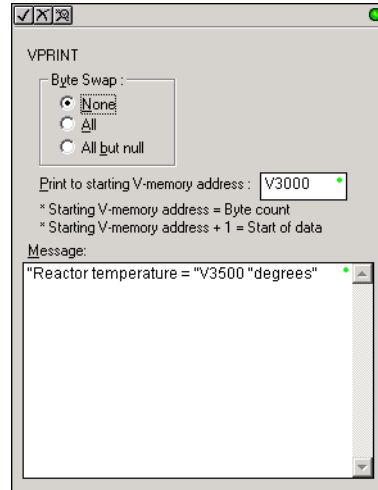
ASCII Print to V-memory (VPRINT)

-  230
-  240
-  250-1
-  260
-  262

DS	Used
HPP	N/A

The ASCII Print to V-memory instruction will write a specified ASCII string into a series of V-memory registers. Other features include Byte Swap, options to suppress or convert leading zeros or spaces, and _Date and _Time options for U.S., European, and Asian date formats and 12- or 24-hour time formats.

- **Byte Swap:** swaps the high-byte and low-byte within each V-memory register to which the ASCII string is printed. See the SWAPB instruction for details.
- **Print to Starting V-memory Address:** specifies the beginning of a series of V-memory addresses where the ASCII string will be placed by the VPRINT instruction.
- **Starting V-memory Address:** the first V-memory register of the series of registers specified will contain the ASCII string's length in bytes.
- **Starting V-memory Address + 1:** the 2nd and subsequent registers will contain the ASCII string printed to V-memory.



Parameter	D2-260/D2-262 Range
Print to Starting V-memory Address	All V-memory (See page 3-57)

Discrete Bit Flags	Description
SP53	On if the CPU cannot execute the instruction.
SP71	On when a value used by the instruction is invalid.

VPRINT Time/Date Stamping

The codes in the table below can be used in the VPRINT ASCII string message to "print to V-memory" the current time and/or date.

#	Character Code	Date/Time Stamp Options
1	<u>_Date:us</u>	American standard (month/day/2 digit year)
2	<u>_Date:e</u>	European standard (day/month/2 digit year)
3	<u>_Date:a</u>	Asian standard (2 digit year/month/day)
4	<u>_Time:12</u>	standard 12 hour clock (0-12 hour:min am/pm)
5	<u>_Time:24</u>	standard 24 hour clock (0-23 hour:min am/pm)

VPRINT V-memory element

The following modifiers can be used in the VPRINT ASCII string message to “print to V-memory” register contents in integer format or real format. Use V-memory number or V-memory number with “:” and data type. The data types are shown in the table below. The Character code must be capital letters.



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

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

Examples:

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

The following modifiers can be added to any of the modifies above to suppress or convert leading zeros or spaces. The character code must be capital letters.

#	Character Code	Description
1	S	Suppresses leading spaces
2	C0	Converts leading spaces to zeros
3	0	Suppresses leading zeros

Example with V2000 = 0018 (binary format)

V-memory Register with Modifier	Number of Characters			
	1	2	3	4
V2000	0	0	1	8
V2000:B	0	0	1	2
V2000:B0	1	2		

Example with V2000 = sp sp18 (binary format) where sp = space

V-memory Register with Modifier	Number of Characters			
	1	2	3	4
V2000	sp	sp	1	8
V2000:B	sp	sp	1	2
V2000:BS	1	2		
V2000:BC0	0	0	1	2

VPRINT V-memory text element

The following is used for “printing to V-memory” text stored in registers. Use the % followed by the number of characters after V-memory number for representing the text. If you assign “0” as the number of characters, the function will read the character count from the first location. Then it will start at the next V-memory location and read that number of ASCII codes for the text from memory.

Example:

V2000 % 16 16 characters in V2000 to V2007 are printed.

V2000 % 0 The characters in V2001 to Vxxxx (determined by the number in V2000) will be printed.

VPRINT Bit element

The following is used for “printing to V-memory” the state of the designated bit in V-memory or a control relay bit. The bit element can be assigned by the designating point (.) and bit number preceded by the V-memory number or relay number. The output type is described as shown in the table below.

#	Data format	Description
1	none	Print 1 for an ON state, and 0 for an OFF state
2	: BOOL	Print “TRUE” for an ON state, and “FALSE” for an OFF state
3	: ONOFF	Print “ON” for an ON state, and “OFF” for an OFF state

Example:

V2000.15 Prints the status of bit 15 in V2000, in 1/0 format

C100 Prints the status of C100 in 1/0 format

C100 : BOOL Prints the status of C100 in TRUE/FALSE format

C100 : ON/OFF Prints the status of C100 in ON/OFF format

V2000.15 : BOOL Prints the status of bit 15 in V2000 in TRUE/FALSE format

The maximum numbers of characters you can VPRINT is 128. The number of characters required for each element, regardless of whether the :S, :C0 or :0 modifiers are used, is listed in the table below.

Element Type	Maximum Characters
Text, 1 character	1
16-bit binary	6
32-bit binary	11
4-digit BCD	4
8-digit BCD	8
Floating point (real number)	13
Floating point (real with exponent)	13
V-memory/text	2
Bit (1/0 format)	1
Bit (TRUE/FALSE format)	5
Bit (ON/OFF format)	3

Text element

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

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

Chapter 5: Standard RLL Instructions

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

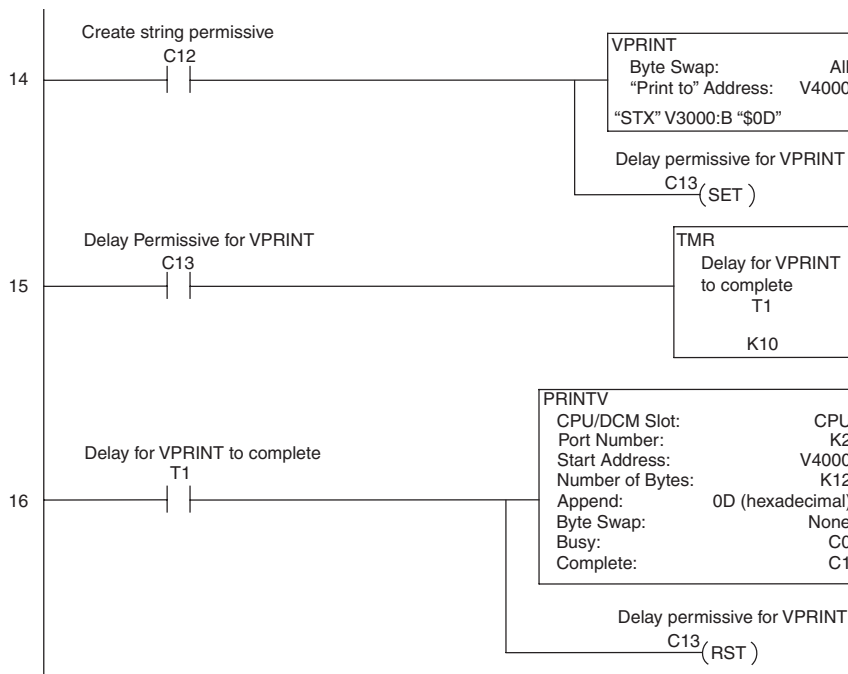
Example:

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

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

VPRINT Example Combined with PRINTV Instruction

The VPRINT instruction is used to create a string in V-memory. The PRINTV is used to print the string out of port 2.



ASCII Print from V-memory (PRINTV)

230

240

250-1

260

262

DS	Used
HPP	N/A

The ASCII Print from V-memory instruction will send an ASCII string out of the designated communications port from a specified series of V-memory registers for a specified length in number of bytes. Other features include user specified Append Characters to be placed after the desired data string for devices that require specific termination character(s), Byte Swap options, and user specified flags for Busy and Complete.

- Port Number: must be D2-260/D2-262 port 2 (K2).
- Start Address: specifies the beginning of series of V-memory registers that contain the ASCII string to print.
- Number of Bytes: specifies the length of the string to print.
- Append Characters: specifies ASCII characters to be added to the end of the string for devices that require specific termination characters.
- Byte Swap: swaps the high-byte and low-byte within each V-memory register of the string while printing. See the SWAPB instruction for details.
- Busy Bit: will be ON while the instruction is printing ASCII data.
- Complete Bit: will be set once the ASCII data has been printed and reset when the PRINTV instruction permissive bits are disabled.

Parameter	D2-260/D2-262 Range
Port Number	port 2 (K2)
Start Address	All V-memory (See page 3-57)
Number of Bytes	All V-memory (See page 3-57) or K1-128
Bits: Busy, Complete	C0-3777

Discrete Bit Flags	Description
SP53	On if the CPU cannot execute the instruction.
SP71	On when a value used by the instruction is invalid.
SP116	On when CPU port 2 is communicating with another device.
SP117	On when CPU port 2 has experienced a communication error.

See the facing page for an example using the PRINTV instruction.

ASCII Swap Bytes (SWAPB)

230

240

250-1

260

262

DS	Used
HPP	N/A

The ASCII Swap Bytes instruction swaps byte positions (high-byte to low-byte and low-byte to high-byte) within each V-memory register of a series of V-memory registers for a specified number of bytes.

- Starting Address: specifies the beginning of a series of V-memory registers the instruction will use to begin byte swapping
- Number of Bytes: specifies the number of bytes, beginning with the Starting Address, to byte swap

Parameter	D2-260/D2-262 Range
Starting Address	All V-memory (See page 3-57)
Number of Bytes	All V-memory (See page 3-57) or K1 to 128

Discrete Bit Flags	Description
SP53	On if the CPU cannot execute the instruction.
SP71	On when a value used by the instruction is invalid.

Byte Swap Preferences

No Byte Swapping
(AIN, AEX, PRINTV, VPRINT)

A B C D E



	Byte High	Low
V2000	0005h	
V2001	B	A
V2002	D	C
V2003	xx	E

Byte Swap All

A B C D E
B A D C E



	Byte High	Low
V2000	0005h	
V2001	A	B
V2002	C	D
V2003	E	xx

Byte Swap All but Null

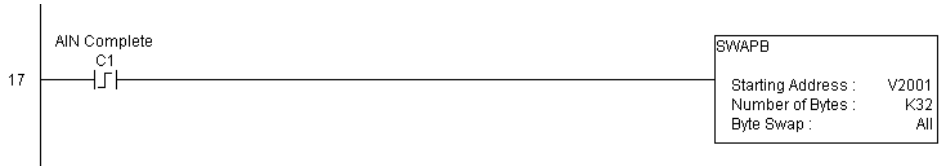
A B C D E
B A D C E



	Byte High	Low
V2000	0005h	
V2001	A	B
V2002	C	D
V2003	xx	E

SWAPB Example

The AIN Complete bit is used to trigger the SWAPB instruction. Use a one-shot so the SWAPB only executes once.



ASCII Clear Buffer (ACRB)

- ☒ 230 The ASCII Clear Buffer instruction will clear the ASCII receive buffer of the specified communications port number.
- ☒ 240
- ☒ 250-1 Port Number: must be D2-260/D2-262 port 2 (K2)

☒ 260

☒ 262

ACRB Example

The AIN Complete bit or the AIN diagnostic bits are used to clear the ASCII buffer.

DS	Used
HPP	N/A

☒ ☒ ☒

ACRB

CPU/DCM :

☒ CPU
☐ DCM

Slot Number : K32

Port Number : K2



Intelligent Box (IBox) Instructions (D2-250-1, D2-260 and D2-262 Only)

A new class of instructions, called Ibox Instructions, became available with the introduction of DirectSOFT. These powerful, yet easy-to-use instructions simplify many of the more complicated tasks that could previously be accomplished only through the use of multiple RLL Instructions. The IBox Instructions are supported by D2-250-1, D2-260 and D2-262 PLCs. The D2-250-1 CPU requires firmware version v4.60 or later, while the D2-260 and D2-262 CPUs requires firmware version v1.0 or later. For more information on DirectSOFT or to download our free version, please visit our Web site at: www.automationdirect.com.

Analog Helper IBoxes		
Instruction	IBox #	Page
Analog Input / Output Combo Module Pointer Setup (ANLGCMB)	IB-462	5-233
Analog Input Module Pointer Setup (ANLGIN)	IB-460	5-235
Analog Output Module Pointer Setup (ANLGOUT)	IB-461	5-237
Analog Scale 12-Bit BCD to BCD (ANSCL)	IB-423	5-239
Analog Scale 12-Bit Binary to Binary (ANSCLB)	IB-403	5-240
Filter Over Time-BCD (FILTER)	IB-422	5-241
Filter Over Time-Binary (FILTERB)	IB-402	5-243
Hi/Low Alarm-BCD (HILOAL)	IB-421	5-245
Hi/Low Alarm-Binary (HILOALB)	IB-401	5-247

Discrete Helper IBoxes		
Instruction	Ibox #	Page
Off Delay Timer (OFFDTMR)	IB-302	5-249
On Delay Timer (ONDTMR)	IB-301	5-251
One Shot (ONESHOT)	IB-303	5-253
Push On / Push Off Circuit (PONOFF)	IB-300	5-254

Memory IBoxes		
Instruction	Ibox #	Page
Move Single Word (MOVEW)	IB-200	5-255
Move Double Word (MOVED)	IB-201	5-256



NOTE: Check your CPU firmware version using DirectSOFT: PLC Menu > Diagnostics > System Information. The latest firmware and update tool are available from: <http://support.automationdirect.com/firmware/index.html>

Intelligent Box (IBox) Instructions, Continued (D2-250-1, D2-260 and D2-262 Only)

Math IBoxes		
Instruction	Ibox #	Page
BCD to Real with Implied Decimal Point (BCDTOR)	IB-560	5-257
Double BCD to Real with Implied Decimal Point (BCDTORD)	IB-562	5-258
Math-BCD (MATHBCD)	IB-521	5-259
Math-Binary (MATHBIN)	IB-501	5-261
Math-Real (MATHR)	IB-541	5-263
Real to BCD with Implied Decimal Point and Rounding (RTOBCD)	IB-561	5-264
Real to Double BCD with Implied Decimal Point and Rounding (RTOBCDD)	IB-563	5-265
Square BCD (SQUARE)	IB-523	5-266
Square Binary (SQUAREB)	IB-503	5-267
Square Real(SQUARER)	IB-543	5-268
Sum BCD Numbers (SUMBCD)	IB-522	5-269
Sum Binary Numbers (SUMBIN)	IB-502	5-270
Sum Real Numbers (SUMR)	IB-542	5-271

Communication IBoxes		
Instruction	Ibox #	Page
ECOM100 Configuration (ECOM100)	IB-710	5-273
ECOM100 Disable DHCP (ECDHCPD)	IB-736	5-275
ECOM100 Enable DHCP (ECDHCPE)	IB-735	5-277
ECOM100 Query DHCP Setting (ECDHCPQ)	IB-734	5-279
ECOM100 Send E-mail (ECEMAIL)	IB-711	5-281
ECOM100 Restore Default E-mail Setup (ECEMRDS)	IB-713	5-287
ECOM100 E-mail Setup (ECEMSUP)	IB-712	5-290
ECOM100 IP Setup (ECIPSUP)	IB-717	5-294
ECOM100 Read Description (ECRDDES)	IB-726	5-296
ECOM100 Read Gateway Address (ECRDGWA)	IB-730	5-298
ECOM100 Read IP Address (ECDRIP)	IB-722	5-300
ECOM100 Read Module ID (ECDRMID)	IB-720	5-302
ECOM100 Read Module Name (ECRDNAM)	IB-724	5-304
ECOM100 Read Subnet Mask (ECRDSNM)	IB-732	5-306
ECOM100 Write Description (ECWRDES)	IB-727	5-308
ECOM100 Write Gateway Address (ECWRGWA)	IB-731	5-310
ECOM100 Write IP Address (ECWRIP)	IB-723	5-312
ECOM100 Write Module ID (ECWRMID)	IB-721	5-314
ECOM100 Write Name (ECWRNAM)	IB-725	5-316
ECOM100 Write Subnet Mask (ECWRSNM)	IB-733	5-318
ECOM100 RX Network Read (ECRX)	IB-740	5-320
ECOM100 WX Network Write (ECWX)	IB-741	5-323
NETCFG Network Configuration (NETCFG)	IB-700	5-326
Network RX Read (NETRX)	IB-701	5-328
Network WX Write (NETWX)	IB-702	5-331

Intelligent Box (IBox) Instructions, Continued (D2-250-1, D2-260 and D2-262 Only)

Counter I/O IBoxes (Work with H2-CTRIO and H2-CTRIO2)		
Instruction	Ibox #	Page
CTRIO Configuration (CTRIO)	IB-1000	5-334
CTRIO Add Entry to End of Preset Table (CTRADPT)	IB-1005	5-336
CTRIO Clear Preset Table (CTRCLRT)	IB-1007	5-339
CTRIO Edit Preset Table Entry (CTREDPT)	IB-1003	5-342
CTRIO Edit Preset Table Entry and Reload (CTREDRL)	IB-1002	5-346
CTRIO Initialize Preset Table (CTRINPT)	IB-1004	5-350
CTRIO Initialize Preset Table (CTRINTR)	IB-1010	5-354
CTRIO Load Profile (CTRLDPR)	IB-1001	5-358
CTRIO Read Error (CTRRDER)	IB-1014	5-361
CTRIO Run to Limit Mode (CTRRRLM)	IB-1011	5-363
CTRIO Run to Position Mode (CTRRTPM)	IB-1012	5-366
CTRIO Velocity Mode (CTRVELO)	IB-1013	5-369
CTRIO Write File to ROM (CTRWFTFR)	IB-1006	5-372



NOTE: Check your CPU firmware version using DirectSOFT: PLC Menu > Diagnostics > System Information. The latest firmware and update tool are available from:
<http://support.automationdirect.com/firmware/index.html>

Analog Input/Output Combo Module Pointer Setup (ANLGCMB) (IB-462)

230

240

250-1

260

262

DS5	Used
HPP	N/A

The Analog Input/Output Combo Module Pointer Setup instruction generates the logic to configure the pointer method for an analog input/output combination module on the first PLC scan following a Program to Run transition.

The ANLGCMB IBox determines the data format and Pointer addresses based on the CPU type, the Base# and the module Slot#.

The Input Data Address is the starting location in user V-memory where the analog input data values will be stored, one location for each input channel enabled.

The Output Data Address is the starting location in user V-memory where the analog output data values will be stored by ladder code or external device, one location for each output channel enabled.

Since the IBox logic only executes on the first scan, the instruction cannot have any input logic.

☒☐☐

Analog Input/Output Combo Module Pointer Setup

ANLGCMB IB-462

Base # (K0-Local)	K0
Slot #	K0
Number of Input Channels	K1
Input Data Format (0-BCD 1-BIN)	K0
Input Data Address	V400
Number of Output Channels	K1
Output Data Format (0-BCD 1-BIN)	K0
Output Data Address	V400

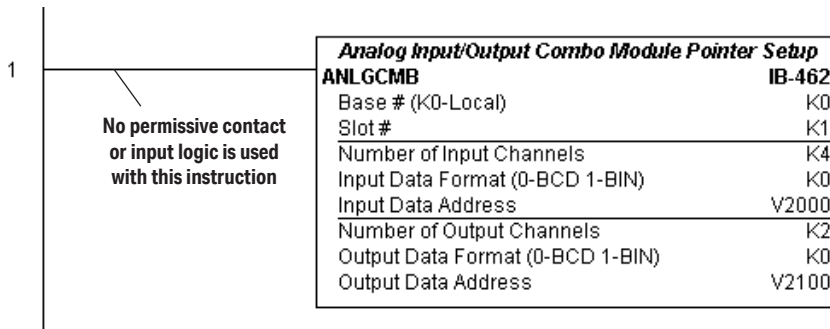
ANLGCMB Parameters

- Base # (K0-Local): specifies which base the module is in.
- Slot #: specifies which slot is occupied by the analog module.
- Number of Input Channels: specifies the number of analog input channels to scan.
- Input Data Format (0-BCD 1-BIN): specifies the analog input data format (BCD or Binary)—the binary format may be used for displaying data on some OI panels.
- Input Data Address: specifies the starting V-memory location that will be used to store the analog input data.
- Number of Output Channels: specifies the number of analog output channels that will be used.
- Output Data Format (0-BCD 1-BIN): specifies the format of the analog output data (BCD or Binary).
- Output Data Address: specifies the starting V-memory location that will be used to source the analog output data.

Parameter		DL205 Range
Base # (K0-Local)	K	K0-3
Slot #	K	K0-7
Number of Input Channels	K	K1-8
Input Data Format (0-BCD 1-BIN)	K	BCD: K0; Binary: K1
Input Data Address	V	See DL205 V-memory map—Data Words
Number of Output Channels	K	K1-8
Output Data Format (0-BCD 1-BIN)	K	BCD: K0; Binary: K1
Output Data Address	V	See DL205 V-memory map—Data Words

ANLGCMC Example

In the following example, the ANLGCMC instruction is used to set up the pointer method for an analog I/O combination module that is installed in option slot 2. Four input channels are enabled and the analog data will be written to V2000–V2003 in BCD format. Two output channels are enabled and the analog values will be read from V2100–V2101 in BCD format.



NOTE: An Analog I/O IBox instruction is used without a permissive contact. The top line will be identified in **BOLD** italics, and the instruction name and ID will be in **BOLD** characters.

Analog Input Module Pointer Setup (ANLGIN) (IB-460)

 230

 240

 250-1

 260

 262

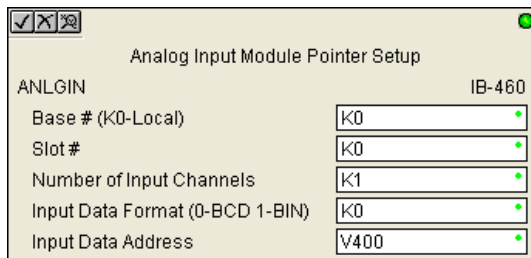
DS5	Used
HPP	N/A

Analog Input Module Pointer Setup generates the logic to configure the pointer method for one analog input module on the first PLC scan following a Program to Run transition.

This IBox determines the data format and Pointer addresses based on the CPU type, the Base#, and the Slot#.

The Input Data Address is the starting location in user V-memory where the analog input data values will be stored, one location for each input channel enabled.

Since this logic only executes on the first scan, this IBox cannot have any input logic.



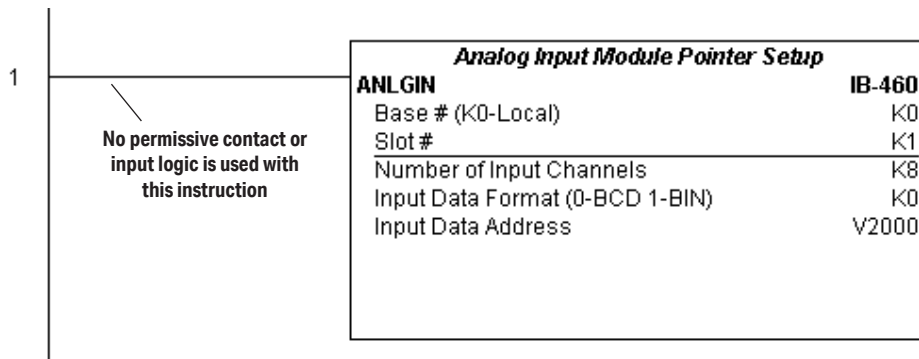
ANLGIN Parameters

- Base # (K0-Local): specifies which base the analog module is in.
- Slot #: specifies which PLC slot is occupied by the analog module.
- Number of Input Channels: specifies the number of input channels to scan.
- Input Data Format (0-BCD 1-BIN): specifies the analog input data format (BCD or Binary)—the binary format may be used for displaying data on some OI panels.
- Input Data Address: specifies the starting V-memory location that will be used to store the analog input data.

Parameter		DL205 Range
Base # (K0-Local)	K	K0-3
Slot #	K	K0-7
Number of Input Channels	K	K1-8
Input Data Format (0-BCD 1-BIN)	K	BCD: K0; Binary: K1
Input Data Address	V	See DL205 V-memory map—Data Words

ANLGIN Example

In the following example, the ANLGIN instruction is used to set up the pointer method for an analog input module that is installed in option slot 1. Eight input channels are enabled and the analog data will be written to V2000–V2007 in BCD format.



NOTE: An Analog I/O IBox instruction is used without a permissive contact. The top line will be identified in **BOLD italics**, and the instruction name and ID will be in **BOLD** characters.

Analog Output Module Pointer Setup (ANLGOUT) (IB-461)

 230

 240

 250-1

 260

 262

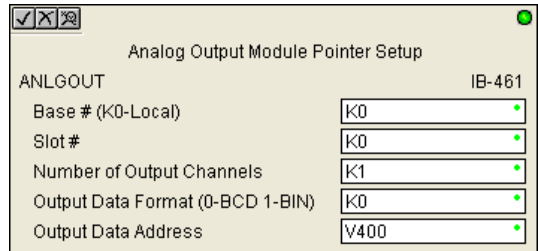
DS5	Used
HPP	N/A

Analog Output Module Pointer Setup generates the logic to configure the pointer method for one analog output module on the first PLC scan following a Program to Run transition.

This IBox determines the data format and Pointer addresses based on the CPU type, the Base#, and the Slot#.

The Output Data Address is the starting location in user V-memory where the analog output data values will be placed by ladder code or external device, one location for each output channel enabled.

Since this logic only executes on the first scan, this IBox cannot have any input logic.



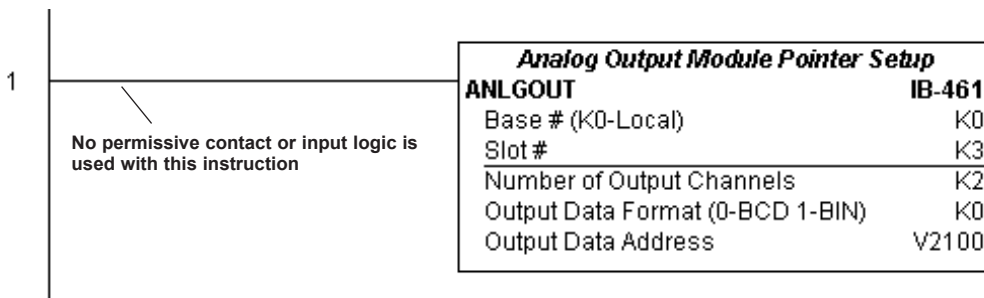
ANLGOUT Parameters

- Base # (K0-Local): specifies which base the analog module is in
- Slot #: specifies which PLC slot is occupied by the analog module
- Number of Output Channels: specifies the number of analog output channels that will be used
- Output Data Format (0-BCD 1-BIN): specifies the format of the analog output data (BCD or Binary)
- Output Data Address: specifies the starting V-memory location that will be used to source the analog output data

Parameter		DL205 Range
Base # (K0-Local)	K	K0-3
Slot #	K	K0-7
Number of Output Channels	K	K1-8
Output Data Format (0-BCD 1-BIN)	K	BCD: K0; Binary: K1
Output Data Address	V	See DL205 V-memory map-Data Words

ANLGOUT Example

In the following example, the ANLGOUT instruction is used to set up the pointer method for an analog output module that is installed in option slot 3. Two output channels are enabled and the analog data will be read from V2100–V2101 in BCD format.



NOTE: An Analog I/O IBox instruction is used without a permissive contact. The top line will be identified in ***BOLD*** italics, and the instruction name and ID will be in ***BOLD*** characters.

Analog Scale 12-Bit BCD to BCD (ANSCL) (IB-423)

☒ 230

☒ 240

☒ 250-1

☒ 260

☒ 262

DS5	Used
HPP	N/A

Analog Scale 12-Bit BCD to BCD scales a 12-bit BCD analog value (0 to 4095 BCD) into BCD engineering units. You specify the engineering unit high value (when raw is 4095), and the engineering low value (when raw is 0), and the output V-memory address where you want to place the scaled engineering unit value. The engineering units are generated as BCD and can be the full range of 0 to 9999 (see ANSCLB—Analog Scale 12-Bit Binary to Binary if your raw units are in Binary format).

Note that this IBox only works with unipolar unsigned raw values. It does NOT work with bipolar or sign plus magnitude raw values.

Analog Scale 12 Bit BCD to BCD	
ANSCL	IB-423
Raw (0-4095 BCD)	V2000
High Engineering	1000
Low Engineering	K0
Engineering (BCD)	V3000

ANSCL Parameters

- Raw (0 to 4095 BCD): specifies the V-memory location of the unipolar unsigned raw 0 to 4095 unscaled value
- High Engineering: specifies the high engineering value when the raw input is 4095
- Low Engineering: specifies the low engineering value when the raw input is 0
- Engineering (BCD): specifies the V-memory location where the scaled engineering BCD value will be placed

Parameter	DL205 Range
Raw (0-4095 BCD)	V,P See DL205 V-memory map—Data Words
High Engineering	K K0-9999
Low Engineering	K K0-9999
Engineering (BCD)	V,P See DL205 V-memory map—Data Words

ANSCL Example

In the following example, the ANSCL instruction is used to scale a raw value (0 to 4095 BCD) that is in V2000. The engineering scaling range is set 0 to 100 (low engineering value—high engineering value). The scaled value will be placed in V2100 in BCD format.



Analog Scale 12-Bit Binary to Binary (ANSCLB) (IB-403)

Analog Scale 12-Bit Binary to Binary scales a 12-bit binary analog value (0 to 4095 decimal) into binary (decimal) engineering units. You specify the engineering unit high value (when raw is 4095), and the engineering low value (when raw is 0), and the output V-memory address where you want to place the scaled engineering unit value. The engineering units are generated as binary and can be the full range of 0 to 65535 (see ANSCL-Analog Scale 12-Bit BCD to BCD if your raw units are in BCD format).

Note that this IBox only works with unipolar unsigned raw values. It does NOT work with bipolar, sign plus magnitude, or signed 2's complement raw values.

ANSCLB Parameters

- Raw (12-bit binary): specifies the V-memory location of the unipolar unsigned raw decimal unscaled value (12-bit binary = 0 to 4095 decimal)
- High Engineering: specifies the high engineering value when the raw input is 4095 decimal
- Low Engineering: specifies the low engineering value when the raw input is 0 decimal
- Engineering (binary): specifies the V-memory location where the scaled engineering decimal value will be placed

Parameter		DL205 Range
Raw (12-bit binary)	V,P	See DL205 V-memory map-Data Words
High Engineering	K	K0-65535
Low Engineering	K	K0-65535
Engineering (binary)	V,P	See DL205 V-memory map-Data Words

ANSCLB Example

In the following example, the ANSCLB instruction is used to scale a raw value (0 to 4095 binary) that is in V2000. The engineering scaling range is set 0 to 1000 (low engineering value-high engineering value). The scaled value will be placed in V2100 in binary format.



Filter Over Time–BCD (FILTER) (IB-422)

Filter Over Time BCD will perform a first-order filter on the Raw Data on a defined time interval. The equation is:

$$\text{New} = \text{Old} + [(\text{Raw} - \text{Old}) / \text{FDC}]$$

where,

- New: New Filtered Value
- Old: Old Filtered Value
- FDC: Filter Divisor Constant
- Raw: Raw Data

The Filter Divisor Constant is an integer in the range K1 to K100, such that if it equaled K1 then no filtering would be done.

The rate at which the calculation is performed is specified by time in hundredths of a second (0.01 seconds) as the Filter Freq Time parameter. Note that this Timer instruction is embedded in the IBox and must NOT be used anywhere else in your program. Power flow controls whether the calculation is enabled. If it is disabled, the Filter Value is not updated. On the first scan from Program to Run mode, the Filter Value is initialized to 0 to give the calculation a consistent starting point.

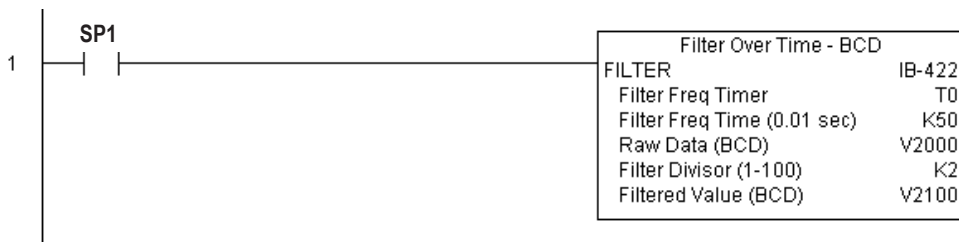
FILTER Parameters

- Filter Frequency Timer: specifies the Timer (T) number which is used by the Filter instruction.
- Filter Frequency Time (0.01sec): specifies the rate at which the calculation is performed.
- Raw Data (BCD): specifies the V-memory location of the raw unfiltered BCD value.
- Filter Divisor (1 to 100): this constant is used to control the filtering effect. A larger value will increase the smoothing effect of the filter. A value of 1 results with no filtering.
- Filtered Value (BCD): specifies the V-memory location where the filtered BCD value will be placed.

Parameter		DL205 Range
Filter Frequency Timer	T	T0-377
Filter Frequency Time (0.01 sec)	K	K0-9999
Raw Data (BCD)	V	See DL205 V-memory map–Data Words
Filter Divisor (1-100)	K	K1-100
Filtered Value (BCD)	V	See DL205 V-memory map–Data Words

FILTER Example

In the following example, the Filter instruction is used to filter a BCD value that is in V2000. Timer(T0) is set to 0.5 sec, the rate at which the filter calculation will be performed. The filter constant is set to 2. A larger value will increase the smoothing effect of the filter. A value of 1 results with no filtering. The filtered value will be placed in V2100.



Filter Over Time–Binary (FILTERB) (IB-402)

230

240

250-1

260

262

DS5	Used
HPP	N/A

Filter Over Time in Binary (decimal) will perform a first-order filter on the Raw Data on a defined time interval. The equation is:

New = Old + [(Raw–Old) / FDC] where

- New: New Filtered Value
- Old: Old Filtered Value
- FDC: Filter Divisor Constant
- Raw: Raw Data

The Filter Divisor Constant is an integer in the range K1 to K100, such that if it equaled K1 then no filtering would be done.

The rate at which the calculation is performed is specified by time in hundredths of a second (0.01 seconds) as the Filter Freq Time parameter. Note that this Timer instruction is embedded in the IBox and must NOT be used anywhere else in your program. Power flow controls whether the calculation is enabled. If it is disabled, the Filter Value is not updated. On the first scan from Program to Run mode, the Filter Value is initialized to 0 to give the calculation a consistent starting point.

FILTERB Parameters

- Filter Frequency Timer: specifies the Timer (T) number that is used by the Filter instruction.
- Filter Frequency Time (0.01sec): specifies the rate at which the calculation is performed.
- Raw Data (Binary): specifies the V-memory location of the raw unfiltered binary (decimal) value.
- Filter Divisor (1 to 100): this constant is used to control the filtering effect. A larger value will increase the smoothing effect of the filter. A value of 1 results with no filtering.
- Filtered Value (Binary): specifies the V-memory location where the filtered binary (decimal) value will be placed.

Parameter		DL205 Range
Filter Frequency Timer	T	T0-377
Filter Frequency Time (0.01 sec)	K	K0-9999
Raw Data (Binary)	V	See DL205 V-memory map–Data Words
Filter Divisor (1-100)	K	K1-100
Filtered Value (Binary)	V	See DL205 V-memory map–Data Words

FILTERB Example

In the following example, the FILTERB instruction is used to filter a binary value that is in V2000. Timer(T1) is set to 0.5 sec, the rate at which the filter calculation will be performed. The filter constant is set to 3. A larger value will increase the smoothing effect of the filter. A value of 1 results with no filtering. The filtered value will be placed in V2100



Hi/Low Alarm–BCD (HILOAL) (IB-421)

 230

 240

 250-1

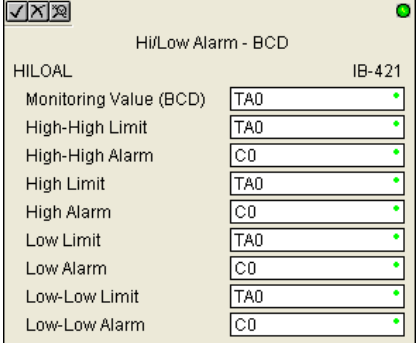
 260

 262

DS5	Used
HPP	N/A

Hi/Low Alarm–BCD monitors a BCD value V-memory location and sets four possible alarm states, High-High, High, Low, and Low-Low whenever the IBox has power flow. You enter the alarm thresholds as constant (K) BCD values (K0-K9999) and/or BCD value V-memory locations.

You must ensure that threshold limits are valid, that is $HH \geq H > L \geq LL$. Note that when the High-High or Low-Low alarm condition is true, that the High and Low alarms will also be set, respectively. This means you may use the same threshold limit and same alarm bit for the High-High and the High alarms in case you only need one “High” alarm. Also note that the boundary conditions are inclusive. That is, if the Low boundary is K50, and the Low-Low boundary is K10, and if the Monitoring Value equals 10, then the Low Alarm AND the Low-Low alarm will both be ON. If there is no power flow to the IBox, then all alarm bits will be turned off regardless of the value of the Monitoring Value parameter.



HiLow Alarm - BCD		IB-421
HILOAL		
Monitoring Value (BCD)	TA0	
High-High Limit	TA0	
High-High Alarm	C0	
High Limit	TA0	
High Alarm	C0	
Low Limit	TA0	
Low Alarm	C0	
Low-Low Limit	TA0	
Low-Low Alarm	C0	

HILOAL Parameters

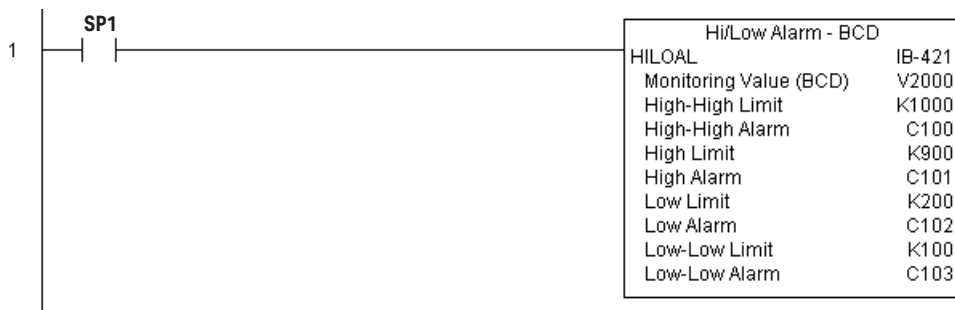
- Monitoring Value (BCD): specifies the V-memory location of the BCD value to be monitored
- High-High Limit: V-memory location or constant specifies the high-high alarm limit
- High-High Alarm: On when the high-high limit is reached
- High Limit: V-memory location or constant specifies the high alarm limit
- High Alarm: On when the high limit is reached
- Low Limit: V-memory location or constant specifies the low alarm limit
- Low Alarm: On when the low limit is reached
- Low-Low Limit: V-memory location or constant specifies the low-low alarm limit
- Low-Low Alarm: On when the low-low limit is reached

Parameter		DL205 Range
Monitoring Value (BCD)	V	See DL205 V-memory map–Data Words
High-High Limit	V, K	K0-9999; or see DL205 V-memory map–Data Words
High-High Alarm	X, Y, C, GX, GY, B	See DL205 V-memory map
High Limit	V, K	K0-9999; or see DL205 V-memory map–Data Words
High Alarm	X, Y, C, GX, GY, B	See DL205 V-memory map
Low Limit	V, K	K0-9999; or see DL205 V-memory map–Data Words
Low Alarm	X, Y, C, GX, GY, B	See DL205 V-memory map
Low-Low Limit	V, K	K0-9999; or see DL205 V-memory map–Data Words
Low-Low Alarm	X, Y, C, GX, GY, B	See DL205 V-memory map

HILOAL Example

In the following example, the HILOAL instruction is used to monitor a BCD value that is in V2000. If the value in V2000 meets/exceeds the High limit of K900, C101 will turn on. If the value continues to increase to meet/exceed the High-High limit, C100 will turn on. Both bits would be on in this case. The High and High-High limits and alarms can be set to the same value if one "High" limit or alarm is desired to be used.

If the value in V2000 meets or falls below the Low limit of K200, C102 will turn on. If the value continues to decrease to meet or fall below the Low-Low limit of K100, C103 will turn on. Both bits would be on in this case. The Low and Low-Low limits and alarms can be set to the same value if one "Low" limit or alarm is desired to be used.



230 Hi/Low Alarm–Binary (HILOALB) (IB-401)

 240

 250-1

 260

 262

DS5	Used
HPP	N/A

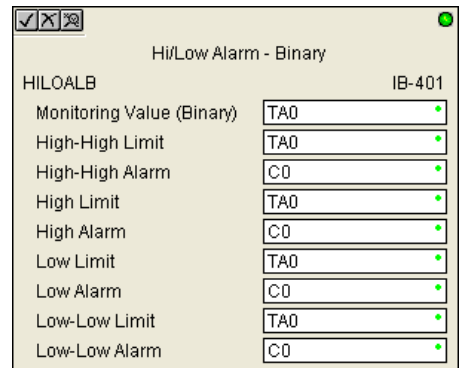
Hi/Low Alarm–Binary monitors a binary (decimal) V-memory location and sets four possible alarm states, High-High, High, Low, and Low-Low whenever the IBox has power flow.

You enter the alarm thresholds as constant (K) decimal values (K0-K65535) and/or binary (decimal) V-memory locations.

You must ensure that threshold limits are valid, that is $HH \geq H > L \geq LL$. Note that when the High-High or Low-Low alarm condition is true, that the High and Low alarms will also be set, respectively. This means you may use the same threshold limit and same alarm bit for the High-High and the High alarms in case you only need one “High” alarm. Also note that the boundary conditions are inclusive. That is, if the Low boundary is K50, and the Low-Low boundary is K10, and if the Monitoring Value equals 10, then the Low Alarm AND the Low-Low alarm will both be ON. If there is no power flow to the IBox, then all alarm bits will be turned off regardless of the value of the Monitoring Value parameter.

HILOALB Parameters

- Monitoring Value (Binary): specifies the V-memory location of the Binary value to be monitored
- High-High Limit: V-memory location or constant specifies the High-High alarm limit
- High-High Alarm: On when the High-High limit is reached
- High Limit: V-memory location or constant specifies the High alarm limit
- High Alarm: On when the High limit is reached
- Low Limit: V-memory location or constant specifies the Low alarm limit
- Low Alarm: On when the Low limit is reached
- Low-Low Limit: V-memory location or constant specifies the Low-Low alarm limit
- Low-Low Alarm: On when the Low-Low limit is reached



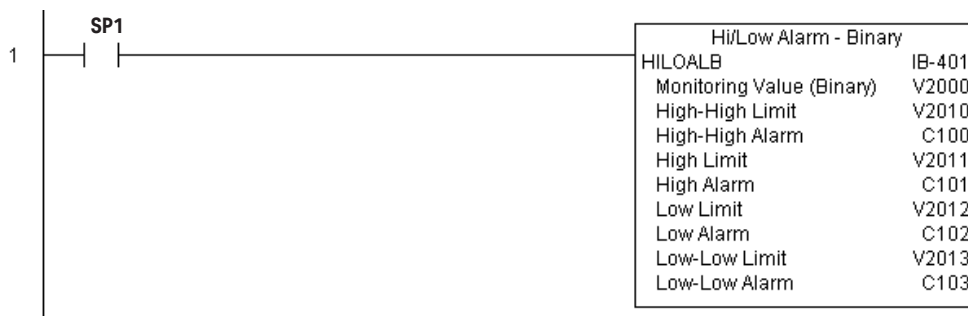
Hi/Low Alarm - Binary	
HILOALB	IB-401
Monitoring Value (Binary)	TA0
High-High Limit	TA0
High-High Alarm	C0
High Limit	TA0
High Alarm	C0
Low Limit	TA0
Low Alarm	C0
Low-Low Limit	TA0
Low-Low Alarm	C0

Parameter		DL205 Range
Monitoring Value (Binary)	V	See DL205 V-memory map–Data Words
High-High Limit	V, K	K0-65535; or see DL205 V-memory map–Data Words
High-High Alarm	X, Y, C, GX,GY, B	See DL205 V-memory map
High Limit	V, K	K0-65535; or see DL205 V-memory map–Data Words
High Alarm	X, Y, C, GX,GY, B	See DL205 V-memory map
Low Limit	V, K	K0-65535; or see DL205 V-memory map–Data Words
Low Alarm	X, Y, C, GX,GY,B	See DL205 V-memory map
Low-Low Limit	V, K	K0-65535; or see DL205 V-memory map–Data Words
Low-Low Alarm	X, Y, C, GX,GY, B	See DL205 V-memory map

HILOALB Example

In the following example, the HILOALB instruction is used to monitor a binary value that is in V2000. If the value in V2000 meets/exceeds the High limit of the binary value in V2011, C101 will turn on. If the value continues to increase to meet/exceed the High-High limit value in V2010, C100 will turn on. Both bits would be on in this case. The High and High-High limits and alarms can be set to the same V-memory location/value if one "High" limit or alarm is desired to be used.

If the value in V2000 meets or falls below the low limit of the binary value in V2012, C102 will turn on. If the value continues to decrease to meet or fall below the Low-Low limit in V2013, C103 will turn on. Both bits would be on in this case. The Low and Low-Low limits and alarms can be set to the same V-memory location/value if one "Low" limit or alarm is desired to be used.



Off Delay Timer (OFFDTMR) (IB-302)

230

240

250-1

260

262

Off Delay Timer will delay the “turning off” of the Output parameter by the specified Off Delay Time (up to 99.99 seconds) based on the power flow into the IBox. Once the IBox receives power, the Output bit will turn on immediately. When the power flow to the IBox turns off, the Output bit WILL REMAIN ON for the specified amount of time (in hundredths of a second). Once the Off Delay Time has expired, the output will turn Off. If the power flow to the IBox comes back on BEFORE the Off Delay Time, then the timer is RESET and the Output will remain On—so you must continuously have NO power flow to the IBox for AT LEAST the specified Off Delay Time before the Output will turn Off.

DS5	Used
HPP	N/A

This IBox utilizes a Timer resource (TMRF), which cannot be used anywhere else in your program.

OFFDTMR Parameters

- Timer Number: specifies the Timer (TMRF) number which is used by the OFFDTMR instruction
- Off Delay Time (0.01sec): specifies how long the Output will remain on once power flow to the Ibox is removed (up to 99.99 seconds).
- Output: specifies the output that will be delayed “turning off” by the Off Delay Time.

Off Delay Timer

IB-302

OFFDTMR

Timer Number

T0

Off Delay Time (0.01 sec)

TA0

Output

C0

Parameter		DL205 Range
Timer Number	T	T0-377
Off Delay Time	K,V	K0-9999; See DL205 V-memory map–Data Words
Output	X, Y, C, GX,GY, B	See DL205 V-memory map

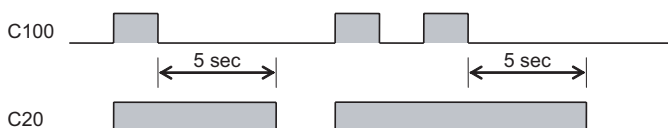
OFFDTMR Example

In the following example, the OFFDTMR instruction is used to delay the “turning off” of output C20. Timer 2 (T2) is set to 5 seconds, the “off-delay” period.

When C100 turns on, C20 turns on and will remain on while C100 is on. When C100 turns off, C20 will remain on for the specified Off Delay Time (5 secs), and then turn off.



Example timing diagram



On Delay Timer (ONDTMR) (IB-301)

 230

 240

 250-1

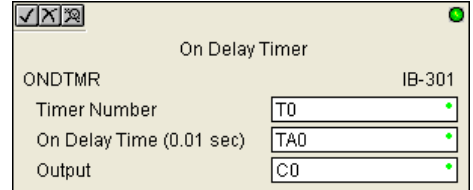
 260

 262

DS5	Used
HPP	N/A

On Delay Timer will delay the “turning on” of the Output parameter by the specified amount of time (up to 99.99 seconds) based on the power flow into the IBox. Once the IBox loses power, the Output is turned off immediately. If the power flow turns off BEFORE the On Delay Time, then the timer is RESET and the Output is never turned on, so you must have continuous power flow to the IBox for at least the specified On Delay Time before the Output turns On.

This IBox utilizes a Timer resource (TMRF), which cannot be used anywhere else in your program.



ONDTMR Parameters

- Timer Number: specifies the Timer (TMRF) number which is used by the ONDTMR instruction
- On Delay Time (0.01sec): specifies how long the Output will remain on once power flow to the Ibox is removed (up to 99.99 seconds).
- Output: specifies the output that will be delayed “turning on” by the On Delay Time.

Parameter		DL205 Range
Timer Number	T	T0-377
On Delay Time	K,V	K0-9999; See DL205 V-memory map-Data Words
Output	X, Y, C, GX,GY, B	See DL205 V-memory map

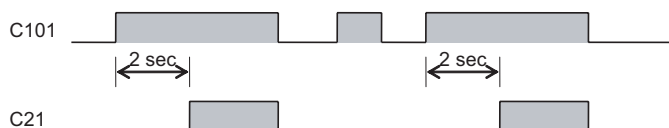
ONDTMR Example

In the following example, the ONDTMR instruction is used to delay the “turning on” of output C21. Timer 1 (T1) is set to 2 seconds, the “on-delay” period.

When C101 turns on, C21 is delayed turning on by 2 seconds. When C101 turns off, C21 turns off immediately.



Example timing diagram



One Shot (ONESHOT) (IB-303)

230

240

250-1

260

262

One Shot will turn on the given bit output parameter for one scan on an OFF to ON transition of the power flow into the IBox. This IBox is simply a different name for the PD Coil (Positive Differential).

ONESHOT Parameters

- Discrete Output: specifies the output that will be on for one scan

DS5	Used
HPP	N/A

☒☐☐

One Shot

ONESHOT IB-303

Discrete Output C0

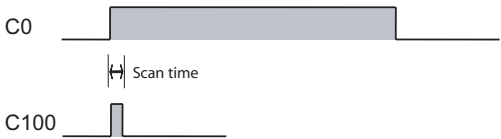
Parameter	DL205 Range
Discrete Output X, Y, C	See DL205 V-memory map

ONESHOT Example

In the following example, the ONESHOT instruction is used to turn C100 on for one PLC scan after C0 goes from an off to on transition. The input logic must produce an off to on transition to execute the One Shot instruction.



Example Timing Diagram



Push On/Push Off Circuit (PONOFF) (IB-300)

Push On/Push Off Circuit toggles an output state whenever its input power flow transitions from off to on. Requires an extra bit parameter for scan-to-scan state information. This extra bit must NOT be used anywhere else in the program. This is also known as a “flip-flop circuit.”

☒ 230

☒ 240

☒ 250-1

☒ 260

☒ 262

DS5	Used
HPP	N/A

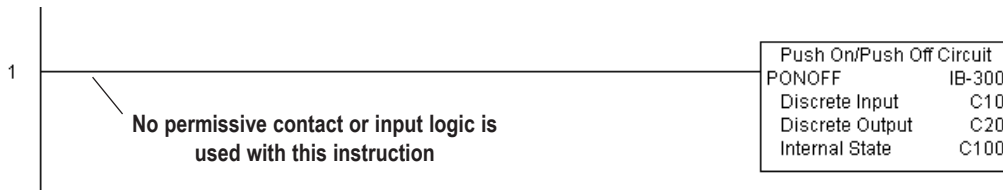
PONOFF Parameters

- Discrete Input: specifies the input that will toggle the specified output
- Discrete Output: specifies the output that will be “turned on/off” or toggled
- Internal State: specifies a work bit that is used by the instruction

Parameter	DL205 Range
Discrete Input X,Y,C,S,T,CT,GX,GY,SP,B,PB	See DL205 V-memory map
Discrete Output X,Y,C,GX,GY,B	See DL205 V-memory map
Internal State X, Y, C	See DL205 V-memory map

PONOFF Example

In the following example, the PONOFF instruction is used to control the on and off states of the output C20 with a single input C10. When C10 is pressed once, C20 turns on. When C10 is pressed again, C20 turns off. C100 is an internal bit used by the instruction.



NOTE: Neither a permissive nor input logic is required with this instruction.

Move Single Word (MOVEW) (IB-200)

Move Single Word moves (copies) a word to a memory location directly or indirectly via a pointer, either as a HEX constant, from a memory location, or indirectly through a pointer.

230

240

250-1

260

262

DS5	Used
HPP	N/A

MOVEW Parameters

- From WORD: specifies the word
- To WORD: specifies the location which where the “From WORD” will be moved

✓✕⌂

Move Single Word

IB-200

MOVEW

From WORD

TA0

To WORD

TA0

Parameter	DL205 Range
From WORD	V,P,K K0-FFFF; See DL205 V-memory map-Data Words
To WORD	V,P See DL205 V-memory map-Data Words

MOVEW Example

In the following example, the MOVEW instruction is used to move 16 bits of data from V2000 to V3000 when C100 turns on.



Move Double Word (MOVED) (IB-201)

Move Double Word moves (copies) a double word to two consecutive memory locations directly or indirectly via a pointer, either as a double HEX constant, from a double memory location, or indirectly through a pointer to a double memory location.

☒ 230

☒ 240

☒ 250-1

☒ 260

☒ 262

DS5	Used
HPP	N/A

MOVED Parameters

- From DWORD: specifies the double word that will be moved to another location
- To DWORD: specifies the location to which where the "From DWORD" will be moved

☒ ☒ ☒

Move Double Word

MOVED IB-201

From DWORD TA0

To DWORD TA0

Parameter		DL205 Range
From WORD	V,P,K	K0-FFFFFFF; See DL205 V-memory map-Data Words
To WORD	V,P	See DL205 V-memory map-Data Words

MOVED Example

In the following example, the MOVED instruction is used to move 32 bits of data from V2000 and V2001 to V3000 and V3001 when C100 turns on.



BCD to Real with Implied Decimal Point (BCDTOR) (IB-560)

230

240

250-1

260

262

DS5	Used
HPP	N/A

BCD to Real with Implied Decimal Point converts the given 4-digit WORD BCD value to a Real number, with the implied number of decimal points (K0-K4).

For example, BCDTOR K1234 with an implied number of decimal points equal to K1, would yield R123.4

BCDTOR Parameters

- Value (WORD BCD): specifies the word or constant that will be converted to a Real number
- Number of Decimal Points: specifies the number of implied decimal points in the Result DWORD
- Result (DWORD REAL): specifies the location where the Real number will be placed

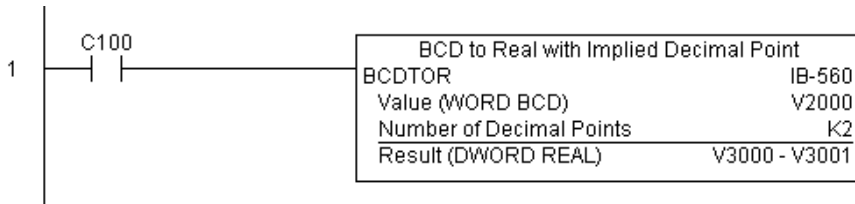
BCD to Real with Implied Decimal Point	
BCDTOR	IB-560
Value (WORD BCD)	TA0
Number of Decimal Points	K0
Result (DWORD REAL)	V400

Parameter	DL205 Range
Value (WORD BCD)	V,P,K K0-9999; See DL205 V-memory map-Data Words
Number of Decimal Points	K K0-4
Result (DWORD REAL)	V See DL205 V-memory map-Data Words

BCDTOR Example

In the following example, the BCDTOR instruction is used to convert the 16-bit data in V2000 from a 4-digit BCD data format to a 32-bit REAL (floating point) data format and store into V3000 and V3001 when C100 turns on.

K2 in the Number of Decimal Points implies the data will have two digits to the right of the decimal point.



Double BCD to Real with Implied Decimal Point (BCDTORD) (IB-562)

230

Double BCD to Real with Implied Decimal Point converts the given 8-digit DWORD BCD value to a Real number, given an implied number of decimal points (K0-K8).

240

For example, BCDTORD K12345678 with an implied number of decimal points equal to K5, would yield R123.45678

250-1

260

BCDTORD Parameters

262

- Value (DWORD BCD): specifies the Dword or constant that will be converted to a Real number
- Number of Decimal Points: specifies the number of implied decimal points in the Result DWORD
- Result (DWORD REAL): specifies the location where the Real number will be placed

DS5	Used
HPP	N/A

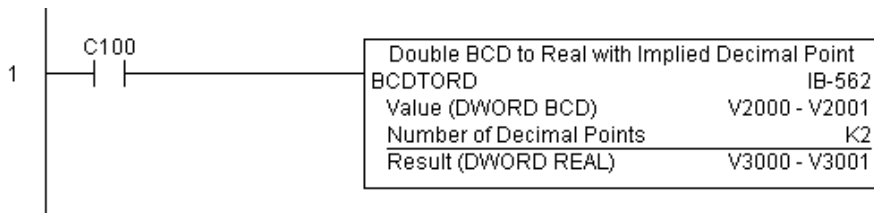
Double BCD to Real with Implied Decimal Point	
BCDTORD	IB-562
Value (DWORD BCD)	TA0
Number of Decimal Points	K0
Result (DWORD REAL)	V400

Parameter	DL205 Range
Value (WORD BCD)	V,P,K K0-99999999; See DL205 V-memory map-Data Words
Number of Decimal Points	K K0-8
Result (DWORD REAL)	V See DL205 V-memory map-Data Words

BCDTORD Example

In the following example, the BCDTORD instruction is used to convert the 32-bit data in V2000 from an 8-digit BCD data format to a 32-bit REAL (floating point) data format and store into V3000 and V3001 when C100 turns on.

K2 in the Number of Decimal Points implies the data will have two digits to the right of the decimal point.



Math-BCD (MATHBCD) (IB-521)

- 230
- 240
- 250-1
- 260
- 262

DS5	Used
HPP	N/A

Math-BCD Format lets you enter complex mathematical expressions like you would in Visual Basic, Excel, or C++ to do complex calculations, nesting parentheses up to 4 levels deep. In addition to +-* /, you can do Modulo (% aka Remainder), Bit-wise And (&) Or (|) Xor (^), and some BCD functions-Convert to BCD (BCD), Convert to Binary (BIN), BCD Complement (BCDCPL), Convert from Gray Code (GRAY), Invert Bits (INV), and BCD/HEX to Seven Segment Display (SEG).

Example: ((V2000 + V2001) / (V2003-K100)) * GRAY(V3000 & K001F)

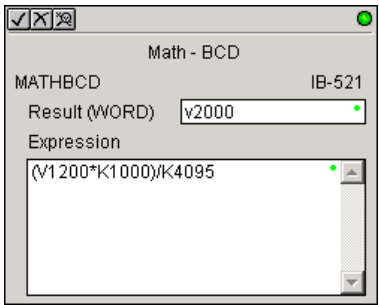
Every V-memory reference MUST be to a single-word BCD formatted value. Intermediate results can go up to 32-bit values, but as long as the final result fits in a 16-bit BCD word, the calculation is valid. Typical example of this is scaling using multiply then divide, (V2000 * K1000) / K4095. The multiply term most likely will exceed 9999 but fits within 32 bits. The divide operation will divide 4095 into the 32-bit accumulator, yielding a result that will always fit in 16 bits.

You can reference binary V-memory values by using the BCD conversion function on a V-memory location but NOT an expression. That is BCD(V2000) is okay and will convert V2000 from Binary to BCD, but BCD(V2000 + V3000) will add V2000 as BCD, to V3000 as BCD, then interpret the result as Binary and convert it to BCD-NOT GOOD.

Also, the final result is a 16-bit BCD number and so you could do BIN around the entire operation to store the result as Binary.

MATHBCD Parameters

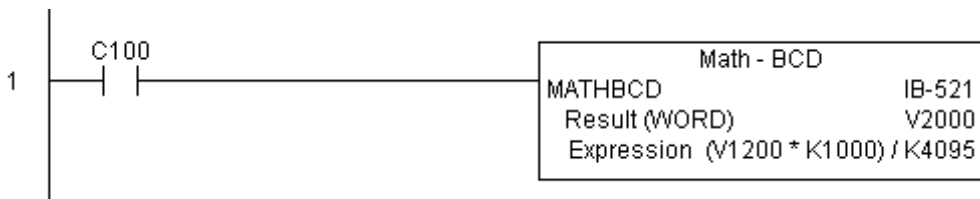
- Result (WORD): specifies the location where the BCD result of the mathematical expression will be placed (result must fit into 16-bit single V-memory location).
- Expression: specifies the mathematical expression to be executed and the result is stored in specified Result (WORD). Each V-memory location used in the expression must be in BCD format.



Parameter		DL205 Range
WORD Result	V	See DL205 V-memory map-Data Words
Expression		Text

MATHBCD Example

In the following example, the MATHBCD instruction is used to calculate the math expression which multiplies the BCD value in V1200 by 1000, then divides by 4095 and loads the resulting value in V2000 when C100 turns on.



Math-Binary (MATHBIN) (IB-501)

	230
	240
	250-1
	260
	262
DS5	Used
HPP	N/A

Math-Binary Format lets you enter complex mathematical expressions like you would in Visual Basic, Excel, or C++ to do complex calculations, nesting parentheses up to 4 levels deep. In addition to +-* /, you can do Modulo (% aka Remainder), Shift Right (>>) and Shift Left (<<), Bit-wise And (&) Or (|) Xor (^), and some binary functions—Convert to BCD (BCD), Convert to Binary (BIN), Decode Bits (DECO), Encode Bits (ENCO), Invert Bits (INV), HEX to Seven Segment Display (SEG), and Sum Bits (SUM).

Example: ((V2000 + V2001) / (V2003-K10)) * SUM(V3000 & K001F)

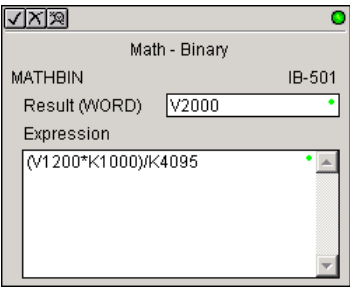
Every V-memory reference MUST be to a single-word binary formatted value. Intermediate results can go up to 32-bit values, but as long as the final result fits in a 16-bit binary word, the calculation is valid. Typical example of this is scaling using multiply then divide, (V2000 * K1000) / K4095. The multiply term most likely will exceed 65535 but fits within 32 bits. The divide operation will divide 4095 into the 32-bit accumulator, yielding a result that will always fit in 16 bits.

You can reference BCD V-memory values by using the BIN conversion function on a V-memory location but NOT an expression. That is, BIN(V2000) is okay and will convert V2000 from BCD to Binary, but BIN(V2000 + V3000) will add V2000 as Binary, to V3000 as Binary, then interpret the result as BCD and convert it to Binary—NOT GOOD.

Also, the final result is a 16-bit binary number and so you could do BCD around the entire operation to store the result as BCD.

MATHBIN Parameters

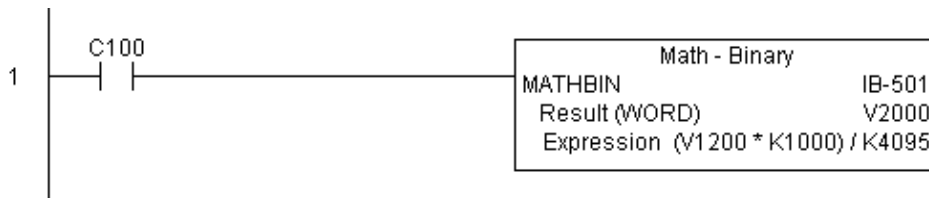
- Result (WORD): specifies the location where the binary result of the mathematical expression will be placed (result must fit into 16-bit single V-memory location).
- Expression: specifies the mathematical expression to be executed and the result is stored in specified Result (WORD). Each V-memory location used in the expression must be in binary format.



Parameter		DL205 Range
WORD Result	V	See DL205 V-memory map—Data Words
Expression		Text

MATHBIN Example

In the following example, the MATHBIN instruction is used to calculate the math expression which multiplies the Binary value in V1200 by 1000 then divides by 4095 and loads the resulting value in V2000 when C100 turns on.



Math-Real (MATHR) (IB-541)

- 230
- 240
- 250-1
- 260
- 262

DS5	Used
HPP	N/A

Math-Real Format lets you enter complex mathematical expressions like you would in Visual Basic, Excel, or C++ to do complex calculations, nesting parentheses up to 4 levels deep. In addition to +-* /, you can do Bit-wise And (&) Or (|) and Xor (^). The D2-260 and D2-262 also support several Real functions-Arc Cosine (ACOSR), Arc Sine (ASINR), Arc Tangent (ATANR), Cosine (COSR), Convert Radians to Degrees (DEGR), Invert Bits (INV), Convert Degrees to Radians (RADR), HEX to Seven Segment Display (SEG), Sine (SINR), Square Root (SQRTR), and Tangent (TANR).

Example: ((V2000 + V2002) / (V2004-R2.5)) * SINR(RADR(V3000 / R10.0))

Every V-memory reference MUST be able to fit into a double-word Real formatted value.

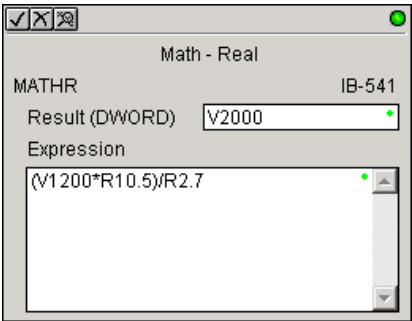
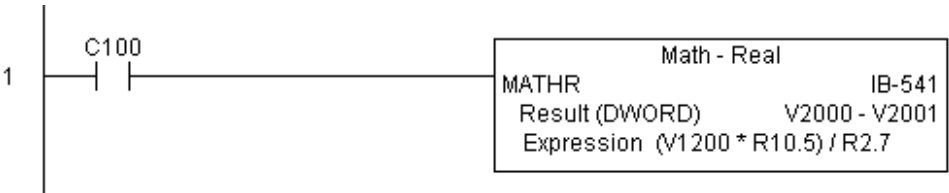
MATHR Parameters

- Result (DWORD): specifies the location where the Real result of the mathematical expression will be placed (result must fit into a double-word Real formatted location).
- Expression: specifies the mathematical expression to be executed and the result is stored in specified Result (DWORD) location. Each V-memory location used in the expression must be in Real format.

Parameter	DL205 Range
DWORD Result	V See DL205 V-memory map-Data Words
Expression	Text

MATHR Example

In the following example, the MATHR instruction is used to calculate the math expression which multiplies the REAL (floating point) value in V1200 by 10.5 then divides by 2.7 and loads the resulting 32-bit value in V2000 and V2001 when C100 turns on.



Real to BCD with Implied Decimal Point and Rounding (RTOBCD) (IB-561)

Real to BCD with Implied Decimal Point and Rounding converts the absolute value of the given Real number to a 4-digit BCD number, compensating for an implied number of decimal points (K0-K4) and performs rounding.

For example, RTOBCD R56.74 with an implied number of decimal points equal to K1, would yield 567 BCD. If the implied number of decimal points was 0, then the function would yield 57 BCD (note that it rounded up).

If the Real number is negative, the Result will equal its positive, absolute value.

230

240

250-1

260

262

DS5	Used
HPP	N/A

RTOBCD Parameters

- Value (DWORD Real): specifies the Real Dword location or number that will be converted and rounded to a BCD number with decimal points
- Number of Decimal Points: specifies the number of implied decimal points in the Result WORD
- Result (WORD BCD): specifies the location where the rounded/ implied decimal points BCD value will be placed

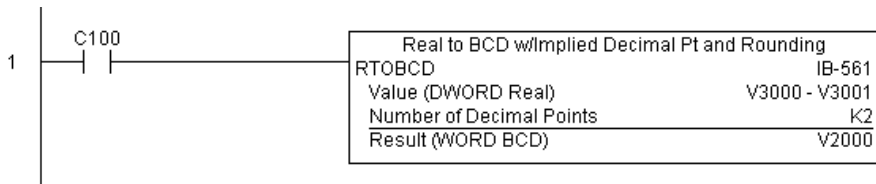
Real to BCD w/Implied Decimal Pt and Rounding	
RTOBCD IB-561	
Value (DWORD Real)	TA0
Number of Decimal Points	K0
Result (WORD BCD)	V400

Parameter		DL205 Range
Value (DWORD Real)	V,P,R	R; See DL205 V-memory map-Data Words
Number of Decimal Points	K	K0-4
Result (WORD REAL)	V	See DL205 V-memory map-Data Words

RTOBCD Example

In the following example, the RTOBCD instruction is used to convert the 32-bit REAL (floating point) data format in V3000 and V3001 to the 4-digit BCD data format and store in V2000 when C100 turns on.

K2 in the Number of Decimal Points implies the data will have two implied decimal points.



Real to Double BCD with Implied Decimal Point and Rounding
(RTOBCDD) (IB-563)

230

240

250-1

260

262

Real to Double BCD with Implied Decimal Point and Rounding converts the absolute value of the given Real number to an 8-digit DWORD BCD number, compensating for an implied number of decimal points (K0-K8) and performs rounding.

For example, RTOBCDD R38156.74 with an implied number of decimal points equal to K1, would yield 381567 BCD. If the implied number of decimal points was 0, then the function would yield 38157 BCD (note that it rounded up).

If the Real number is negative, the Result will equal its positive, absolute value.

DS5	Used
HPP	N/A

RTOBCDD Parameters

- Value (DWORD Real): specifies the Dword Real number that will be converted and rounded to a BCD number with decimal points
- Number of Decimal Points: specifies the number of implied decimal points in the Result DWORD
- Result (DWORD BCD): specifies the location where the rounded/ implied decimal points DWORD BCD value will be placed

☒ ☒ ☒

Real to Double BCD w/Implied Decimal Pt and Rounding
RTOBCDD IB-563

Value (DWORD Real)

TA0

Number of Decimal Points

K0

Result (DWORD BCD)

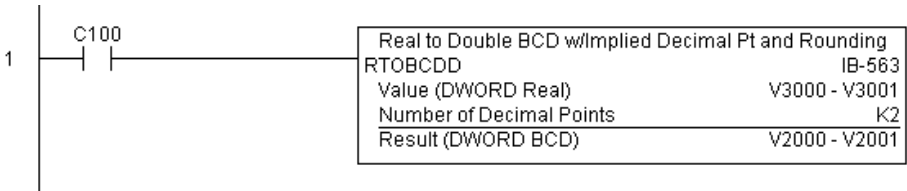
V400

Parameter		DL205 Range
Value (DWORD Real)	V,P,R	R; See DL205 V-memory map-Data Words
Number of Decimal Points	K	K0-8
Result (DWORD BCD)	V	See DL205 V-memory map-Data Words

RTOBCDD Example

In the following example, the RTOBCDD instruction is used to convert the 32-bit REAL (floating point) data format in V3000 and V3001 to the 8-digit BCD data format and store in V2000 and V2001 when C100 turns on.

K2 in the Number of Decimal Points implies the data will have two implied decimal points.



Square BCD (SQUARE) (IB-523)

Square BCD squares the given 4-digit WORD BCD number and writes it as an 8-digit DWORD BCD result.

230

240

250-1

260

262

SQUARE Parameters

- Value (WORD BCD): specifies the BCD Word or constant that will be squared
- Result (DWORD BCD): specifies the location where the squared DWORD BCD value will be placed

DS5	Used
HPP	N/A

☒
☒
☒
☒

Square BCD

SQUARE

IB-523

Value (WORD BCD)

TA0

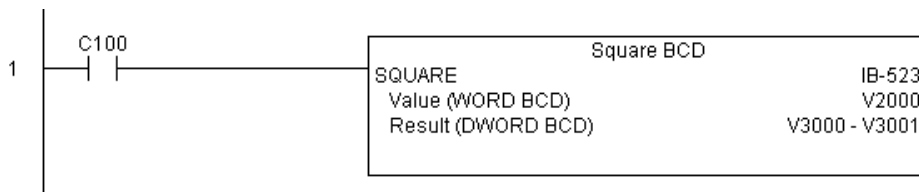
Result (DWORD BCD)

V400

SQUARE Example

Parameter	DL205 Range
Value (WORD BCD)	V,P,K K0-9999 ; See DL205 V-memory map-Data Words
Result (DWORD BCD)	V See DL205 V-memory map-Data Words

In the following example, the SQUARE instruction is used to square the 4-digit BCD value in V2000 and store the 8-digit double word BCD result in V3000 and V3001 when C100 turns on.



Square Binary (SQUAREB) (IB-503)

Square Binary squares the given 16-bit WORD Binary number and writes it as a 32-bit DWORD Binary result.

230

240

250-1

260

262

DS5	Used
HPP	N/A

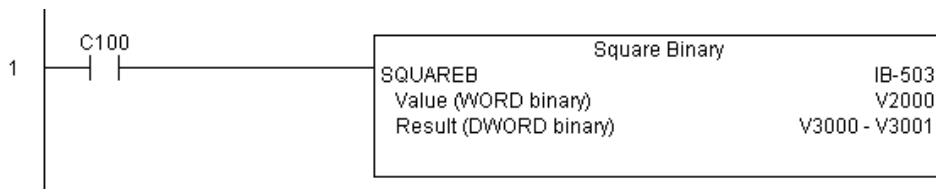
SQUAREB Parameters

- Value (WORD Binary): specifies the binary Word or constant that will be squared
- Result (DWORD Binary): specifies the location where the squared DWORD binary value will be placed

Parameter		DL205 Range
Value (WORD Binary)	V,P,K	K0-65535; See DL205 V-memory map-Data Words
Result (DWORD Binary)	V	See DL205 V-memory map-Data Words

SQUAREB Example

In the following example, the SQUAREB instruction is used to square the single-word Binary value in V2000 and store the 8-digit double-word Binary result in V3000 and V3001 when C100 turns on.



Square Real (SQUARER) (IB-543)

Square Real squares the given REAL DWORD number and writes it to a REAL DWORD result.

 230

 240

 250-1

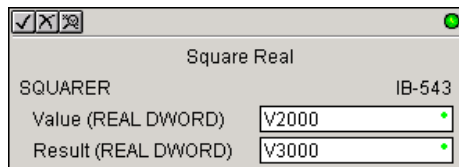
 260

 262

SQUARER Parameters

- Value (REAL DWORD): specifies the Real DWORD location or number that will be squared
- Result (REAL DWORD): specifies the location where the squared Real DWORD value will be placed

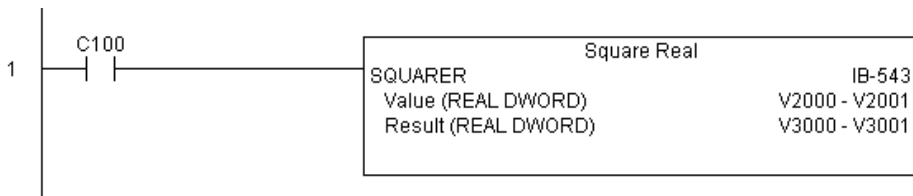
DS5	Used
HPP	N/A



Parameter		DL205 Range
Value (REAL DWORD)	V,P,R	R; See DL205 V-memory map-Data Words
Result (REAL DWORD)	V	See DL205 V-memory map-Data Words

SQUARER Example

In the following example, the SQUARER instruction is used to square the 32-bit floating point REAL value in V2000 and V2001 and store the REAL value result in V3000 and V3001 when C100 turns on.



Sum BCD Numbers (SUMBCD) (IB-522)

Sum BCD Numbers sums up a list of consecutive 4-digit WORD BCD numbers into an 8-digit DWORD BCD result.

230

240

250-1

260

262

You specify the group's starting and ending V-memory addresses (inclusive). When enabled, this instruction will add all the numbers in the group (so you may want to place a differential contact driving the enable).

SUMBCD could be used as the first part of calculating an average.

SUMBCD Parameters

DS5	Used
HPP	N/A

- Start Address: specifies the starting address of a block of V-memory location values to be added together (BCD)
- End Addr (inclusive): specifies the ending address of a block of V-memory location values to be added together (BCD)
- Result (DWORD BCD): specifies the location where the sum of the block of V-memory BCD values will be placed

☒☒☒

Sum BCD Numbers

IB-522

SUMBCD

Start AddressV2000

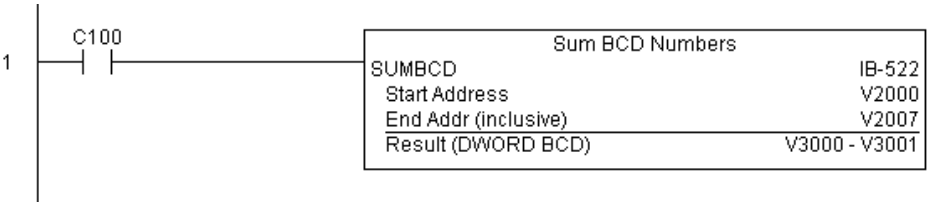
End Addr (inclusive)V2007

Result (DWORD BCD)V3000

Parameter		DL205 Range
Start Address	V	See DL205 V-memory map-Data Words
End Address (inclusive)	V	See DL205 V-memory map-Data Words
Result (DWORD BCD)	V	See DL205 V-memory map-Data Words

SUMBCD Example

In the following example, the SUMBCD instruction is used to total the sum of all BCD values in words V2000 thru V2007 and store the resulting 8-digit double word BCD value in V3000 and V3001 when C100 turns on.



Sum Binary Numbers (SUMBIN) (IB-502)

Sum Binary Numbers sums up a list of consecutive 16-bit WORD Binary numbers into a 32-bit DWORD binary result.

230

240

250-1

260

262

You specify the group's starting and ending V-memory addresses (inclusive). When enabled, this instruction will add all the numbers in the group (so you may want to place a differential contact driving the enable).

SUMBIN could be used as the first part of calculating an average.

SUMBIN Parameters

DS5	Used
HPP	N/A

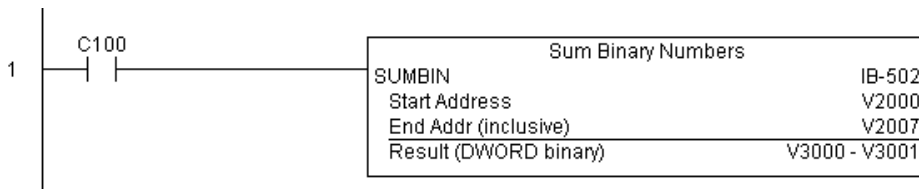
- Start Address: specifies the starting address of a block of V-memory location values to be added together (Binary)
- End Addr (inclusive): specifies the ending address of a block of V-memory location values to be added together (Binary)
- Result (DWORD Binary): specifies the location where the sum of the block of V-memory binary values will be placed

Sum Binary Numbers	
SUMBIN	IB-502
Start Address	V2000
End Addr (inclusive)	V2007
Result (DWORD binary)	V3000

Parameter		DL205 Range
Start Address	V	See DL205 V-memory map-Data Words
End Address (inclusive)	V	See DL205 V-memory map-Data Words
Result (DWORD Binary)	V	See DL205 V-memory map-Data Words

SUMBIN Example

In the following example, the SUMBIN instruction is used to total the sum of all Binary values in words V2000 thru V2007 and store the resulting 8-digit double word Binary value in V3000 and V3001 when C100 turns on.



Sum Real Numbers (SUMR) (IB-542)

230

Sum Real Numbers sums up a list of consecutive REAL DWORD numbers into a REAL DWORD result.

240

You specify the group's starting and ending V-memory addresses (inclusive).

250-1

Remember that Real numbers are DWORDs and occupy 2 words of V-memory each, so the number of Real values summed up is equal to half the number of memory locations.

260

Note that the End Address can be EITHER word of the 2 word ending address, for example, if you wanted to add the 4 Real numbers stored in V2000 thru V2007 (V2000, V2002, V2004, and V2006), you can specify V2006 OR V2007 for the ending address and

262

you will get the same result.

DS5	Used
HPP	N/A

When enabled, this instruction will add all the numbers in the group (so you may want to place a differential contact driving the enable).

SUMR could be used as the first part of calculating an average.

SUMR Parameters

- Start Address (DWORD): specifies the starting address of a block of V-memory location values to be added together (Real)
- End Addr (inclusive) (DWORD): specifies the ending address of a block of V-memory location values to be added together (Real)
- Result (DWORD): specifies the location where the sum of the block of V-memory Real values will be placed

✓✕⌂

Sum Real Numbers

SUMR IB-542

Start Address (DWORD) V2000

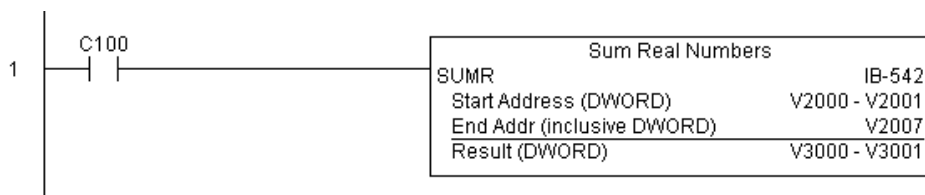
End Addr (inclusive DWORD) V2007

Result (DWORD) V3000

Parameter		DL205 Range
Start Address (DWORD)	V	See DL205 V-memory map-Data Words
End Address (inclusive DWORD)	V	See DL205 V-memory map-Data Words
Result (DWORD)	V	See DL205 V-memory map-Data Words

SUMR Example

In the following example, the SUMR instruction is used to total the sum of all floating point REAL number values in words V2000 thru V2007 and store the resulting 32-bit floating point REAL number value in V3000 and V3001 when C100 turns on.



ECOM100 Configuration (ECOM100) (IB-710)

230

240

250-1

260

262

ECOM100 Configuration defines all the common information for one specific ECOM100 module which is used by the other ECOM100 IBoxes; for example, ECRX–ECOM100 Network Read, ECEMAIL–ECOM100 Send Email, ECIPSUP–ECOM100 IP Setup, etc.

You MUST have the ECOM100 Configuration IBox at the top of your ladder/stage program with any other configuration IBoxes. The Message Buffer parameter specifies the starting address of a 65 WORD buffer. This is 101 Octal addresses (e.g., V1400 thru V1500).

If you have more than one ECOM100 in your PLC, you must have a different ECOM100 Configuration IBox for EACH ECOM100 module in your system that utilizes any ECOM IBox instructions.

DS5	Used
HPP	N/A

The Workspace and Status parameters and the entire Message Buffer are internal, private registers used by the ECOM100 Configuration IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

In order for MOST ECOM100 IBoxes to function, you must turn ON dip switch 7 on the ECOM100 circuit board. You can keep dip switch 7 off if you are ONLY using ECOM100 Network Read and Write IBoxes (ECRX, ECWX).

ECOM100 Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number.
- Slot: specifies which PLC slot is occupied by the ECOM100 module.
- Status: specifies a V-memory location that will be used by the instruction.
- Workspace: specifies a V-memory location that will be used by the instruction.
- Msg Buffer: specifies the starting address of a 65-word buffer that will be used by the module for configuration.

Parameter		DL205 Range
ECOM100#	K	K0-255
Slot	K	K0-7
Status	V	See DL205 V-memory map–Data Words
Workspace	V	See DL205 V-memory map–Data Words
Msg Buffer (65 words used)	V	See DL205 V-memory map–Data Words

ECOM100 Example

The ECOM100 Config IBox coordinates all of the interaction with other ECOM100-based IBoxes (ECxxxx). You must have an ECOM100 Config IBox for each ECOM100 module in your system. Configuration IBoxes must be at the top of your program and must execute every scan.

This IBox defines ECOM100# K0 to be in slot 3. Any ECOM100 IBoxes that need to reference this specific module (such as ECEMAIL, ECRX, ...) would enter K0 for their ECOM100# parameter.

The Status register is for reporting any completion or error information to other ECOM100 IBoxes. This V-memory register must not be used anywhere else in the entire program.

The Workspace register is used to maintain state information about the ECOM100, along with proper sharing and interlocking with the other ECOM100 IBoxes in the program. This V-memory register must not be used anywhere else in the entire program.

The Message Buffer of 65 words (130 bytes) is a common pool of memory that is used by other ECOM100 IBoxes (such as ECEMAIL). This way, you can have a bunch of ECEMAIL IBoxes, but only need 1 common buffer for generating and sending each EMail. These V-memory registers must not be used anywhere else in your entire program.

1	<i>ECOM100 Config</i>	
	ECOM100	IB-710
	ECOM100 #	K0
	Slot	K3
	Status	V1501
	Workspace	V1502
	Msg Buffer (65 WORDs)	V1400 - V1500

No permissive contact or input logic is used with this instruction



NOTE: An ECOM100 IBox instruction is used without a permissive contact. The top line will be identified in **BOLD italics**, and the instruction name and ID will be in **BOLD** characters.

ECOM100 Disable DHCP (ECDHCPD) (IB-736)

230

240

250-1

260

262

ECOM100 Disable DHCP will set up the ECOM100 to use its internal TCP/IP settings on a leading edge transition to the IBox. To configure the ECOM100's TCP/IP settings manually, use the NetEdit3 utility, or you can do it programmatically from your PLC program using the ECOM100 IP Setup (ECIPSUP), or the individual ECOM100 IBoxes: ECOM Write IP Address (ECWRIP), ECOM Write Gateway Address (ECWRGWA), and ECOM100 Write Subnet Mask (ECWRSNM).

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

DS5	Used
HPP	N/A

Either the Success or Error bit parameter will turn on once the command is complete. If there is an error, the Error Code parameter will report an ECOM100 error code (less than 100), or a PLC logic error (greater than 1000).

The "Disable DHCP" setting is stored in Flash-ROM in the ECOM100 and the execution of this IBox will disable the ECOM100 module for at least a half second until it writes the Flash-ROM. Therefore, it is HIGHLY RECOMMENDED that you only execute this IBox ONCE, on the second scan. Since it requires a LEADING edge to execute, use a NORMALLY CLOSED SP0 (STR NOT First Scan) to drive the power flow to the IBox.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.

ECDHCPD Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not completed successfully
- Error Code: specifies the location where the Error Code will be written

ECOM100 Disable DHCP
IB-736

ECDHCPD

ECOM100 # K0

Workspace V503

Success C100

Error C101

Error Code V2000

Parameter		DL205 Range
ECOM100#	K	K0-255
Workspace	V	See DL205 V-memory map-Data Words
Success	X,Y,C,GX,GY,B	See DL205 V-memory map
Error	X,Y,C,GX,GY,B	See DL205 V-memory map
Error Code	V	See DL205 V-memory map-Data Words

ECDHCPD Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module. V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130-byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.



NOTE: An ECOM100 IBox instruction is used without a permissive contact. The top line will be identified in **BOLD italics**, and the instruction name and ID will be in **BOLD characters**.

Rung 2: On the 2nd scan, disable DHCP in the ECOM100. DHCP is the same protocol used by PCs for using a DHCP Server to automatically assign the ECOM100's IP Address, Gateway Address, and Subnet Mask. Typically disabling DHCP is done by assigning a hard-coded IP Address either in NetEdit or using one of the ECOM100 IP Setup IBoxes, but this IBox allows you to disable DHCP in the ECOM100 using your ladder program. The ECDHCPD is leading edge triggered, not power-flow driven (similar to a counter input leg). The command to disable DHCP will be sent to the ECOM100 whenever the power flow into the IBox goes from OFF to ON. If successful, turn on C100. If there is a failure, turn on C101. If it fails, you can look at V2000 for the specific error code.



ECOM100 Enable DHCP (ECDHCPE) (IB-735)

✗ 230

✗ 240

✓ 250-1

✓ 260

✓ 262

ECOM100 Enable DHCP will tell the ECOM100 to obtain its TCP/IP setup from a DHCP Server on a leading edge transition to the IBox.

The IBox will be successful once the ECOM100 has received its TCP/IP settings from the DHCP server. Since it is possible for the DHCP server to be unavailable, a Timeout parameter is provided so that the IBox can complete, but with an Error (Error Code = 1004 decimal).

See also the ECOM100 IP Setup (ECIPSUP) IBox 717 to directly set up ALL of the TCP/IP parameters in a single instruction—IP Address, Subnet Mask, and Gateway Address.

DS5	Used
HPP	N/A

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Either the Success or Error bit parameter will turn on once the command is complete. If there is an error, the Error Code parameter will report an ECOM100 error code (less than 100), or a PLC logic error (greater than 1000).

The "Enable DHCP" setting is stored in Flash-ROM in the ECOM100 and the execution of this IBox will disable the ECOM100 module for at least a half second until it writes the Flash-ROM. Therefore, it is HIGHLY RECOMMENDED that you only execute this IBox ONCE, on the second scan. Since it requires a LEADING edge to execute, use a NORMALLY CLOSED SP0 (STR NOT First Scan) to drive the power flow to the IBox.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.

ECDHCPE Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number.
- Timeout(sec): specifies a timeout period so that the instruction may have time to complete.
- Workspace: specifies a V-memory location that will be used by the instruction.
- Success: specifies a bit that will turn on once the request is completed successfully.
- Error: specifies a bit that will turn on if the instruction is not completed successfully.
- Error Code: specifies the location where the Error Code will be written.

ECOM100 Enable DHCP IB-735

ECDHCPE	
ECOM100 #	K0
Timeout(sec.)	K5
Workspace	V400
Success	C0
Error	C0
Error Code	V400

Parameter		DL205 Range
ECOM100#	K	K0-255
Timeout (sec)	K	K5-127
Workspace	V	See DL205 V-memory map—Data Words
Success	X,Y,C,GX,GY,B	See DL205 V-memory map
Error	X,Y,C,GX,GY,B	See DL205 V-memory map
Error Code	V	See DL205 V-memory map—Data Words

ECDHCPE Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module. V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130-byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.



NOTE: An ECOM100 IBox instruction is used without a permissive contact. The top line will be identified in **BOLD italics**, and the instruction name and ID will be in **BOLD** characters.

Rung 2: On the second scan, enable DHCP in the ECOM100. DHCP is the same protocol used by PCs for using a DHCP Server to automatically assign the ECOM100's IP Address, Gateway Address, and Subnet Mask. Typically this is done using NetEdit, but this IBox allows you to enable DHCP in the ECOM100 using your ladder program. The ECDHCPE is leading edge triggered, not power-flow driven (similar to a counter input leg). The commands to enable DHCP will be sent to the ECOM100 whenever the power flow into the IBox goes from OFF to ON. The ECDHCPE does more than just set the bit to enable DHCP in the ECOM100, it polls the ECOM100 once every second to see if the ECOM100 has found a DHCP server and has a valid IP Address. Therefore, a timeout parameter is needed in case the ECOM100 cannot find a DHCP server. If a timeout does occur, the Error bit will turn on and the error code will be 1005 decimal. The Success bit will turn on only if the ECOM100 finds a DHCP Server and is assigned a valid IP Address. If successful, turn on C100. If there is a failure, turn on C101. If it fails, you can look at V2000 for the specific error code.



ECOM100 Query DHCP Setting (ECDHCPQ) (IB-734)

ECOM100 Query DHCP Setting will determine if DHCP is enabled in the ECOM100 on a leading edge transition to the IBox. The DHCP Enabled bit parameter will be ON if DHCP is enabled, OFF if disabled.

The Workspace parameter is an internal, private register used by this IBox and **MUST BE UNIQUE** in this one instruction and **MUST NOT** be used anywhere else in your program.

Either the Success or Error bit parameter will turn on once the command is complete.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.

- ☐ 230
- ☐ 240
- ☒ 250-1
- ☒ 260
- ☒ 262

DS5	Used
HPP	N/A

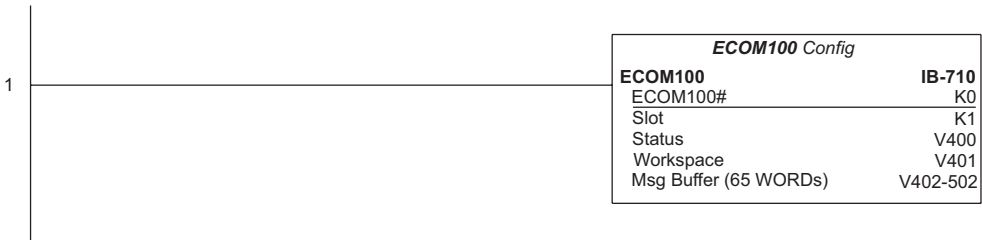
ECDHCPQ Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxx IBoxes that need to reference this ECOM100 module must reference this logical number.
- Workspace: specifies a V-memory location that will be used by the instruction.
- Success: specifies a bit that will turn on once the instruction is completed successfully.
- Error: specifies a bit that will turn on if the instruction is not completed successfully.
- DHCP Enabled: specifies a bit that will turn on if the ECOM100's DHCP is enabled or remain off if disabled—after instruction query, be sure to check the state of the Success/Error bit state along with DHCP Enabled bit state to confirm a successful module query.

Parameter		DL205 Range
ECOM100#	K	K0-255
Workspace	V	See DL205 V-memory map—Data Words
Success	X,Y,C,GX,GY,B	See DL205 V-memory map
Error	X,Y,C,GX,GY,B	See DL205 V-memory map
DHCP Enabled	X,Y,C,GX,GY,B	See DL205 V-memory map

ECDHCPQ Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module. V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130-byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.



NOTE: An ECOM100 IBox instruction is used without a permissive contact. The top line will be identified in **BOLD italics**, and the instruction name and ID will be in **BOLD characters**.

Rung 2: On the second scan, read whether DHCP is enabled or disabled in the ECOM100 and store it in C5. DHCP is the same protocol used by PCs for using a DHCP Server to automatically assign the ECOM100's IP Address, Gateway Address, and Subnet Mask. The ECDHCPQ is leading edge triggered, not power-flow driven (similar to a counter input leg). The command to read (Query) whether DHCP is enabled or not will be sent to the ECOM100 whenever the power flow into the IBox goes from OFF to ON. If successful, turn on C100. If there is a failure, turn on C101.



ECOM100 Send E-mail (ECEMAIL) (IB-711)

- 230
- 240
- 250-1
- 260
- 262

DS5	Used
HPP	N/A

ECOM100 Send EMail, on a leading edge transition, will behave as an EMail client and send an SMTP request to your SMTP Server to send the EMail message to the EMail addresses in the To: field and also to those listed in the Cc: list hard coded in the ECOM100. It will send the SMTP request based on the specified ECOM100#, which corresponds to a specific unique ECOM100 Configuration (ECOM100) at the top of your program.

The Body: field supports what the PRINT and VPRINT instructions support for text and embedded variables, allowing you to embed real-time data in your EMail (e.g., "V2000 = " V2000:B).

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Either the Success or Error bit parameter will turn on once the request is complete. If there is an error, the Error Code parameter will report an ECOM100 error code (less than 100), an SMPT protocol error (between 100 and 999), or a PLC logic error (greater than 1000).

Since the ECOM100 is only an EMail Client and requires access to an SMTP Server, you MUST have the SMTP parameters configured properly in the ECOM100 via the ECOM100's Home Page and/or the EMail Setup instruction (ECEMSUP). To get to the ECOM100's Home Page, use your favorite Internet browser and browse to the ECOM100's IP Address, e.g., <http://192.168.12.86>

You are limited to approximately 100 characters of message data for the entire instruction, including the To: Subject: and Body: fields. To save space, the ECOM100 supports a hard coded list of EMail addresses for the Carbon Copy field (cc:) so that you can configure those IN the ECOM100, and keep the To: field small (or even empty), to leave more room for the Subject: and Body: fields.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.

ECEMAIL Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not completed successfully
- Error Code: specifies the location where the Error Code will be written
- To: specifies an E-mail address that the message will be sent to
- Subject: subject of the e-mail message
- Body: supports what the PRINT and VPRINT instructions support for text and embedded variables, allowing you to embed real-time data in the EMail message

Parameter		DL205 Range
ECOM100#	K	K0-255
Workspace	V	See DL205 V-memory map–Data Words
Success	X,Y,C,GX,GY,B	See DL205 V-memory map
Error	X,Y,C,GX,GY,B	See DL205 V-memory map
Error Code	V	See DL205 V-memory map
To:		Text
Subject:		Text
Body:		See PRINT and VPRINT instructions

ECEMAIL Decimal Status Codes

This list of status codes is based on the list in the ECOM100 Mock Slave Address 89 Command Specification.

ECOM100 Status codes can be classified into four different areas based on its decimal value.

ECOM100 Status Code Areas	
Error	Description
0-1	Normal Status—no error
2-99	Internal ECOM100 errors
100-999	Standard TCP/IP protocol errors (SMTP, HTTP, etc)
1000+	IBox ladder logic assigned errors (SP Slot Error, etc)

For the ECOM100 Send Email IBOX, the status codes below are specific to this IBox.

Normal Status 0-1

ECOMM100 Send EMAIL IBOX Status Codes	
Error	Description
0-1	Success—ECEMAIL completed successfully.
1	Busy—ECEMAIL IBOX logic sets the Error register to this value when the ECEMAIL starts a new request.

Internal ECOM100 Errors (2-99)

Internal ECOM 100 Errors (2-99)	
Error	Description
10-19	Timeout Errors—last digit shows where in ECOM100's SMTP state logic the timeout occurred; regardless of the last digit, the SMTP conversation with the SMTP Server timed out.
	SMTP Internal Errors (20-29)
20	TCP Write Error
21	No Sendee
22	Invalid State
23	Invalid Data
24	Invalid SMTP Configuration
25	Memory Allocation Error

ECEMAIL IBox Ladder Logic Assigned Errors (1000+)

ECEMAIL IBox Ladder Logic Assigned Errors (1000+)	
Error	Description
101	SP Slot Error—the SP error bit for the ECOM100's slot turned on. Possibly using RX or WX instructions on the ECOM100 and walking on the ECEMAIL execution. Use should use ECRX and ECX IBoxes,

ECEMAIL Decimal Status Codes

SMTP Protocol Errors–SMTP (100-999)

SMTP Protocol Errors–SMTP (100-999)	
Error	Description
1xx	Informational replies
2xx	Success replies
200	(Non-standard success response)
211	System status or system help reply
214	Help message
220	<domain> Service ready. Ready to start TLS
221	<domain> Service closing transmission channel
250	Ok, queuing for node <node> started. Requested mail action okay, completed
251	Ok, no messages waiting for node <node>. User not local; will forward to <forward-path>
252	OK, pending messages for node <node> started. Cannot VRFY user (e.g., info is not local), but will take message for this user and attempt delivery.
253	OK, messages pending messages for node <node> started
3xx`	(re) direction replies
354	Start mail input; end with <CRLF>.<CRLF>
355	Octet-offset is the transaction offset.
4xx	client/request error replies
421	<domain> Service not available, closing transmission channel
432	A password transition is needed
450	Requested mail action not taken: mailbox unavailable. ATRN request refused.
451	Requested action aborted; local error in processing. Unable to process ATRN request now.
452	Requested action not taken: insufficient system storage
453	You have no mail
454	TLS is not available due to temporary reason. Encryption required for requested authentication mechanism.
458	Unable to queue messages for node <node>
459	Node <node> not allowed: <reason>
5xx	Server/process error replies
500	Syntax error, command unrecognized. Syntax error.
501	Syntax error in parameters or arguments
502	Command not implemented
503	Bad sequence of commands
504	Command parameter not implemented
521	<domain> does not accept mail
530	Access denied. Must issue a STARTTLS command first. Encryption required for requested authentication mechanism.
534	Authentication mechanism is too weak.
Continued next page	

SMTP Protocol Errors-SMTP (100-999)	
Error	Description
538	Encryption required for requested authentication mechanism.
550	Requested action not taken; mailbox unavailable.
551	User not local; please try <forward path>
552	Requested mail action aborted; exceeded storage allocation
553	Requested action not taken; mailbox name not allowed.
554	Transaction failed

ECEMAIL Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module. V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130-byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.



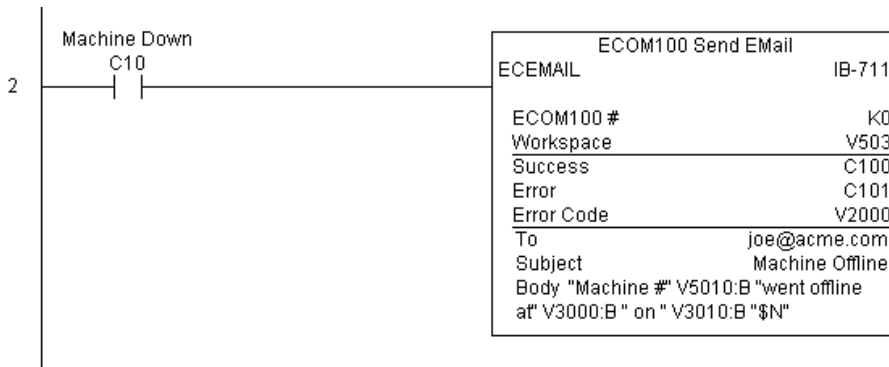
NOTE: An ECOM100 IBox instruction is used without a permissive contact. The top line will be identified in **BOLD italics**, and the instruction name and ID will be in **BOLD characters**.



Rung 2: When a machine goes down, send an email to Joe in maintenance and to the VP over production showing what machine is down along with the date/time stamp of when it went down.

The ECEMAIL is leading edge triggered, not power-flow driven (similar to a counter input leg). An email will be sent whenever the power flow into the IBox goes from OFF to ON. This helps prevent self-inflicted spamming.

If the EMail is sent, turn on C100. If there is a failure, turn on C101. If it fails, you can look at V2000 for the SMTP error code or other possible error codes.



ECOM100 Restore Default E-mail Setup (ECEMRDS) (IB-713)

230

240

250-1

260

262

DS5	Used
HPP	N/A

ECOM100 Restore Default EMail Setup, on a leading edge transition, will restore the original EMail Setup data stored in the ECOM100 back to the working copy based on the specified ECOM100#, which corresponds to a specific unique ECOM100 Configuration (ECOM100) at the top of your program.

When the ECOM100 is first powered up, it copies the EMail setup data stored in ROM to the working copy in RAM. You can then modify this working copy from your program using the ECOM100 EMail Setup (ECEMSUP) IBox. After modifying the working copy, you can later restore the original setup data via your program by using this IBox.

The Workspace parameter is an internal, private register used by this IBox and **MUST BE UNIQUE** in this one instruction and **MUST NOT** be used anywhere else in your program.

Either the Success or Error bit parameter will turn on once the command is complete. If there is an error, the Error Code parameter will report an ECOM100 error code (less than 100), or a PLC logic error (greater than 1000).

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.

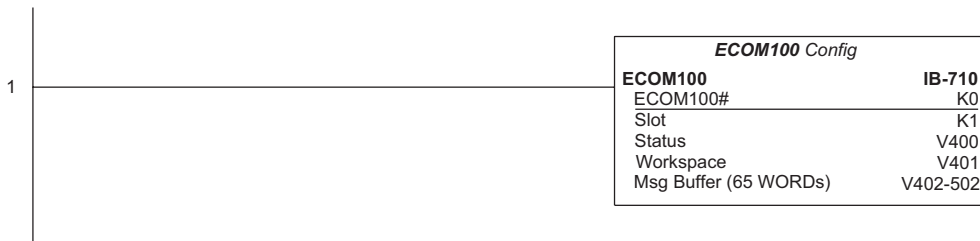
ECEMRDS Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not completed successfully
- Error Code: specifies the location where the Error Code will be written

Parameter		DL205 Range
ECOM100#	K	K0-255
Workspace	V	See DL205 V-memory map—Data Words
Success	X,Y,C,GX,GY,B	See DL205 V-memory map
Error	X,Y,C,GX,GY,B	See DL205 V-memory map
Error Code	V	See DL205 V-memory map—Data Words

ECEMRDS Example

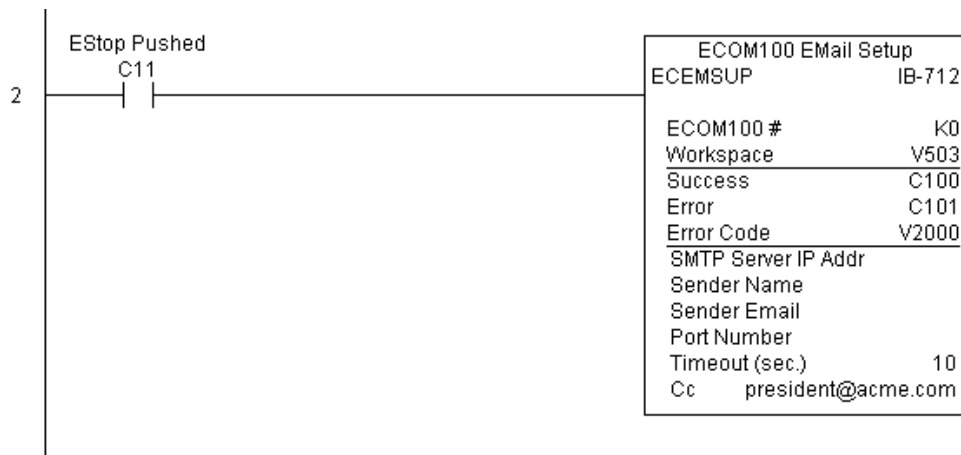
Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module. V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130-byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.



NOTE: An ECOM100 IBox instruction is used without a permissive contact. The top line will be identified in **BOLD italics**, and the instruction name and ID will be in **BOLD** characters.

Rung 2: Whenever an EStop is pushed, ensure that the president of the company gets copies of all Emails being sent.

The ECOM100 Email Setup IBox allows you to set/change the SMTP Email settings stored in the ECOM100.



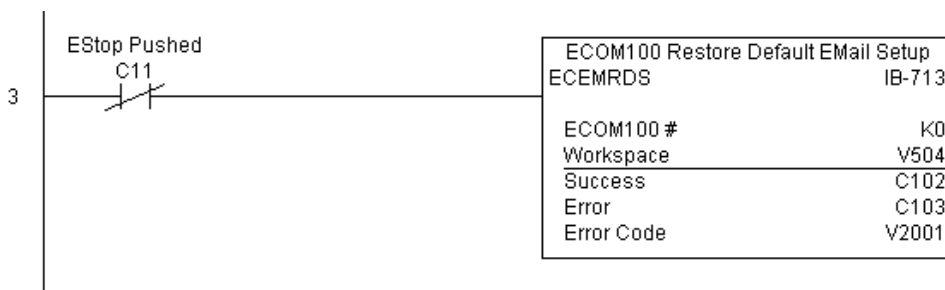
(Example continued on next page)

ECEMRDS Example (cont'd)

Rung 3: Once the EStop is pulled out, take the president off the cc: list by restoring the default EMail setup in the ECOM100.

The ECEMRDS is leading edge triggered, not power-flow driven (similar to a counter input leg). The ROM-based EMail configuration stored in the ECOM100 will be copied over the "working copy" whenever the power flow into the IBox goes from OFF to ON (the working copy can be changed by using the ECEMSUP IBox).

If successful, turn on C102. If there is a failure, turn on C103. If it fails, you can look at V2001 for the specific error code.



ECOM100 E-mail Setup (ECEMSUP) (IB-712)

✕ 230

✕ 240

✓ 250-1

✓ 260

✓ 262

DS5	Used
HPP	N/A

ECOM100 EMail Setup, on a leading edge transition, will modify the working copy of the EMail setup currently in the ECOM100 based on the specified ECOM100#, which corresponds to a specific unique ECOM100 Configuration (ECOM100) at the top of your program.

You may pick and choose any or all fields to be modified using this instruction. Note that these changes are cumulative: if you execute multiple ECOM100 EMail Setup IBoxes, then all of the changes are made in the order they are executed. Also note that you can restore the original ECOM100 EMail Setup that is stored in the ECOM100 to the working copy by using the ECOM100 Restore Default EMail Setup (ECEMRDS) IBox.

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Either the Success or Error bit parameter will turn on once the command is complete. If there is an error, the Error Code parameter will report an ECOM100 error code (less than 100), or a PLC logic error (greater than 1000).

You are limited to approximately 100 characters/bytes of setup data for the entire instruction. So if needed, you could divide the entire setup across multiple ECEMSUP IBoxes on a field-by-field basis, for example do the Carbon Copy (cc:) field in one ECEMSUP IBox and the remaining setup parameters in another.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.

ECEMSUP Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number.
- Workspace: specifies a V-memory location that will be used by the instruction.
- Success: specifies a bit that will turn on once the request is completed successfully.
- Error: specifies a bit that will turn on if the instruction is not completed successfully.
- Error Code: specifies the location where the Error Code will be written.
- SMTP Server IP Addr: optional parameter that specifies the IP Address of the SMTP Server on the ECOM100's network.
- Sender Name: optional parameter that specifies the sender name that will appear in the "From:" field to those who receive the e-mail.
- Sender EMail: optional parameter that specifies the sender EMail address that will appear in the "From:" field to those who receive the e-mail.

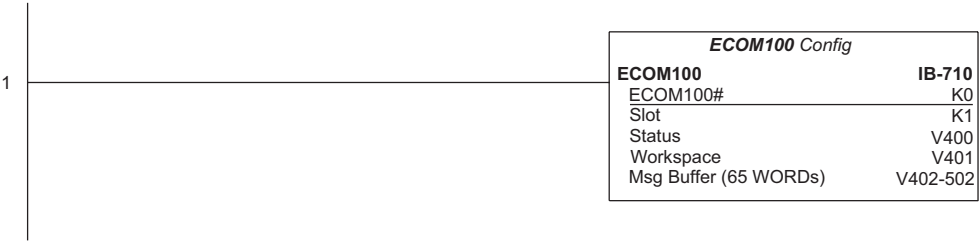
ECEMSUP Parameters (cont'd)

- Port Number: optional parameter that specifies the TCP/IP Port Number to send SMTP requests; usually this does not need to be configured (see your network administrator for information on this setting).
- Timeout (sec): optional parameter that specifies the number of seconds to wait for the SMTP Server to send the EMail to all the recipients.
- Cc: optional parameter that specifies a list of "carbon copy" Email addresses to send all EMail to.

Parameter		DL205 Range
ECOM100#	K	K0-255
Workspace	V	See DL205 V-memory map-Data Words
Success	X,Y,C,GX,GY,B	See DL205 V-memory map
Error	X,Y,C,GX,GY,B	See DL205 V-memory map
Error Code	V	See DL205 V-memory map-Data Words

ECEMSUP Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module. V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130-byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.



(Example continued on next page)

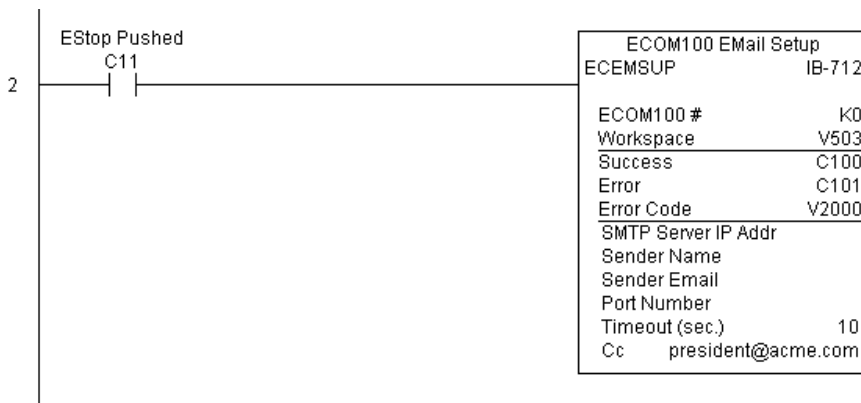


NOTE: An ECOM100 IBox instruction is used without a permissive contact. The top line will be identified in ***BOLD italics***, and the instruction name and ID will be in **BOLD** characters.

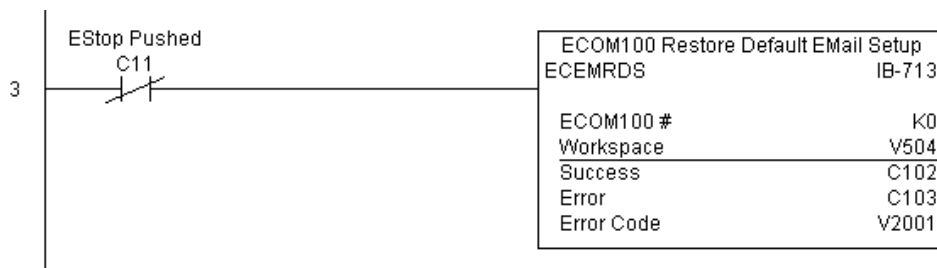
ECEMSUP Example (cont'd)

Rung 2: Whenever an EStop is pushed, ensure that president of the company gets copies of all EMail being sent. The ECOM100 EMail Setup IBox allows you to set/change the SMTP EMail settings stored in the ECOM100. The ECEMSUP is leading edge triggered, not power-flow driven (similar to a counter input leg). At power-up, the ROM-based EMail configuration stored in the ECOM100 is copied to a RAM-based "working copy". You can change this working copy by using the ECEMSUP IBox. To restore the original ROM-based configuration, use the Restore Default EMail Setup ECEMRDS IBox.

If successful, turn on C100. If there is a failure, turn on C101. If it fails, you can look at V2000 for the specific error code.



Rung 3: Once the EStop is pulled out, take the president off the cc: list by restoring the default EMail setup in the ECOM100.



ECOM100 IP Setup (ECIPSUP) (IB-717)

✗ 230

✗ 240

✓ 250-1

✓ 260

✓ 262

ECOM100 IP Setup will configure the three TCP/IP parameters in the ECOM100: IP Address, Subnet Mask, and Gateway Address, on a leading edge transition to the IBox. The ECOM100 is specified by the ECOM100#, which corresponds to a specific unique ECOM100 Configuration (ECOM100) IBox at the top of your program.

The Workspace parameter is an internal, private register used by this IBox and **MUST BE UNIQUE** in this one instruction and **MUST NOT** be used anywhere else in your program.

Either the Success or Error bit parameter will turn on once the command is complete. If there is an error, the Error Code parameter will report an ECOM100 error code (less than 100), or a PLC logic error (greater than 1000).

This setup data is stored in Flash-ROM in the ECOM100 and will disable the ECOM100 module for at least a half second until it writes the Flash-ROM. Therefore, it is **HIGHLY RECOMMENDED** that you only execute this IBox **ONCE** on the second scan. Since it requires a **LEADING** edge to execute, use a **NORMALLY CLOSED SP0** (NOT First Scan) to drive the power flow to the IBox.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.

DS5	Used
HPP	N/A

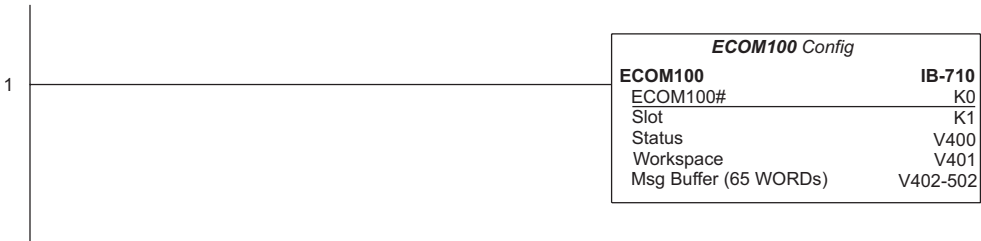
ECIPSUP Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number.
- Workspace: specifies a V-memory location that will be used by the instruction.
- Success: specifies a bit that will turn on once the request is completed successfully.
- Error: specifies a bit that will turn on if the instruction is not completed successfully.
- Error Code: specifies the location where the Error Code will be written.
- IP Address: specifies the module's IP Address.
- Subnet Mask: specifies the Subnet Mask for the module to use.
- Gateway Address: specifies the Gateway Address for the module to use.

Parameter		DL205 Range
ECOM100#	K	K0-255
Workspace	V	See DL205 V-memory map-Data Words
Success	X,Y,C,GX,GY,B	See DL205 V-memory map
Error	X,Y,C,GX,GY,B	See DL205 V-memory map
Error Code	V	See DL205 V-memory map-Data Words
IP Address	IP Address	0.0.0.1. to 255.255.255.254
Subnet Mask Address	IP Address Mask	0.0.0.1. to 255.255.255.254
Gateway Address	IP Address	0.0.0.1. to 255.255.255.254

ECIPSUP Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module. V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130-byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.

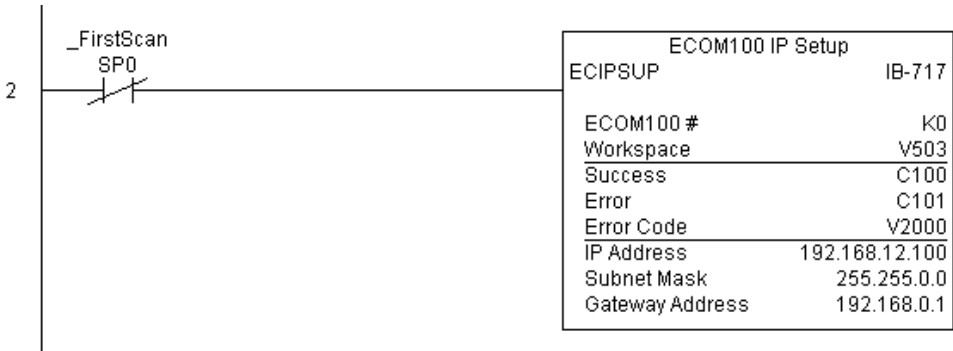


NOTE: An ECOM100 IBox instruction is used without a permissive contact. The top line will be identified in **BOLD italics**, and the instruction name and ID will be in **BOLD** characters.

Rung 2: On the second scan, configure all of the TCP/IP parameters in the ECOM100:
IP Address: 192.168.12.100
Subnet Mask: 255.255.0.0
Gateway Address: 192.168.0.1

The ECIPSUP is leading edge triggered, not power-flow driven (similar to a counter input leg). The command to write the TCP/IP configuration parameters will be sent to the ECOM100 whenever the power flow into the IBox goes from OFF to ON.

If successful, turn on C100. If there is a failure, turn on C101. If it fails, you can look at V2000 for the specific error code.



ECOM100 Read Description (ECRDDES) (IB-726)

ECOM100 Read Description will read the ECOM100's Description field up to the number of specified characters on a leading edge transition to the IBox.

✗ 230

✗ 240

✓ 250-1

✓ 260

✓ 262

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Either the Success or Error bit parameter will turn on once the command is complete.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.

ECOM100 Read Description	
IB-726	
ECOM100 #	K0
Workspace	V400
Success	C0
Error	C0
Description	V400
Num Chars	K1

DS5	Used
HPP	N/A

ECRDDES Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not successfully completed
- Description: specifies the starting buffer location where the ECOM100's Description will be placed
- Num Chars: specifies the number of characters (bytes) to read from the ECOM100's Description field

Parameter		DL205 Range
ECOM100#	K	K0-255
Workspace	V	See DL205 V-memory map-Data Words
Success	X,Y,C,GX,GY,B	See DL205 V-memory map
Error	X,Y,C,GX,GY,B	See DL205 V-memory map
Description	V	See DL205 V-memory map-Data Words
Num Chars	K	K1-128

ECRDDES Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module. V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130-byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.



NOTE: An ECOM100 IBox instruction is used without a permissive contact. The top line will be identified in **BOLD italics**, and the instruction name and ID will be in **BOLD** characters.

Rung 2: On the 2nd scan, read the Module Description of the ECOM100 and store it in V3000 thru V3007 (16 characters). This text can be displayed by an HMI.

The ECRDDES is leading edge triggered, not power-flow driven (similar to a counter input leg). The command to read the module description will be sent to the ECOM100 whenever the power flow into the IBox goes from OFF to ON.

If successful, turn on C100. If there is a failure, turn on C101.



ECOM100 Read Gateway Address (ECRDGWA) (IB-730)

 230

 240

 250-1

 260

 262

ECOM100 Read Gateway Address will read the four parts of the Gateway IP address and store them in four consecutive V-memory locations in decimal format, on a leading edge transition to the IBox.

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

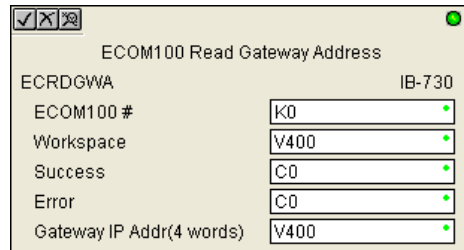
Either the Success or Error bit parameter will turn on once the command is complete.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.

DS5	Used
HPP	N/A

ECRDGWA Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxx IBoxes that need to reference this ECOM100 module must reference this logical number.
- Workspace: specifies a V-memory location that will be used by the instruction.
- Success: specifies a bit that will turn on once the request is completed successfully.
- Error: specifies a bit that will turn on if the instruction is not completed successfully.
- Gateway IP Addr: specifies the starting address where the ECOM100's Gateway Address will be placed in 4 consecutive V-memory locations.



Parameter		DL205 Range
ECOM100#	K	K0-255
Workspace	V	See DL205 V-memory map-Data Words
Success	X,Y,C,GX,GY,B	See DL205 V-memory map
Error	X,Y,C,GX,GY,B	See DL205 V-memory map
Gateway IP Address (4 Words)	V	See DL205 V-memory map-Data Words

ECRDGWA Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module. V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130-byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.

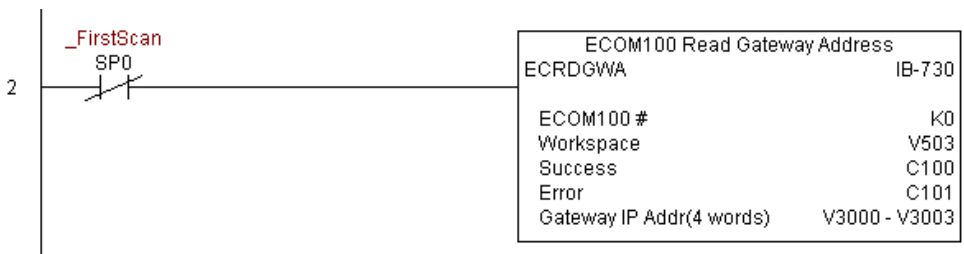


NOTE: An ECOM100 IBox instruction is used without a permissive contact. The top line will be identified in **BOLD** italics, and the instruction name and ID will be in **BOLD** characters.

Rung 2: On the second scan, read the Gateway Address of the ECOM100 and store it in V3000 thru V3003 (4 decimal numbers). The ECOM100's Gateway Address could be displayed by an HMI.

The ECRDGWA is leading edge triggered, not power-flow driven (similar to a counter input leg). The command to read the Gateway Address will be sent to the ECOM100 whenever the power flow into the IBox goes from OFF to ON.

If successful, turn on C100. If there is a failure, turn on C101.



ECOM100 Read IP Address (ECDIP) (IB-722)

 230

 240

 250-1

 260

 262

ECOM100 Read IP Address will read the four parts of the IP address and store them in four consecutive V-memory locations in decimal format, on a leading edge transition to the IBox.

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

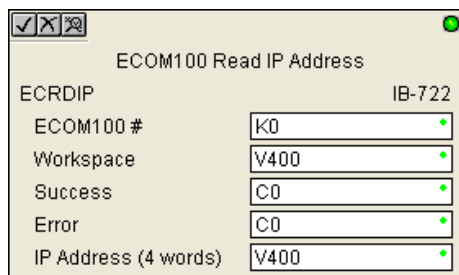
Either the Success or Error bit parameter will turn on once the command is complete.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.

DS5	Used
HPP	N/A

ECRDIP Parameters

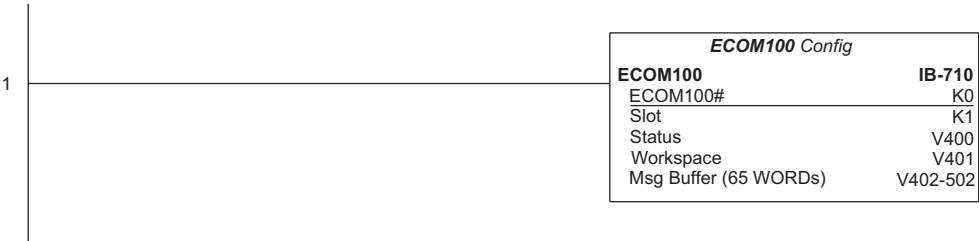
- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not completed successfully
- IP Address: specifies the starting address where the ECOM100's IP Address will be placed in four consecutive V-memory locations



Parameter		DL205 Range
ECOM100#	K	K0-255
Workspace	V	See DL205 V-memory map-Data Words
Success	X,Y,C,GX,GY,B	See DL205 V-memory map
Error	X,Y,C,GX,GY,B	See DL205 V-memory map
IP Address (4 Words)	V	See DL205 V-memory map-Data Words

ECRDIP Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module. V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130-byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.

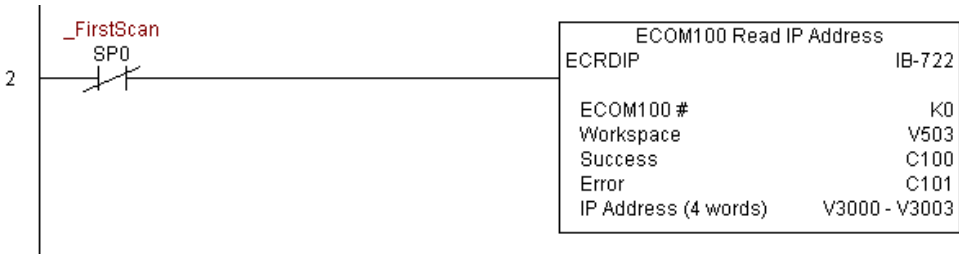


NOTE: An ECOM100 IBox instruction is used without a permissive contact. The top line will be identified in **BOLD italics**, and the instruction name and ID will be in **BOLD** characters.

Rung 2: On the second scan, read the IP Address of the ECOM100 and store it in V3000 thru V3003 (four decimal numbers). The ECOM100's IP Address could be displayed by an HMI.

The ECRDIP is leading edge triggered, not power-flow driven (similar to a counter input leg). The command to read the IP Address will be sent to the ECOM100 whenever the power flow into the IBox goes from OFF to ON.

If successful, turn on C100. If there is a failure, turn on C101.



ECOM100 Read Module ID (ECRDMID) (IB-720)

 230

 240

 250-1

 260

 262

ECOM100 Read Module ID will read the binary (decimal) WORD sized Module ID on a leading edge transition to the IBox.

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

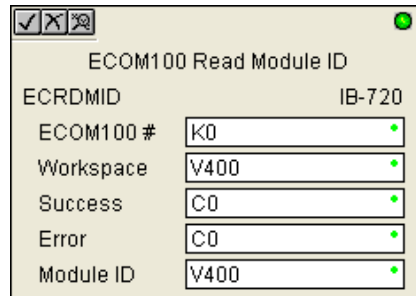
Either the Success or Error bit parameter will turn on once the command is complete.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.

DS5	Used
HPP	N/A

ECRDMID Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not completed successfully
- Module ID: specifies the location where the ECOM100's Module ID (decimal) will be placed



Parameter		DL205 Range
ECOM100#	K	K0-255
Workspace	V	See DL205 V-memory map-Data Words
Success	X,Y,C,GX,GY,B	See DL205 V-memory map
Error	X,Y,C,GX,GY,B	See DL205 V-memory map
Module ID	V	See DL205 V-memory map-Data Words

ECRDMID Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module. V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130-byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.



NOTE: An ECOM100 IBox instruction is used without a permissive contact. The top line will be identified in **BOLD italics**, and the instruction name and ID will be in **BOLD** characters.

Rung 2: On the second scan, read the Module ID of the ECOM100 and store it in V2000. The ECRDMID is leading edge triggered, not power-flow driven (similar to a counter input leg). The command to read the module ID will be sent to the ECOM100 whenever the power flow into the IBox goes from OFF to ON. If successful, turn on C100. If there is a failure, turn on C101.



ECOM100 Read Module Name (ECRDNAM) (IB-724)

ECOM100 Read Name will read the Module Name up to the number of specified characters on a leading edge transition to the IBox.

 230

 240

 250-1

 260

 262

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

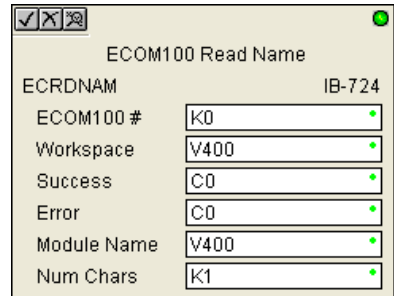
Either the Success or Error bit parameter will turn on once the command is complete.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.

DS5	Used
HPP	N/A

ECRDNAM Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxx IBoxes that need to reference this ECOM100 module must reference this logical number.
- Workspace: specifies a V-memory location that will be used by the instruction.
- Success: specifies a bit that will turn on once the request is completed successfully.
- Error: specifies a bit that will turn on if the instruction is not completed successfully.
- Module Name: specifies the starting buffer location where the ECOM100's Module Name will be placed.
- Num Chars: specifies the number of characters (bytes) to read from the ECOM100's Name field.



Parameter		DL205 Range
ECOM100#	K	K0-255
Workspace	V	See DL205 V-memory map–Data Words
Success	X,Y,C,GX,GY,B	See DL205 V-memory map
Error	X,Y,C,GX,GY,B	See DL205 V-memory map
Module Name	V	See DL205 V-memory map–Data Words
Num Chars	K	K1-128

ECRDNAM Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module. V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130-byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.

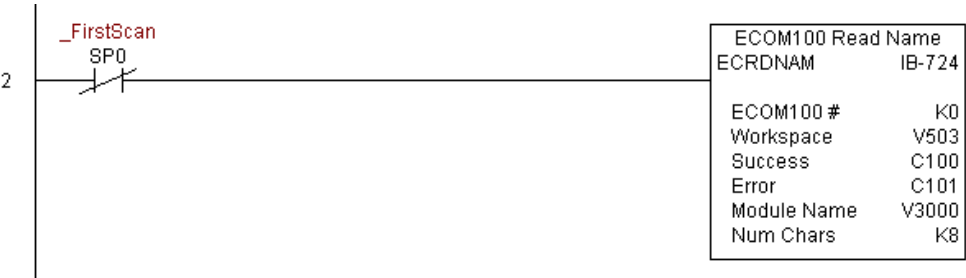


NOTE: An ECOM100 IBox instruction is used without a permissive contact. The top line will be identified in **BOLD italics**, and the instruction name and ID will be in **BOLD** characters.

Rung 2: On the second scan, read the Module Name of the ECOM100 and store it in V3000 thru V3003 (8 characters). This text can be displayed by an HMI.

The ECRDNAM is leading edge triggered, not power-flow driven (similar to a counter input leg). The command to read the module name will be sent to the ECOM100 whenever the power flow into the IBox goes from OFF to ON.

If successful, turn on C100. If there is a failure, turn on C101.



ECOM100 Read Subnet Mask (ECRDSNM) (IB-732)

ECOM100 Read Subnet Mask will read the four parts of the Subnet Mask and store them in 4 consecutive V-memory locations in decimal format, on a leading edge transition to the IBox.

☒ 230

☒ 240

☒ 250-1

☒ 260

☒ 262

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Either the Success or Error bit parameter will turn on once the command is complete.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.

DS5	Used
HPP	N/A

ECRDSNM Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxx IBoxes that need to reference this ECOM100 module must reference this logical number.
- Workspace: specifies a V-memory location that will be used by the instruction.
- Success: specifies a bit that will turn on once the request is completed successfully.
- Error: specifies a bit that will turn on if the instruction is not completed successfully.
- Subnet Mask: specifies the starting address where the ECOM100's Subnet Mask will be placed in four consecutive V-memory locations.

ECOM100 Read Subnet Mask
IB-732

ECOM100 #	K0	✓
Workspace	V400	✓
Success	C0	✓
Error	C0	✓
Subnet Mask (4 words)	V400	✓

Parameter		DL205 Range
ECOM100#	K	K0-255
Workspace	V	See DL205 V-memory map–Data Words
Success	X,Y,C,GX,GY,B	See DL205 V-memory map
Error	X,Y,C,GX,GY,B	See DL205 V-memory map
Subnet Mask (4 Words)	V	See DL205 V-memory map–Data Words

ECRDSNM Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module. V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130-byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.

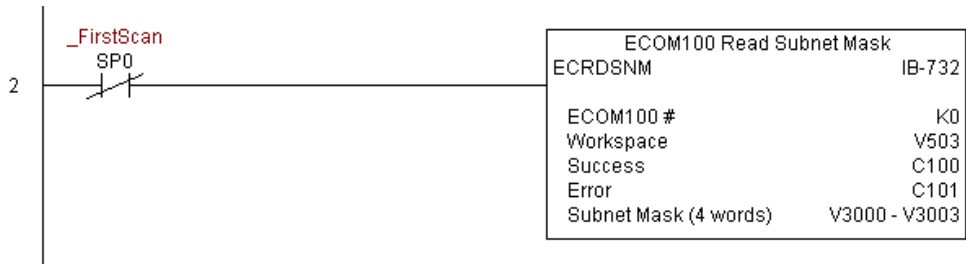


NOTE: An ECOM100 IBox instruction is used without a permissive contact. The top line will be identified in **BOLD italics**, and the instruction name and ID will be in **BOLD** characters.

Rung 2: On the second scan, read the Subnet Mask of the ECOM100 and store it in V3000 thru V3003 (4 decimal numbers). The ECOM100's Subnet Mask could be displayed by an HMI.

The ECRDSNM is leading edge triggered, not power-flow driven (similar to a counter input leg). The command to read the Subnet Mask will be sent to the ECOM100 whenever the power flow into the IBox goes from OFF to ON.

If successful, turn on C100. If there is a failure, turn on C101.



ECOM100 Write Description (ECWRDES) (IB-727)

✗ 230

✗ 240

✓ 250-1

✓ 260

✓ 262

ECOM100 Write Description will write the given Description to the ECOM100 module on a leading edge transition to the IBox. If you use a dollar sign (\$) or double quote ("), use the PRINT/VPRINT escape sequence of TWO dollar signs (\$\$) for a single dollar sign or dollar sign-double quote (\$") for a double quote character.

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Either the Success or Error bit parameter will turn on once the command is complete. If there is an error, the Error Code parameter will report an ECOM100 error code (less than 100), or a PLC logic error (greater than 1000).

DS5	Used
HPP	N/A

The Description is stored in Flash-ROM in the ECOM100 and the execution of this IBox will disable the ECOM100 module for at least a half second until it writes the Flash-ROM. Therefore, it is HIGHLY RECOMMENDED that you only execute this IBox ONCE on the second scan. Since it requires a LEADING edge to execute, use a NORMALLY CLOSED SPO (STR NOT First Scan) to drive the power flow to the IBox.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.

ECWRDES Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not completed successfully
- Error Code: specifies the location where the Error Code will be written
- Description: specifies the Description that will be written to the module

ECOM100 Write Description
IB-727

ECWRDES

ECOM100 # K0

Workspace V503

Success C100

Error C101

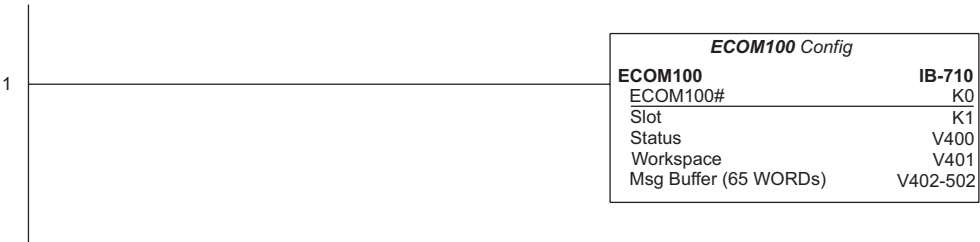
Error Code V2000

Description MODBUS/TCP Network #2

Parameter		DL205 Range
ECOM100#	K	K0-255
Workspace	V	See DL205 V-memory map—Data Words
Success	X,Y,C,GX,GY,B	See DL205 V-memory map
Error	X,Y,C,GX,GY,B	See DL205 V-memory map
Module Name	V	See DL205 V-memory map—Data Words
Description		Text

ECWRDES Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module. V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130-byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.



NOTE: An ECOM100 IBox instruction is used without a permissive contact. The top line will be identified in **BOLD italics**, and the instruction name and ID will be in **BOLD** characters.

Rung 2: On the second scan, set the Module Description of the ECOM100. Typically this is done using NetEdit, but this IBox allows you to configure the module description in the ECOM100 using your ladder program.

The ECWRDES is leading edge triggered, not power-flow driven (similar to a counter input leg). The command to write the module description will be sent to the ECOM100 whenever the power flow into the IBox goes from OFF to ON.

If successful, turn on C100. If there is a failure, turn on C101. If it fails, you can look at V2000 for the specific error code.



ECOM100 Write Gateway Address (ECWRGWA) (IB-731)

☒ 230

☒ 240

☒ 250-1

☒ 260

☒ 262

DS5	Used
HPP	N/A

ECOM100 Write Gateway Address will write the given Gateway IP Address to the ECOM100 module on a leading edge transition to the IBox. See also ECOM100 IP Setup (ECIPSUP) IBox 717 to set up ALL of the TCP/IP parameters in a single instruction—IP Address, Subnet Mask, and Gateway Address.

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Either the Success or Error bit parameter will turn on once the command is complete. If there is an error, the Error Code parameter will report an ECOM100 error code (less than 100), or a PLC logic error (greater than 1000).

The Gateway Address is stored in Flash-ROM in the ECOM100 and the execution of this IBox will disable the ECOM100 module for at least a half second until it writes the Flash-ROM. Therefore, it is HIGHLY RECOMMENDED that you only execute this IBox ONCE, on the second scan. Since it requires a LEADING edge to execute, use a NORMALLY CLOSED SPO (STR NOT First Scan) to drive the power flow to the IBox.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.

ECWRGWA Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not completed successfully
- Error Code: specifies the location where the Error Code will be written
- Gateway Address: specifies the Gateway IP Address that will be written to the module

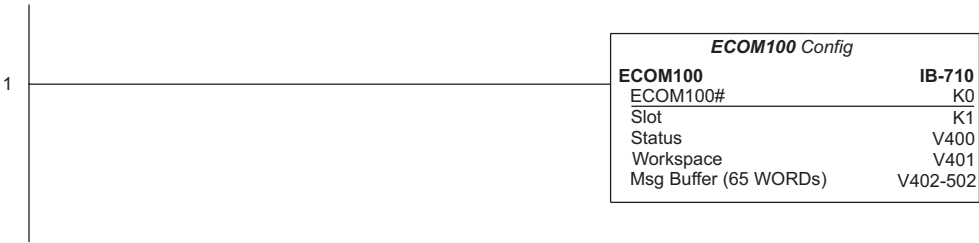
ECOM100 Write Gateway Address
IB-731

ECOM100 #	K0
Workspace	V503
Success	C100
Error	C101
Error Code	V2000
Gateway Address	192.168.0.1

Parameter		DL205 Range
ECOM100#	K	K0-255
Workspace	V	See DL205 V-memory map—Data Words
Success	X,Y,C,GX,GY,B	See DL205 V-memory map
Error	X,Y,C,GX,GY,B	See DL205 V-memory map
Error Code	V	See DL205 V-memory map—Data Words
Gateway Address		0.0.0.1. to 255.255.255.254

ECWRGWA Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module. V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130-byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.

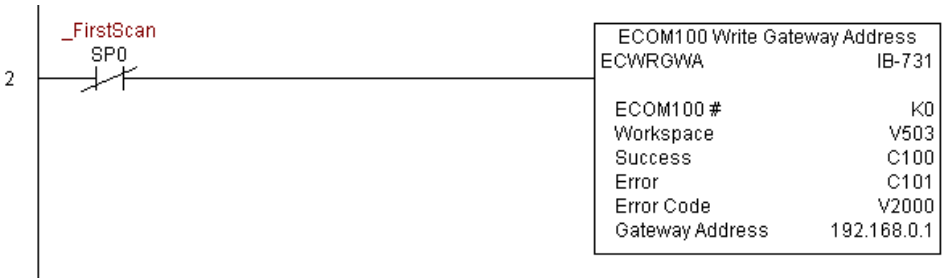


NOTE: An ECOM100 IBox instruction is used without a permissive contact. The top line will be identified in **BOLD italics**, and the instruction name and ID will be in **BOLD** characters.

Rung 2: On the second scan, assign the Gateway Address of the ECOM100 to 192.168.0.1. The ECWRGWA is leading edge triggered, not power-flow driven (similar to a counter input leg). The command to write the Gateway Address will be sent to the ECOM100 whenever the power flow into the IBox goes from OFF to ON.

If successful, turn on C100. If there is a failure, turn on C101. If it fails, you can look at V2000 for the specific error code.

To configure all of the ECOM100 TCP/IP parameters in one IBox, see the ECOM100 IP Setup (ECIPSUP) IBox.



ECOM100 Write IP Address (ECWRIP) (IB-723)

✗ 230

✗ 240

✓ 250-1

✓ 260

✓ 262

DS5	Used
HPP	N/A

ECOM100 Write IP Address will write the given IP Address to the ECOM100 module on a leading edge transition to the IBox. See also ECOM100 IP Setup (ECIPSUP) IBox 717 to setup ALL of the TCP/IP parameters in a single instruction—IP Address, Subnet Mask, and Gateway Address.

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Either the Success or Error bit parameter will turn on once the command is complete. If there is an error, the Error Code parameter will report an ECOM100 error code (less than 100), or a PLC logic error (greater than 1000).

The IP Address is stored in Flash-ROM in the ECOM100 and the execution of this IBox will disable the ECOM100 module for at least a half second until it writes the Flash-ROM. Therefore, it is HIGHLY RECOMMENDED that you only execute this IBox ONCE on the second scan. Since it requires a LEADING edge to execute, use a NORMALLY CLOSED SP0 (STR NOT First Scan) to drive the power flow to the IBox.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.

ECWRIP Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not successfully completed
- Error Code: specifies the location where the Error Code will be written
- IP Address: specifies the IP Address that will be written to the module

ECOM100 Write IP Address
IB-723

ECWRIP

ECOM100 # K0

Workspace V503

Success C100

Error C101

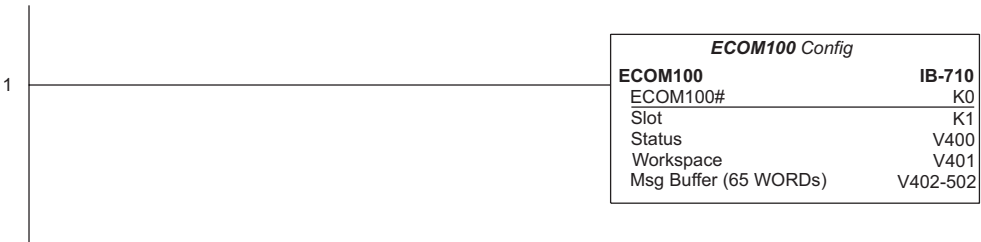
Error Code V2000

IP Address 192 . 168 . 012 . 100

Parameter		DL205 Range
ECOM100#	K	K0-255
Workspace	V	See DL205 V-memory map—Data Words
Success	X,Y,C,GX,GY,B	See DL205 V-memory map
Error	X,Y,C,GX,GY,B	See DL205 V-memory map
Error Code	V	See DL205 V-memory map—Data Words
IP Address		0.0.0.1. to 255.255.255.254

ECWRIP Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module. V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130-byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.



NOTE: An ECOM100 IBox instruction is used without a permissive contact. The top line will be identified in **BOLD italics**, and the instruction name and ID will be in **BOLD** characters.

Rung 2: On the second scan, assign the IP Address of the ECOM100 to 192.168.12.100

The ECWRIP is leading edge triggered, not power-flow driven (similar to a counter input leg). The command to write the IP Address will be sent to the ECOM100 whenever the power flow into the IBox goes from OFF to ON.

If successful, turn on C100. If there is a failure, turn on C101. If it fails, you can look at V2000 for the specific error code.

To configure all of the ECOM100 TCP/IP parameters in one IBox, see the ECOM100 IP Setup (ECIPSUP) IBox.



ECOM100 Write Module ID (ECWRMID) (IB-721)

✗ 230

✗ 240

✓ 250-1

✓ 260

✓ 262

DS5	Used
HPP	N/A

ECOM100 Write Module ID will write the given Module ID on a leading edge transition to the IBox

If the Module ID is set in the hardware using the dipswitches, this IBox will fail and return error code 1005 (decimal).

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Either the Success or Error bit parameter will turn on once the command is complete. If there is an error, the Error Code parameter will report an ECOM100 error code (less than 100), or a PLC logic error (greater than 1000).

The Module ID is stored in Flash-ROM in the ECOM100 and the execution of this IBox will disable the ECOM100 module for at least a half second until it writes the Flash-ROM. Therefore, it is **HIGHLY RECOMMENDED** that you only execute this IBox **ONCE** on the second scan. Since it requires a **LEADING** edge to execute, use a **NORMALLY CLOSED** SPO (STR NOT First Scan) to drive the power flow to the IBox.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.

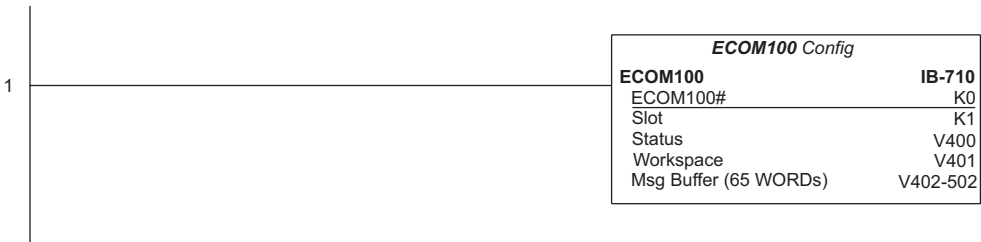
ECWRMID Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Workspace: specifies V-memory location that will be used by the instruction
- Success: specifies bit that will turn on once the request completed successfully
- Error: specifies a bit that will turn on the instruction is not completed successfully
- Error Code: specifies the location where the Error Code will be written
- Module ID: specifies the Module ID that will be written to the module

Parameter		DL205 Range
ECOM100#	K	K0-255
Workspace	V	See DL205 V-memory map–Data Words
Success	X,Y,C,GX,GY,B	See DL205 V-memory map
Error	X,Y,C,GX,GY,B	See DL205 V-memory map
Error Code	V	See DL205 V-memory map–Data Words
Module ID		K0-65535

ECWRMID Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module. V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130-byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.



NOTE: An ECOM100 IBox instruction is used without a permissive contact. The top line will be identified in **BOLD italics**, and the instruction name and ID will be in **BOLD** characters.

Rung 2: On the second scan, set the Module ID of the ECOM100. Typically this is done using NetEdit, but this IBox allows you to configure the module ID of the ECOM100 using your ladder program.

The ECWRMID is leading edge triggered, not power-flow driven (similar to a counter input leg). The command to write the module ID will be sent to the ECOM100 whenever the power flow into the IBox goes from OFF to ON.

If successful, turn on C100. If there is a failure, turn on C101. If it fails, you can look at V2000 for the specific error code.



ECOM100 Write Name (ECWRNAM) (IB-725)

✗ 230

✗ 240

✓ 250-1

✓ 260

✓ 262

DS5	Used
HPP	N/A

ECOM100 Write Name will write the given Name to the ECOM100 module on a leading edge transition to the IBox. If you use a dollar sign (\$) or double quote ("), use the PRINT/VPRINT escape sequence of TWO dollar signs (\$\$) for a single dollar sign or dollar sign-double quote (\$") for a double quote character.

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Either the Success or Error bit parameter will turn on once the command is complete. If there is an error, the Error Code parameter will report an ECOM100 error code (less than 100), or a PLC logic error (greater than 1000).

The Name is stored in Flash-ROM in the ECOM100 and the execution of this IBox will disable the ECOM100 module for at least a half second until it writes the Flash-ROM. Therefore, it is HIGHLY RECOMMENDED that you only execute this IBox ONCE on the second scan. Since it requires a LEADING edge to execute, use a NORMALLY CLOSED SP0 (STR NOT First Scan) to drive the power flow to the IBox.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.

ECWRNAM Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number.
- Workspace: specifies a V-memory location that will be used by the instruction.
- Success: specifies a bit that will turn on once the request is completed successfully.
- Error: specifies a bit that will turn on if the instruction is not completed successfully.
- Error Code: specifies the location where the Error Code will be written.
- Module Name: specifies the Name that will be written to the module.

ECOM100 Write Name (IB-725)

ECWRNAM

ECOM100 # K0

Workspace V503

Success C100

Error C101

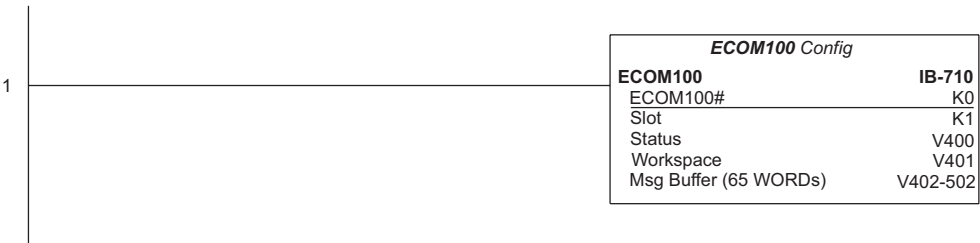
Error Code V2000

Module Name George

Parameter		DL205 Range
ECOM100#	K	K0-255
Workspace	V	See DL205 V-memory map—Data Words
Success	X,Y,C,GX,GY,B	See DL205 V-memory map
Error	X,Y,C,GX,GY,B	See DL205 V-memory map
Error Code	V	See DL205 V-memory map—Data Words
Module Name		Text

ECWRNAM Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module. V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130-byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.



NOTE: An ECOM100 IBox instruction is used without a permissive contact. The top line will be identified in ***BOLD italics***, and the instruction name and ID will be in **BOLD** characters.

Rung 2: On the second scan, set the Module Name of the ECOM100. Typically this is done using NetEdit, but this IBox allows you to configure the module name of the ECOM100 using your ladder program.

The ECWRNAM is leading edge triggered, not power-flow driven (similar to a counter input leg). The command to write the module name will be sent to the ECOM100 whenever the power flow into the IBox goes from OFF to ON.

If successful, turn on C100. If there is a failure, turn on C101. If it fails, you can look at V2000 for the specific error code.



ECOM100 Write Subnet Mask (ECWRSNM) (IB-733)

✗ 230

✗ 240

✓ 250-1

✓ 260

✓ 262

DS5	Used
HPP	N/A

ECOM100 Write Subnet Mask will write the given Subnet Mask to the ECOM100 module on a leading edge transition to the IBox. See also ECOM100 IP Setup (ECIPSUP) IBox 717 to set up ALL of the TCP/IP parameters in a single instruction—IP Address, Subnet Mask, and Gateway Address.

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Either the Success or Error bit parameter will turn on once the command is complete. If there is an error, the Error Code parameter will report an ECOM100 error code (less than 100), or a PLC logic error (greater than 1000).

The Subnet Mask is stored in Flash-ROM in the ECOM100 and the execution of this IBox will disable the ECOM100 module for at least a half second until it writes the Flash-ROM. Therefore, it is HIGHLY RECOMMENDED that you only execute this IBox ONCE on the second scan. Since it requires a LEADING edge to execute, use a NORMALLY CLOSED SP0 (STR NOT First Scan) to drive the power flow to the IBox.

In order for this ECOM100 IBox to function, you must turn ON dip switch 7 on the ECOM100 circuit board.

ECWRSNM Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number.
- Workspace: specifies a V-memory location that will be used by the instruction.
- Success: specifies a bit that will turn on once the request is completed successfully.
- Error: specifies a bit that will turn on if the instruction is not completed successfully.
- Error Code: specifies the location where the Error Code will be written.
- Subnet Mask: specifies the Subnet Mask that will be written to the module.

ECOM100 Write Subnet Mask
IB-733

ECWRSNM

ECOM100 # K0

Workspace V503

Success C100

Error C101

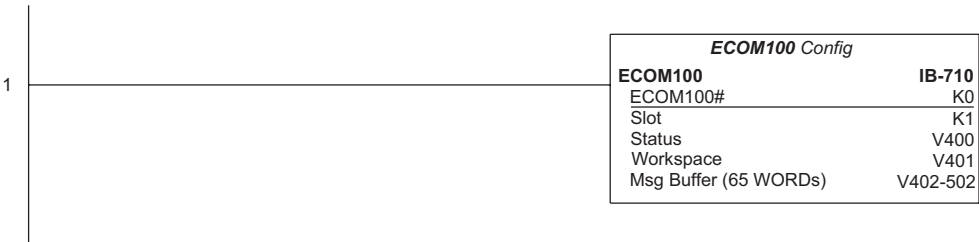
Error Code V2000

Subnet Mask 255 . 255 . 0 . 0

Parameter		DL205 Range
ECOM100#	K	K0-255
Workspace	V	See DL205 V-memory map—Data Words
Success	X,Y,C,GX,GY,B	See DL205 V-memory map
Error	X,Y,C,GX,GY,B	See DL205 V-memory map
Error Code	V	See DL205 V-memory map—Data Words
Subnet Mask		Masked IP Address

ECWRSNM Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module. V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130-byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.



NOTE: An ECOM100 IBox instruction is used without a permissive contact. The top line will be identified in **BOLD italics**, and the instruction name and ID will be in **BOLD** characters.

Rung 2: On the second scan, assign the Subnet Mask of the ECOM100 to 255.255.0.0

The ECWRSNM is leading edge triggered, not power-flow driven (similar to a counter input leg). The command to write the Subnet Mask will be sent to the ECOM100 whenever the power flow into the IBox goes from OFF to ON.

If successful, turn on C100. If there is a failure, turn on C101. If it fails, you can look at V2000 for the specific error code.

To configure all of the ECOM100 TCP/IP parameters in one IBox, see the ECOM100 IP Setup (ECIPSUP) IBox.



ECOM100 RX Network Read (ECRX) (IB-740)

-  230
-  240
-  250-1
-  260
-  262

ECOM100 RX Network Read performs the RX instruction with built-in interlocking with all other ECOM100 RX (ECRX) and ECOM100 WX (ECWX) IBoxes in your program to simplify communications networking. It will perform the RX on the specified ECOM100's network, which corresponds to a specific unique ECOM100 Configuration (ECOM100) IBox at the top of your program.

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

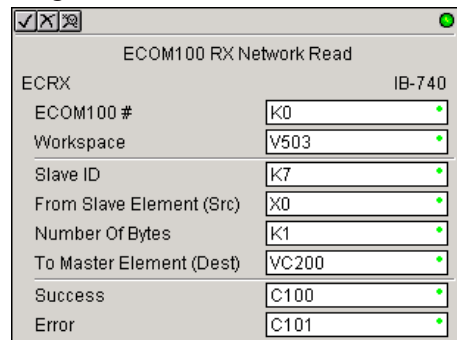
Whenever this IBox has power, it will read element data from the specified slave into the given destination V-memory buffer, giving other ECOM100 RX and ECOM100 WX IBoxes on that ECOM100# network a chance to execute.

For example, if you wish to read and write data continuously from five different slaves, you can have all of these ECRX and ECWX instructions in ONE RUNG driven by SP1 (Always On). They will execute round-robin style, automatically!

DS5	Used
HPP	N/A

ECRX Parameters

- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number
- Workspace: specifies a V-memory location that will be used by the instruction
- Slave ID: specifies the slave ECOM(100) PLC that will be targeted by the ECRX instruction
- From Slave Element (Src): specifies the slave address of the data to be read
- Number of Bytes: specifies the number of bytes to read from the slave ECOM(100) PLC
- To Master Element (Dest): specifies the location where the slave data will be placed in the master ECOM100 PLC
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not completed successfully



Parameter		DL205 Range
ECOM100#	K	K0-255
Workspace	V	See DL205 V-memory map-Data Words
Slave ID	K	K0-90
From Slave Element (Src)	X,Y,C,S,T,CT,GX,GY,V	See DL205 V-memory map
Number of Bytes	K	K1-128
To Master Element (Dest)	V	See DL205 V-memory map-Data Words
Success	X,Y,C,GX,GY,B	See DL205 V-memory map
Error	X,Y,C,GX,GY,B	See DL205 V-memory map

ECRX Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module # as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module. V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130-byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.



(Example continued on next page)



NOTE: An ECOM100 IBox instruction is used without a permissive contact. The top line will be identified in **BOLD** italics, and the instruction name and ID will be in **BOLD** characters.

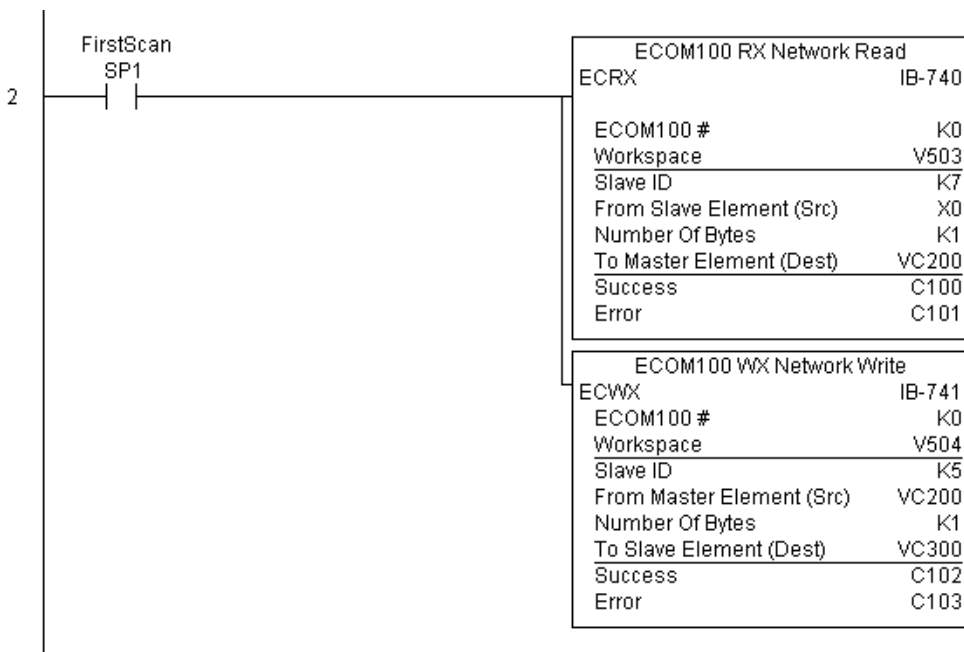
ECRX Example (cont'd)

Rung 2: Using ECOM100# K0, read X0-X7 from Slave K7 and write them to slave K5 as fast as possible. Store them in this local PLC in C200-C207, and write them to C300-C307 in slave K5.

Both the ECRX and ECWX work with the ECOM100 Config IBox to simplify all networking by handling all of the interlocks and proper resource sharing. They also provide very simplified error reporting. You no longer need to worry about any SP "busy bits" or "error bits," or what slot number a module is in, or have any counters or shift registers or any other interlocks for resource management.

In this example, SP1 (always ON) is driving both the ECRX and ECWX IBoxes in the same rung. On the scan that the Network Read completes, the Network Write will start that same scan. As soon as the Network Write completes, any pending operations below it in the program would get a turn. If there are no pending ECOM100 IBoxes below the ECWX, then the very next scan the ECRX would start its request again.

Using the ECRX and ECWX for all of your ECOM100 network reads and writes is the fastest the PLC can do networking. For local Serial Ports, DCM modules, or the original ECOM modules, use the NETCFG and NETRX/NETWX IBoxes.



ECOM100 WX Network Write(ECWX) (IB-741)

- ☒ 230
- ☒ 240
- ☒ 250-1
- ☒ 260
- ☒ 262

DS5	Used
HPP	N/A

ECOM100 WX Network Write performs the WX instruction with built-in interlocking with all other ECOM100 RX (ECRX) and ECOM100 WX (ECWX) IBoxes in your program to simplify communications networking. It will perform the WX on the specified ECOM100#'s network, which corresponds to a specific unique ECOM100 Configuration (ECOM100) IBox at the top of your program.

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Whenever this IBox has power, it will write data from the master's V-memory buffer to the specified slave starting with the given slave element, giving other ECOM100 RX and ECOM100 WX IBoxes on that ECOM100# network a chance to execute.

For example, if you wish to read and write data continuously from five different slaves, you can have all of these ECRX and ECWX instructions in ONE RUNG driven by SP1 (Always On). They will execute round-robin style, automatically!

ECWX Parameters

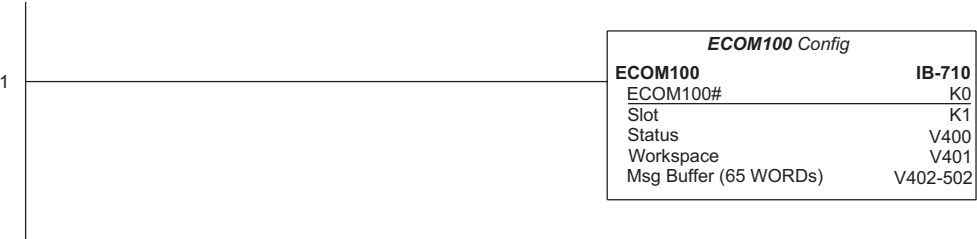
- ECOM100#: this is a logical number associated with this specific ECOM100 module in the specified slot. All other ECxxxx IBoxes that need to reference this ECOM100 module must reference this logical number.
- Workspace: specifies a V-memory location that will be used by the instruction.
- Slave ID: specifies the slave ECOM(100) PLC that will be targeted by the ECWX instruction.
- From Master Element (Src): specifies the location in the master ECOM100 PLC where the data will be sourced from.
- Number of Bytes: specifies the number of bytes to write to the slave ECOM(100) PLC.
- To Slave Element (Dest): specifies the slave address the data will be written to.
- Success: specifies a bit that will turn on once the request is completed successfully.
- Error: specifies a bit that will turn on if the instruction is not completed successfully.

ECOM100 WX Network Write IB-741	
ECWX	
ECOM100 #	K0
Workspace	V504
Slave ID	K5
From Master Element (Src)	VC200
Number Of Bytes	K1
To Slave Element (Dest)	VC300
Success	C102
Error	C103

Parameter		DL205 Range
ECOM100#	K	K0-255
Workspace	V	See DL205 V-memory map-Data Words
Slave ID	K	K0-90
From Master Element (Src)	V	See DL205 V-memory map-Data Words
Number of Bytes	K	K1-128
To Slave Element (Dest)	X,Y,C,S,T,CT,GX,GY,V	See DL205 V-memory map
Success	X,Y,C,GX,GY,B	See DL205 V-memory map
Error	X,Y,C,GX,GY,B	See DL205 V-memory map

ECWX Example

Rung 1: The ECOM100 Config IBox is responsible for coordination/interlocking of all ECOM100 type IBoxes for one specific ECOM100 module. Tag the ECOM100 in slot 1 as ECOM100# K0. All other ECxxxx IBoxes refer to this module number as K0. If you need to move the module in the base to a different slot, then you only need to change this one IBox. V400 is used as a global result status register for the other ECxxxx IBoxes using this specific ECOM100 module. V401 is used to coordinate/interlock the logic in all of the other ECxxxx IBoxes using this specific ECOM100 module. V402-V502 is a common 130-byte buffer available for use by the other ECxxxx IBoxes using this specific ECOM100 module.



NOTE: An ECOM100 IBox instruction is used without a permissive contact. The top line will be identified in BOLD italics, and the instruction name and ID will be in BOLD characters.

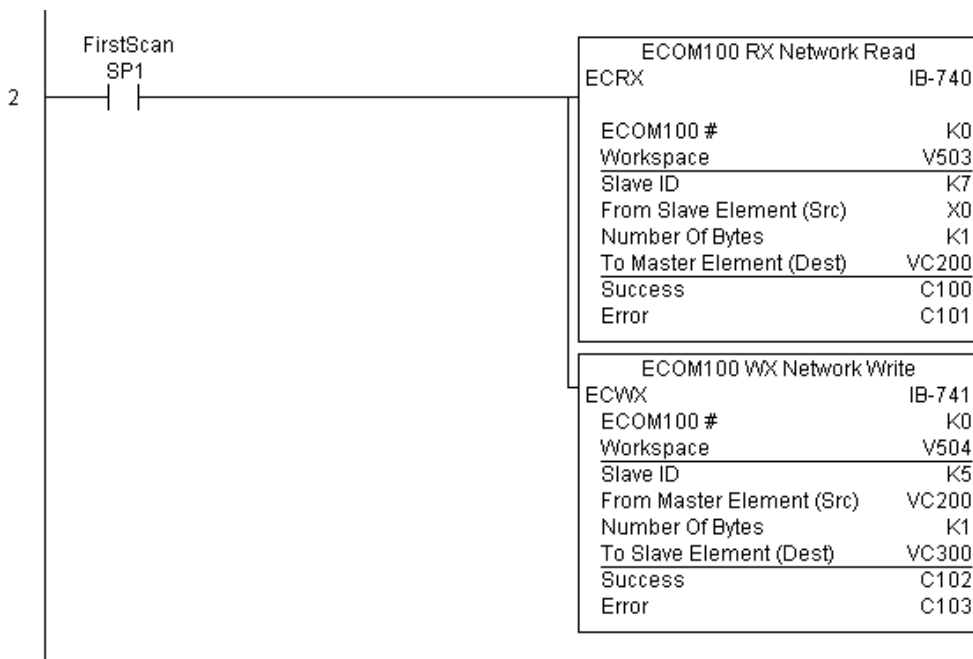
ECWX Example (cont'd)

Rung 2: Using ECOM100# K0, read X0-X7 from Slave K7 and write them to slave K5 as fast as possible. Store them in this local PLC in C200-C207, and write them to C300-C307 in slave K5.

Both the ECRX and ECWX work with the ECOM100 Config IBox to simplify all networking by handling all of the interlocks and proper resource sharing. They also provide very simplified error reporting. You no longer need to worry about any SP "busy bits" or "error bits," or what slot number a module is in, or have any counters or shift registers or any other interlocks for resource management.

In this example, SP1 (always ON) is driving both the ECRX and ECWX IBoxes in the same rung. On the scan that the Network Read completes, the Network Write will start that same scan. As soon as the Network Write completes, any pending operations below it in the program would get a turn. If there are no pending ECOM100 IBoxes below the ECWX, then the very next scan the ECRX would start its request again.

Using the ECRX and ECWX for all of your ECOM100 network reads and writes is the fastest the PLC can do networking. For local Serial Ports, DCM modules, or the original ECOM modules, use the NETCFG and NETRX/NETWX IBoxes.



NETCFG Network Configuration (NETCFG) (IB-700)

230

240

250-1

260

262

DS5	Used
HPP	N/A

Network Config defines all the common information necessary for performing RX/WX Networking using the NETRX and NETWX IBox instructions via a local CPU serial port, DCM or ECOM module.

You must have the Network Config instruction at the top of your ladder/stage program with any other configuration IBoxes.

If you use more than one local serial port, DCM or ECOM in your PLC for RX/WX Networking, you must have a different Network Config instruction and Network number for EACH RX/WX network in your system that utilizes any NETRX/NETWX IBox instructions.

The second parameter "CPU Port or Slot" is the same value as in the high byte of the first LD instruction if you were coding the RX or WX rung yourself. This value is CPU and port specific. Use KF1 for local CPU serial port 2. Use K3 if a DCM or ECOM is located in slot 3 of a local 205 base.

The Workspace parameter is an internal, private register used by the Network Config IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

NETCFG Parameters

- Network#: specifies a unique number for each ECOM(100) or DCM network to use
- CPU Port or Slot: specifies the CPU port number or slot number of DCM/ECOM(100) used
- Workspace: specifies a V-memory location that will be used by the instruction

Network Config	
NETCFG	IB-700
Network #	K0
CPU Port or Slot (ex. KF2 or K3)	KF1
Workspace	V400

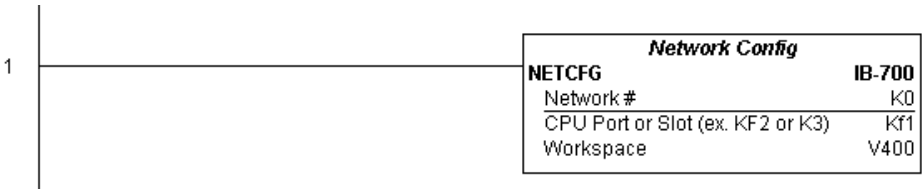
Parameter		DL205 Range
Network#	K	K0-255
CPU Port or Slot	K	K0-FF
Workspace	V	See DL205 V-memory map-Data Words

NETCFG Example

The Network Configuration IBox coordinates all of the interaction with other Network IBoxes (NETRX/NETWX). You must have a Network Configuration IBox for each serial port network, DCM module network, or original ECOM module network in your system. Configuration IBoxes must be at the top of your program and must execute every scan.

This IBox defines Network# K0 to be for the local CPU serial port #2 (KF1). For local CPU serial ports or DCM/ECOM modules, use the same value you would use in the most significant byte of the first LD instruction in a normal RX/WX rung to reference the port or module. Any NETRX or NETWX IBoxes that need to reference this specific network would enter K0 for their Network# parameter.

The Workspace register is used to maintain state information about the port or module, along with proper sharing and interlocking with the other NETRX and NETWX IBoxes in the program. This V-memory register must not be used anywhere else in the entire program.



NOTE: The Network Configuration IBox instruction is used without a permissive contact. The top line will be identified in ***BOLD italics***, and the instruction name and ID will be in ***BOLD characters***.

Network RX Read (NETRX) (IB-701)

230

240

250-1

260

262

Network RX Read performs the RX instruction with built-in interlocking with all other Network RX (NETRX) and Network WX (NETWX) IBoxes in your program to simplify communications networking. It will perform the RX on the specified Network number, which corresponds to a specific unique Network Configuration (NETCFG) at the top of your program.

The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Whenever this IBox has power, it will read element data from the specified slave into the given destination V-memory buffer, giving other Network RX and Network WX IBoxes on that Network number a chance to execute.

For example, if you wish to read and write data continuously from five different slaves, you can have all of these NETRX and NETWX instructions in ONE RUNG driven by SP1 (Always On). They will execute round-robin style, automatically!

NETRX Parameters

- Network#: specifies the (CPU ports, DCMs, ECOMs) Network # defined by the NETCFG instruction.
- Workspace: specifies a V-memory location that will be used by the instruction.
- Slave ID: specifies the slave PLC that will be targeted by the NETRX instruction.
- From Slave Element (Src): specifies the slave address of the data to be read.
- Number of Bytes: specifies the number of bytes to read from the slave device.
- To Master Element (Dest): specifies the location where the slave data will be placed in the master PLC.
- Success: specifies a bit that will turn on once the request is completed successfully.
- Error: specifies a bit that will turn on if the instruction is not completed successfully.

Network RX Read		IB-701
NETRX		
Network #	K0	
Workspace	V401	
Slave ID	K7	
From Slave Element (Src)	X0	
Number Of Bytes	K1	
To Master Element (Dest)	VC200	
Success	C100	
Error	C101	

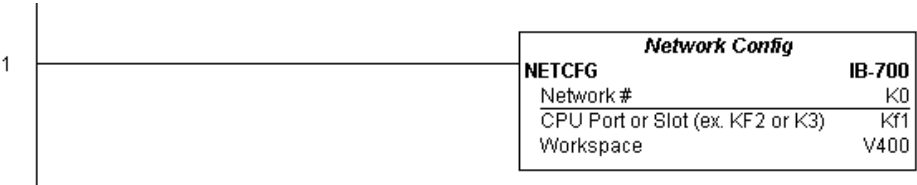
Parameter		DL205 Range
Network#	K	K0-255
Workspace	V	See DL205 V-memory map–Data Words
Slave ID	K	K0-90
From Slave Element (Src)	X,Y,C,S,T,CT,GX,GY,V	See DL205 V-memory map
Number of Bytes	K	K1-128
To Master Element (Dest)	V	See DL205 V-memory map–Data Words
Success	X,Y,C,GX,GY,B	See DL205 V-memory map
Error	X,Y,C,GX,GY,B	See DL205 V-memory map

NETRX Example

Rung 1: The Network Configuration IBox coordinates all of the interaction with other Network IBoxes (NETRX/NETWX). You must have a Network Configuration IBox for each serial port network, DCM module network, or original ECOM module network in your system. Configuration IBoxes must be at the top of your program and must execute every scan.

This IBox defines Network# K0 to be for the local CPU serial port #2 (KF1). For local CPU serial ports or DCM/ECOM modules, use the same value you would use in the most significant byte of the first LD instruction in a normal RX/WX rung to reference the port or module. Any NETRX or NETWX IBoxes that need to reference this specific network would enter K0 for their Network# parameter.

The Workspace register is used to maintain state information about the port or module, along with proper sharing and interlocking with the other NETRX and NETWX IBoxes in the program. This V-memory register must not be used anywhere else in the entire program.



(Example continued on next page)



NOTE: *The Network Configuration IBox instruction is used without a permissive contact. The top line will be identified in BOLD italics, and the instruction name and ID will be in BOLD characters.*

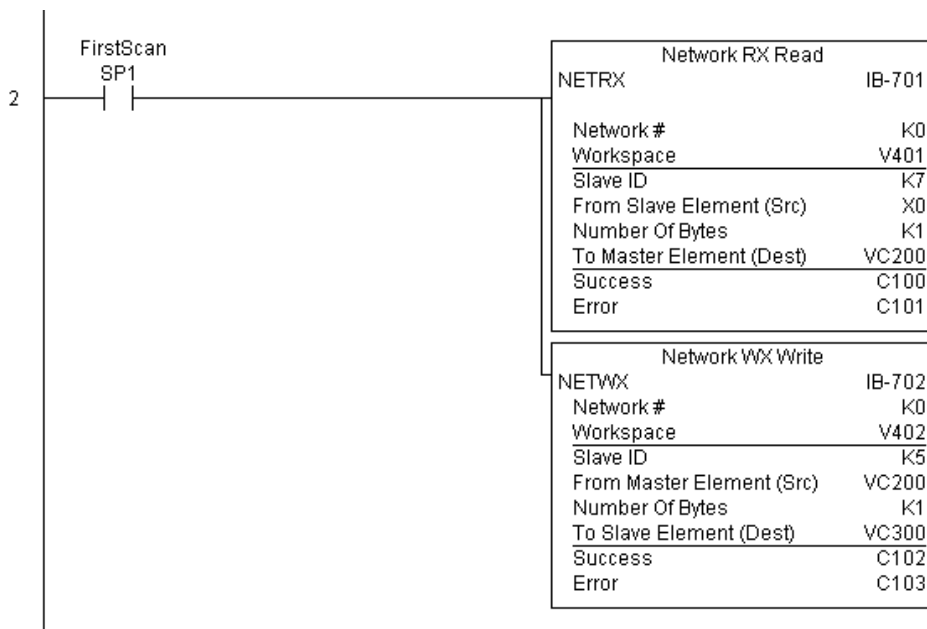
NETRX Example (cont'd)

Rung 2: Using Network# K0, read X0-X7 from Slave K7 and write them to slave K5 as fast as possible. Store them in this local PLC in C200-C207, and write them to C300-C307 in slave K5.

Both the NETRX and NETWX work with the Network Config IBox to simplify all networking by handling all of the interlocks and proper resource sharing. They also provide very simplified error reporting. You no longer need to worry about any SP “busy bits” or “error bits,” or what port number or slot number a module is in, or have any counters or shift registers or any other interlocks for resource management.

In this example, SP1 (always ON) is driving both the NETRX and NETWX IBoxes in the same rung. On the scan that the Network Read completes, the Network Write will start that same scan. As soon as the Network Write completes, any pending operations below it in the program would get a turn. If there are no pending NETRX or NETWX IBoxes below this IBox, then the very next scan the NETRX would start its request again.

Using the NETRX and NETWX for all of your serial port, DCM, or original ECOM network reads and writes is the fastest the PLC can do networking. For ECOM100 modules, use the ECOM100 and ECRX/ECWX IBoxes.



Network WX Write (NETWX) (IB-702)

-  230
-  240
-  250-1
-  260
-  262

Network WX Write performs the WX instruction with built-in interlocking with all other Network RX (NETRX) and Network WX (NETWX) IBoxes in your program to simplify communications networking. It will perform the WX on the specified Network number, which corresponds to a specific unique Network Configuration (NETCFG) at the top of your program.

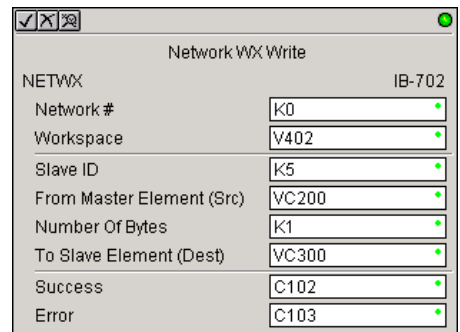
The Workspace parameter is an internal, private register used by this IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

Whenever this IBox has power, it will write data from the master's V-memory buffer to the specified slave starting with the given slave element, giving other Network RX and Network WX IBoxes on that Network number a chance to execute.

For example, if you wish to read and write data continuously from five different slaves, you can have all of these NETRX and NETWX instructions in ONE RUNG driven by SP1 (Always On). They will execute round-robin style, automatically!

NETWX Parameters

- Network#: specifies the (CPU ports, DCMs, ECOMs) Network # defined by the NETCFG instruction
- Workspace: specifies a V-memory location that will be used by the instruction
- Slave ID: specifies the slave PLC that will be targeted by the NETWX instruction
- From Master Element (Src): specifies the location in the master PLC where the data will be sourced
- Number of Bytes: specifies the number of bytes to write to the slave PLC
- To Slave Element (Dest): specifies the slave address the data will be written to
- Success: specifies a bit that will turn on once the request is completed successfully
- Error: specifies a bit that will turn on if the instruction is not completed successfully



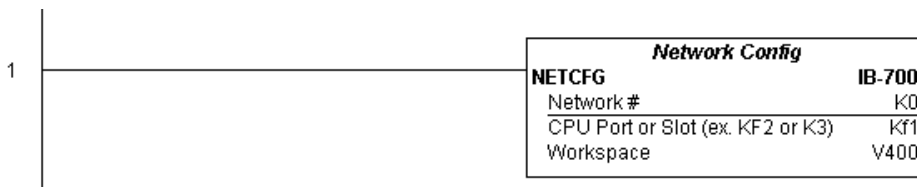
Parameter		DL205 Range
Network#	K	K0-255
Workspace	V	See DL205 V-memory map–Data Words
Slave ID	K,V	K0-90
From Slave Element (Src)	X,Y,C,S,T,CT,GX,GY,V	See DL205 V-memory map
Number of Bytes	K	K1-128
To Master Element (Dest)	V	See DL205 V-memory map–Data Words
Success	X,Y,C,GX,GY,B	See DL205 V-memory map
Error	X,Y,C,GX,GY,B	See DL205 V-memory map

NETWX Example

Rung 1: The Network Configuration IBox coordinates all of the interaction with other Network IBoxes (NETRX/NETWX). You must have a Network Configuration IBox for each serial port network, DCM module network, or original ECOM module network in your system. Configuration IBoxes must be at the top of your program and must execute every scan.

This IBox defines Network# K0 to be for the local CPU serial port #2 (KF1). For local CPU serial ports or DCM/ECOM modules, use the same value you would use in the most significant byte of the first LD instruction in a normal RX/WX rung to reference the port or module. Any NETRX or NETWX IBoxes that need to reference this specific network would enter K0 for their Network# parameter.

The Workspace register is used to maintain state information about the port or module, along with proper sharing and interlocking with the other NETRX and NETWX IBoxes in the program. This V-memory register must not be used anywhere else in the entire program.



(Example continued on next page)



NOTE: The Network Configuration IBox instruction is used without a permissive contact. The top line will be identified in ***BOLD italics***, and the instruction name and ID will be in ***BOLD*** characters.

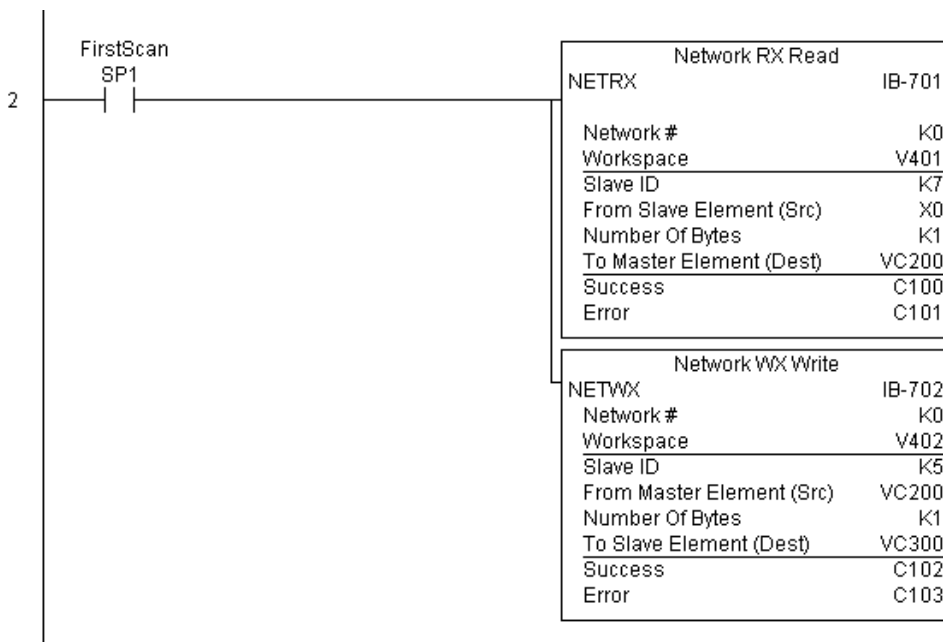
NETWX Example (cont'd)

Rung 2: Using Network# K0, read X0-X7 from Slave K7 and write them to slave K5 as fast as possible. Store them in this local PLC in C200-C207, and write them to C300-C307 in slave K5.

Both the NETRX and NETWX work with the Network Config IBox to simplify all networking by handling all of the interlocks and proper resource sharing. They also provide very simplified error reporting. You no longer need to worry about any SP "busy bits" or "error bits", or what port number or slot number a module is in, or have any counters or shift registers or any other interlocks for resource management.

In this example, SP1 (always ON) is driving both the NETRX and NETWX IBoxes in the same rung. On the scan that the Network Read completes, the Network Write will start that same scan. As soon as the Network Write completes, any pending operations below it in the program would get a turn. If there are no pending NETRX or NETWX IBoxes below this IBox, then the very next scan the NETRX would start its request again.

Using the NETRX and NETWX for all of your serial port, DCM, or original ECOM network reads and writes is the fastest the PLC can do networking. For ECOM100 modules, use the ECOM100 and ECRX/ECWX IBoxes.



CTRIO Configuration (CTRIO) (IB-1000)

CTRIO Config defines all the common information for one specific CTRIO module which is used by the other CTRIO IBox instructions (for example, CTRLDP—CTRIO Load Profile, CTREDRL—CTRIO Edit and Reload Preset Table, CTRRTLM—CTRIO Run to Limit Mode, ...).

The Input/Output parameters for this instruction can be copied directly from the CTRIO Workbench configuration for this CTRIO module. Since the behavior is slightly different when the CTRIO module is in an EBC Base via an ERM, you must specify whether the CTRIO module is in a local base or in an EBC base.

You must have the CTRIO Config IBox at the top of your ladder/stage program along with any other configuration IBoxes.

If you have more than one CTRIO in your PLC, you must have a different CTRIO Config IBox for EACH CTRIO module in your system that utilizes any CTRIO IBox instructions. Each CTRIO Config IBox must have a UNIQUE CTRIO# value. This is how the CTRIO IBoxes differentiate between the different CTRIO modules in your system.

The Workspace parameter is an internal, private register used by the CTRIO Config IBox and MUST BE UNIQUE in this one instruction and MUST NOT be used anywhere else in your program.

CTRIO Parameters

- CTRIO#: specifies a specific CTRIO module based on a user defined number
- Slot: (local base): specifies which PLC slot is occupied by the module (always K0 for EBC base)
- Workspace: specifies a V-memory location that will be used by the instruction
- CTRIO Location: specifies where the module is located (PLC local base or ERM to EBC base)
- Input (local base): This needs to be set to the same V-memory register as is specified in CTRIO Workbench as 'Starting V address for inputs' for this unique CTRIO.
- Output (local base): This needs to be set to the same V-memory register as is specified in CTRIO Workbench as 'Starting V address for outputs' for this unique CTRIO.
- Word Input (EBC base): The starting input V-memory address as defined by the I/O configuration in the ERM Workbench
- Bit Input (EBC base): The starting input Bit address as defined by the I/O configuration in the ERM Workbench
- Word Output (EBC base): The starting output V-memory address as defined by the I/O configuration in the ERM Workbench
- Bit Output (EBC base): The starting output Bit address as defined by the I/O configuration in the ERM Workbench.

The screenshot shows the 'CTRIO Config' IBox with the title 'CTRIO IB-1000'. The 'CTRIO Location' section has 'Local Base' selected. The parameters are: CTRIO # K1, Slot K2, Workspace V400, Input V2000, and Output V2030.

CTRIO in Local Base

The screenshot shows the 'CTRIO Config' IBox with the title 'CTRIO IB-1000'. The 'CTRIO Location' section has 'EBC (Connected via ERM)' selected. The parameters are: CTRIO # K0, Slot K0, Workspace V400, Word Input V2000, Bit Input B40416.0, Word Output V2100, and Bit Output B40515.0.

CTRIO in EBC Base

230

240

250-1

260

262

DS5	Used
HPP	N/A

Parameter		DL205 Range
CTRIO#	K	K0-255
Slot	K	K0-7
Workspace	V	See DL205 V-memory map-Data Words
Input (Word, Bit)	V,B	See DL205 V-memory map-Data Words
Output (Word, Bit)	V,B	See DL205 V-memory map-Data Words

CTRIO Example (local base)

Rung 1: This sets up the CTRIO module in slot 2 of the local base. Each CTRIO module in the system will need a separate CTRIO Config IBox before any CTRxxxx IBoxes can be used. The CTRIO has been configured to use V2000 through V2025 for its input data, and V2030 through V2061 for its output data.

1	CTRIO Config	
	CTRIO	IB-1000
	CTRIO #	K1
	Slot	Local K2
	Workspace	V400
	Input	V2000 - V2025
	Output	V2030 - V2061

CTRIO Example (EBC base)

Overview: ERM Workbench must first be used to assign memory addresses to the I/O modules in the EBC base. Once the CTRIO module memory addresses are established using ERM Workbench, they are used in CTRIO Workbench and in a CTRIO IBox instruction to configure and define a specific CTRIO module. For this example, the CTRIO module uses V2000–V2017 for its Word Input data and B40416.0–B40423.15 for its Bit Input data. The module uses V2100–V2123 for its Word Output data and B40515.0–B40522.15 for its Bit Output data. The starting addresses, V2000 and V40416 (for inputs) and V2100 and V40515 (for outputs) are entered into CTRIO Workbench I/O Map to configure this specific CTRIO module. These starting addresses are the memory locations used in the CTRIO IBox instruction as the Word Input, Bit Input, Word Output and Bit Output addresses as shown below. For more information on this topic, refer to the CTRIO User Manual “Program Control” chapter.

1	CTRIO Config	
	CTRIO	IB-1000
	CTRIO #	K0
	Slot	EBC K0
	Workspace	V400
	Word Input	V2000 - V2017
	Bit Input	B40416.0 - B40423.15
	Word Output	V2100 - V2123
	Bit Output	B40515.0 - B40522.15



NOTE: The CTRIO Configuration IBox instructions do not require a permissive contact. The top line will be identified in **BOLD italics**, and the instruction name and ID will be in **BOLD characters**.

CTRIO Add Entry to End of Preset Table (CTRADPT) (IB-1005)

☐ 230

☐ 240

☒ 250-1

☒ 260

☒ 262

CTRIO Add Entry to End of Preset Table, on a leading edge transition to this IBox, will append an entry to the end of a memory based Preset Table on a specific CTRIO Output resource. This IBox will take more than one PLC scan to execute. Either the Success or Error bit will turn on when the command is complete. If the Error Bit is on, you can use the CTRIO Read Error Code (CTRRDER) IBox to get extended error information.

Entry Type:

K0: Set

K1: Reset

K2: Pulse On (uses Pulse Time)

K3: Pulse Off (uses Pulse Time)

K4: Toggle

K5: Reset Count

Note that the Pulse Time parameter is ignored by some Entry Types.

The Workspace register is for internal use by this IBox instruction and MUST NOT be used anywhere else in your program.

CTRADPT Parameters

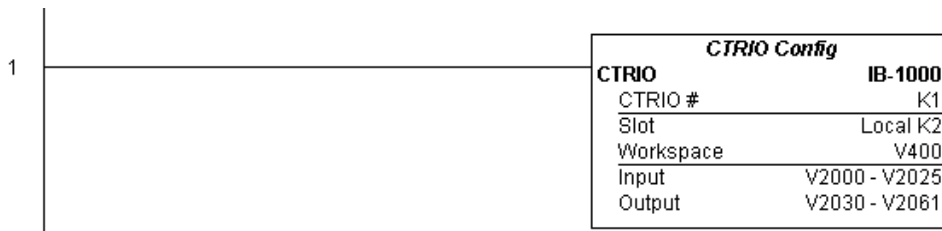
- CTRIO#: specifies a specific CTRIO module based on a user-defined number (see CTRIO Config)
- Output#: specifies a CTRIO output to be used by the instruction
- Entry Type: specifies the Entry Type to be added to the end of a Preset Table
- Pulse Time: specifies a pulse time in msec for the Pulse On and Pulse Off Entry Types
- Preset Count: specifies an initial count value to begin at after Reset
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the instruction has completed successfully
- Error: specifies a bit that will turn on if the instruction does not complete successfully

CTRIO Add Entry to End of Preset Table	
CTRADPT IB-1005	
CTRIO #	K1
Output #	K0
Entry Type	K1
Pulse Time	K0
Preset Count	K20
Workspace	V401
Success	C100
Error	C101

Parameter		DL205 Range
CTRIO#	K	K0-255
Output#	K	K0-3
Entry Type	V,K	K0-5; See DL205 V-memory map-Data Words
Pulse Time	V,K	K0-65535; See DL205 V-memory map-Data Words
Preset Count	V,K	K0-2147434528; See DL205 V-memory map
Workspace	V	See DL205 V-memory map-Data Words
Success	X,Y,C,GX,GY,B	See DL205 V-memory map
Error	X,Y,C,GX,GY,B	See DL205 V-memory map

CTRADPT Example

Rung 1: This sets up the CTrio module in slot 2 of the local base. Each CTrio module in the system will need a separate CTrio Config IBox before any CTRxxxx IBoxes can be used. The CTrio has been configured to use V2000 through V2025 for its input data, and V2030 through V2061 for its output data.

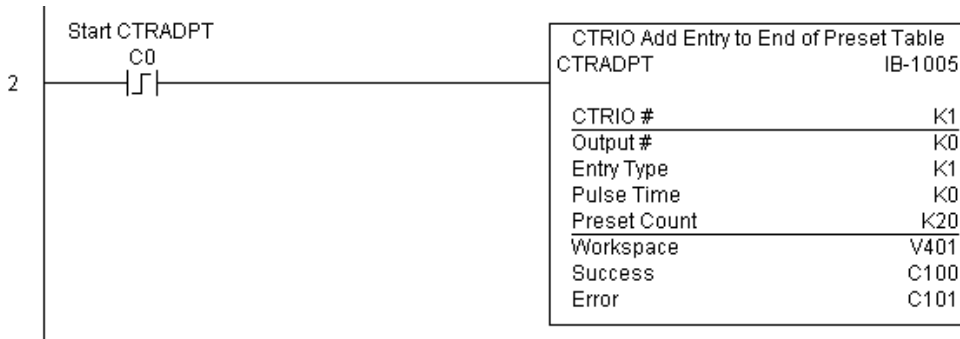


NOTE: The CTrio Configuration IBox instruction does not require a permissive contact. The top line will be identified in **BOLD italics**, and the instruction name and ID will be in **BOLD characters**.

Rung 2: This rung is a sample method for enabling the CTRADPT command. A C-bit is used to allow the programmer to control the command from Data View for testing purposes.

Turning on C0 will cause the CTRADPT instruction to add a new preset to the preset table for output #0 on the CTrio in slot 2. The new preset will be a command to RESET (entry type K1=reset), pulse time is left at zero as the reset type does not use this, and the count at which it will reset will be 20.

Operating procedure for this example code is to load the CTRADPT_ex1.cwb file to your CTrio, then enter the code shown here, change to RUN mode, enable output #0 by turning on C2 in Data View, turn encoder on CTrio to value above 10 and output #0 light will come on and stay on for all counts past 10. Now reset the counter with C1, enable C0 to execute CTRADPT command to add a reset for output #0 at a count of 20, turn on C2 to enable output #0, then turn encoder to value of 10+ (output #0 should turn on) and then continue on to count of 20+ (output #0 should turn off).



(Example continued on next page)

CTRADPT Example (cont'd)

Rung 3: This rung allows the programmer to reset the counter from the ladder logic.



Rung 4: This rung allows the operator to enable output #0 from the ladder code.



CTRIO Clear Preset Table (CTRCLRT) (IB-1007)

 230

 240

 250-1

 260

 262

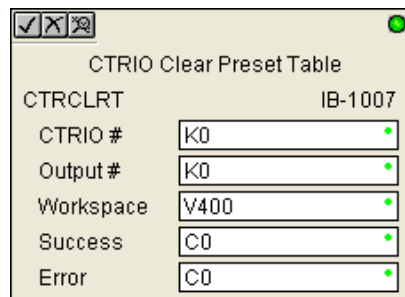
CTRIO Clear Preset Table will clear the RAM-based Preset Table on a leading edge transition to this IBox. This IBox will take more than one PLC scan to execute. Either the Success or Error bit will turn on when the command is complete. If the Error Bit is on, you can use the CTRIO Read Error Code (CTRRDER) IBox to get extended error information.

The Workspace register is for internal use by this IBox instruction and **MUST NOT** be used anywhere else in your program.

CTRCLRT Parameters

DS5	Used
HPP	N/A

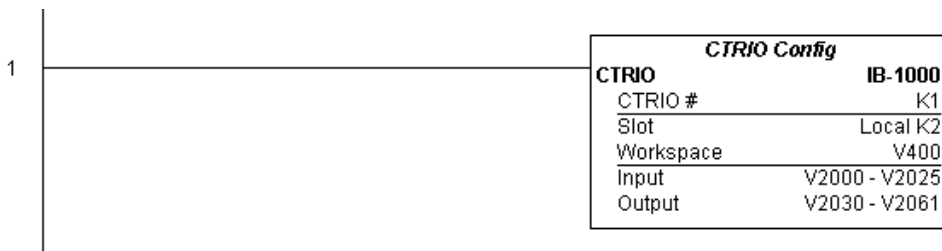
- CTRIO#: specifies a specific CTRIO module based on a user-defined number (see CTRIO Config)
- Output#: specifies a CTRIO output to be used by the instruction
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the instruction has completed successfully
- Error: specifies a bit that will turn on if the instruction does not complete successfully



Parameter		DL205 Range
CTRIO#	K	K0-255
Output#	K	K0-3
Workspace	V	See DL205 V-memory map—Data Words
Success	X,Y,C,GX,GY,B	See DL205 V-memory map
Error	X,Y,C,GX,GY,B	See DL205 V-memory map

CTRCLRT Example

Rung 1: This sets up the CTRIO module in slot 2 of the local base. Each CTRIO module in the system will need a separate CTRIO Config IBox before any CTRxxxx IBoxes can be used. The CTRIO has been configured to use V2000 through V2025 for its input data, and V2030 through V2061 for its output data.



NOTE: The CTRIO Configuration IBox instruction does not require a permissive contact. The top line will be identified in **BOLD italics**, and the instruction name and ID will be in **BOLD** characters.

Rung 2: This rung is a sample method for enabling the CTRCLRT command. A C-bit is used to allow the programmer to control the command from Data View for testing purposes.

Turning on C0 will cause the CTRCLRT instruction to clear the preset table for output #0 on the CTRIO in slot 2.

Operating procedure for this example code is to load the CTRCLRT_ex1.cwb file to your CTRIO, then enter the code shown here, change to RUN mode, enable output #0 by turning on C2 in Data View, turn encoder on CTRIO to value above 10 and output #0 light will come on and stay on until a count of 20 is reached, where it will turn off. Now reset the counter with C1, enable C0 to execute CTRCLRT command to clear the preset table, turn on C2 to enable output #0, then turn encoder to value of 10+ (output #0 should NOT turn on).



CTRCLRT Example (cont'd)

Rung 3: This rung allows the programmer to reset the counter from the ladder logic.



Rung 4: This rung allows the operator to enable output #0 from the ladder code.



CTRIO Edit Preset Table Entry (CTREDPT) (IB-1003)

 230

 240

 250-1

 260

 262

CTRIO Edit Preset Table Entry, on a leading edge transition to this IBox, will edit a single entry in a Preset Table on a specific CTRIO Output resource. This IBox is good if you are editing more than one entry in a file at a time. If you wish to do just one edit and then reload the table immediately, see the CTRIO Edit and Reload Preset Table Entry (CTREDRL) IBox.

This IBox will take more than one PLC scan to execute. Either the Success or Error bit will turn on when the command is complete. If the Error Bit is on, you can use the CTRIO Read Error Code (CTRRDER) IBox to get extended error information.

DS5	Used
HPP	N/A

Entry Type:

K0: Set

K1: Reset

K2: Pulse On (uses Pulse Time)

K3: Pulse Off (uses Pulse Time)

K4: Toggle

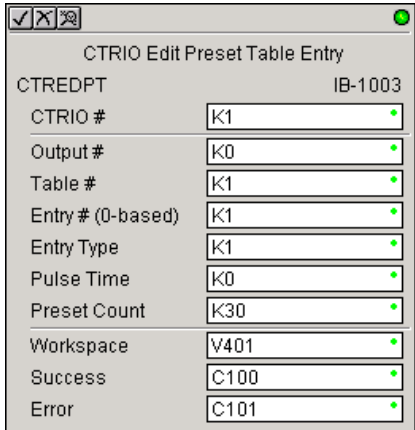
K5: Reset Count

Note that the Pulse Time parameter is ignored by some Entry Types.

The Workspace register is for internal use by this IBox instruction and MUST NOT be used anywhere else in your program.

CTREDPT Parameters

- CTRIO#: specifies a specific CTRIO module based on a user defined number (see CTRIO Config Ibox)
- Output#: specifies a CTRIO output to be used by the instruction
- Table#: specifies the Table number of which an Entry is to be edited
- Entry#: specifies the Entry location in the Preset Table to be edited
- Entry Type: specifies the Entry Type to add during the edit
- Pulse Time: specifies a pulse time in msecs for the Pulse On and Pulse Off Entry Types
- Preset Count: specifies an initial count value to begin at after Reset
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the instruction has successfully completed
- Error: specifies a bit that will turn on if the instruction does not complete successfully

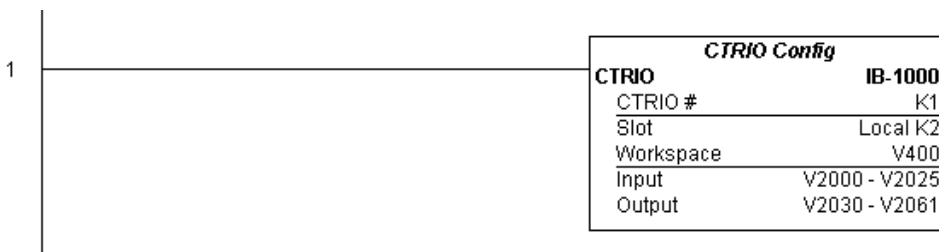


CTRIO Edit Preset Table Entry	
CTREDPT	IB-1003
CTRIO #	K1
Output #	K0
Table #	K1
Entry # (0-based)	K1
Entry Type	K1
Pulse Time	K0
Preset Count	K30
Workspace	V401
Success	C100
Error	C101

Parameter		DL205 Range
CTRIO#	K	K0-255
Output#	K	K0-3
Table#	V,K	K0-255; See DL205 V-memory map–Data Words
Entry#	V,K	K0-255; See DL205 V-memory map–Data Words
Entry Type	V,K	K0-5; See DL205 V-memory map–Data Words
Pulse Time	V,K	K0-65535; See DL205 V-memory map–Data Words
Preset Count	V,K	K0-2147434528; See DL205 V-memory map
Workspace	V	See DL205 V-memory map–Data Words
Success	X,Y,C,GX,GY,B	See DL205 V-memory map
Error	X,Y,C,GX,GY,B	See DL205 V-memory map

CTREDPT Example

Rung 1: This sets up the CTRIO module in slot 2 of the local base. Each CTRIO module in the system will need a separate CTRIO Config IBox before any CTRxxxx IBoxes can be used. The CTRIO has been configured to use V2000 through V2025 for its input data, and V2030 through V2061 for its output data.



(Example continued on next page)



NOTE: The CTRIO Configuration IBox instruction does not require a permissive contact. The top line will be identified in **BOLD italics**, and the instruction name and ID will be in **BOLD** characters.

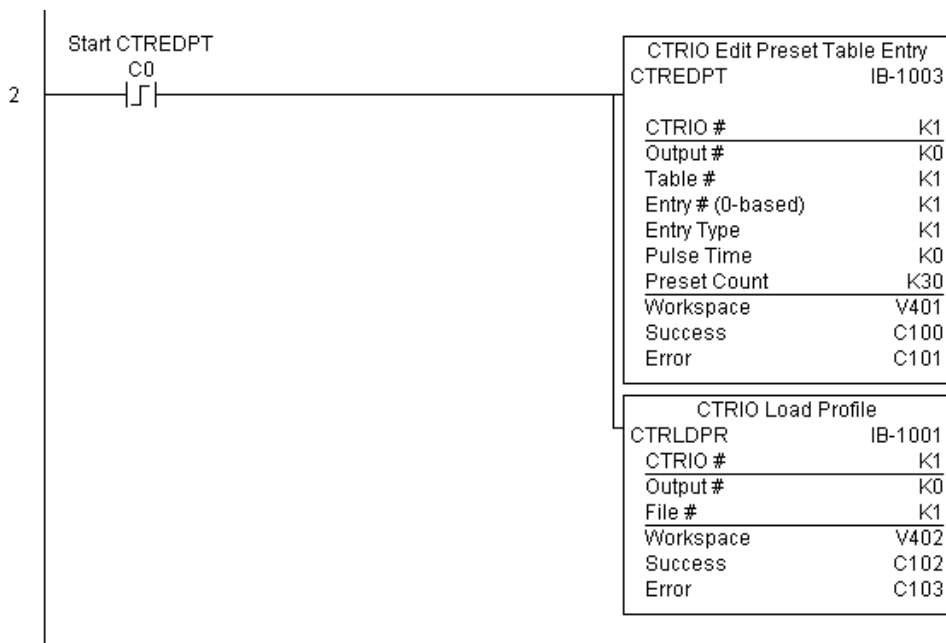
CTREDPT Example (cont'd)

Rung 2: This rung is a sample method for enabling the CTREDPT command. A C-bit is used to allow the programmer to control the command from Data View for testing purposes.

Turning on C0 will cause the CTREDPT instruction to change the second preset from a reset at a count of 20 to a reset at a count of 30 for output #0 on the CTRIO in slot 2.

Operating procedure for this example code is to load the CTREDPT_ex1.cwb file to your CTRIO, then enter the code shown here, change to RUN mode, enable output #0 by turning on C2 in Data View, turn encoder on CTRIO to value above 10 and output #0 light will come on and stay on until a count of 20 is reached, where it will turn off. Now reset the counter with C1, enable C0 to execute CTREDPT command to change the second preset, turn on C2 to enable output #0, then turn encoder to value of 10+ (output #0 should turn on) and then continue past a count of 30 (output #0 should turn off).

Note that we must also reload the profile after changing the preset(s); this is why the CTRLDPR command follows the CTREDPT command in this example.



CTREDPT Example (cont'd)

Rung 3: This rung allows the programmer to reset the counter from the ladder logic.



Rung 4: This rung allows the operator to enable output #0 from the ladder code.



CTRIO Edit Preset Table Entry and Reload (CTREDRL) (IB-1002)

CTRIO Edit Preset Table Entry and Reload, on a leading edge transition to this IBox, will perform this dual operation to a CTRIO Output resource in one CTRIO command. This IBox will take more than one PLC scan to execute. Either the Success or Error bit will turn on when the command is complete. If the Error Bit is on, you can use the CTRIO Read Error Code (CTRRDER) IBox to get extended error information.

☒ 230

☒ 240

☒ 250-1

☒ 260

☒ 262

Entry Type:

K0: Set

K1: Reset

K2: Pulse On (uses Pulse Time)

K3: Pulse Off (uses Pulse Time)

K4: Toggle

K5: Reset Count

Note that the Pulse Time parameter is ignored by some Entry Types.

The Workspace register is for internal use by this IBox instruction and MUST NOT be used anywhere else in your program.

CTREDRL Parameters

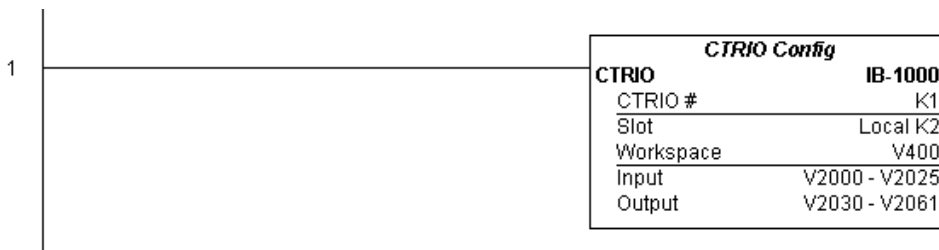
- CTRIO#: specifies a specific CTRIO module based on a user defined number (see CTRIO Config Ibox)
- Output#: specifies a CTRIO output to be used by the instruction
- Table#: specifies the Table number of which an Entry is to be edited
- Entry#: specifies the Entry location in the Preset Table to be edited
- Entry Type: specifies the Entry Type to add during the edit
- Pulse Time: specifies a pulse time in msec for the Pulse On and Pulse Off Entry Types
- Preset Count: specifies an initial count value to begin at after Reset
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the instruction has completed successfully
- Error: specifies a bit that will turn on if the instruction does not complete successfully

CTRIO Edit Preset Table Entry and Reload	
CTREDRL IB-1002	
CTRIO #	K1
Output #	K0
Table #	K1
Entry # (0-based)	K1
Entry Type	K1
Pulse Time	K0
Preset Count	K30
Workspace	V401
Success	C100
Error	C101

Parameter		DL205 Range
CTRIO#	K	K0-255
Output#	K	K0-3
Table#	V,K	K0-255; See DL205 V-memory map–Data Words
Entry#	V,K	K0-255; See DL205 V-memory map–Data Words
Entry Type	V,K	K0-5; See DL205 V-memory map–Data Words
Pulse Time	V,K	K0-65535; See DL205 V-memory map–Data Words
Preset Count	V,K	K0-2147434528; See DL205 V-memory map
Workspace	V	See DL205 V-memory map–Data Words
Success	X,Y,C,GX,GY,B	See DL205 V-memory map
Error	X,Y,C,GX,GY,B	See DL205 V-memory map

CTREDRL Example

Rung 1: This sets up the CTRIO module in slot 2 of the local base. Each CTRIO module in the system will need a separate CTRIO Config IBox before any CTRxxxx IBoxes can be used. The CTRIO has been configured to use V2000 through V2025 for its input data, and V2030 through V2061 for its output data.



(Example continued on next page)



NOTE: The CTRIO Configuration IBox instruction does not require a permissive contact. The top line will be identified in **BOLD italics**, and the instruction name and ID will be in **BOLD** characters.

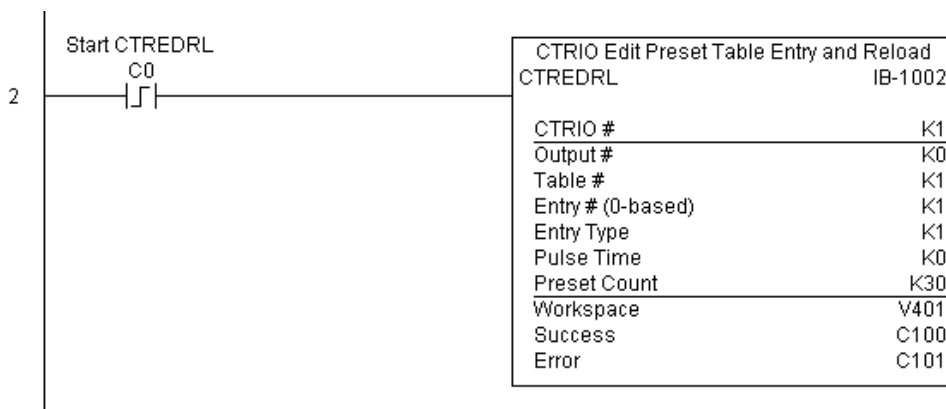
CTREDRL Example (cont'd)

Rung 2: This rung is a sample method for enabling the CTREDRL command. A C-bit is used to allow the programmer to control the command from Data View for testing purposes.

Turning on C0 will cause the CTREDRL instruction to change the second preset in file 1 from a reset value of 20 to a reset value of 30.

Operating procedure for this example code is to load the CTREDRL_ex1.cwb file to your CTRIO, then enter the code shown here, change to RUN mode, enable output #0 by turning on C2 in Data View, turn encoder on CTRIO to value above 10 and output #0 light will come on, continue to a count above 20 and the output #0 light will turn off. Now reset the counter with C1, enable C0 to execute CTREDRL command to change the second preset count value to 30, then turn encoder to value of 10+ (output #0 should turn on) and continue on to a value of 30+ and the output #0 light will turn off.

Note that it is not necessary to reload this file separately, however, the command can only change one value at a time.



(Example continued on next page)

CTREDRL Example (cont'd)

Rung 3: This rung allows the programmer to reset the counter from the ladder logic.



Rung 4: This rung allows the operator to enable output #0 from the ladder code.



CTRIO Initialize Preset Table (CTRINPT) (IB-1004)

CTRIO Initialize Preset Table, on a leading edge transition to this IBox, will create a single entry Preset Table in memory but not as a file, on a specific CTRIO Output resource. This IBox will take more than one PLC scan to execute. Either the Success or Error bit will turn on when the command is complete. If the Error Bit is on, you can use the CTRIO Read Error Code (CTRRDER) IBox to get extended error information.

 230

 240

 250-1

 260

 262

Entry Type:

K0: Set

K1: Reset

K2: Pulse On (uses Pulse Time)

K3: Pulse Off (uses Pulse Time)

K4: Toggle

K5: Reset Count

Note that the Pulse Time parameter is ignored by some Entry Types.

The Workspace register is for internal use by this IBox instruction and MUST NOT be used anywhere else in your program.

CTRINPT Parameters

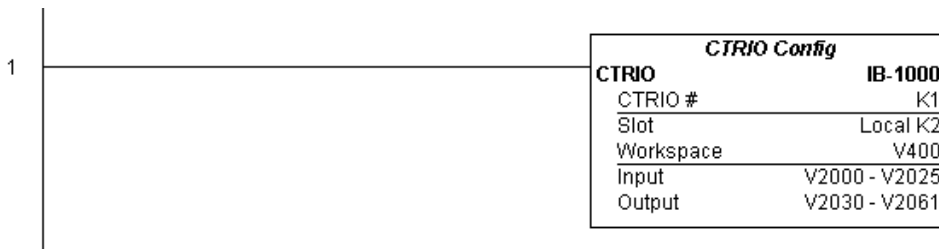
- CTRIO#: specifies a specific CTRIO module based on a user defined number (see CTRIO Config Ibox)
- Output#: specifies a CTRIO output to be used by the instruction
- Entry Type: specifies the Entry Type to add during the edit
- Pulse Time: specifies a pulse time in msec for the Pulse On and Pulse Off Entry Types
- Preset Count: specifies an initial count value to begin at after Reset
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the instruction has completed successfully
- Error: specifies a bit that will turn on if the instruction does not complete successfully

CTRIO Initialize Preset Table	
CTRINPT	IB-1004
CTRIO #	K1
Output #	K0
Entry Type	K0
Pulse Time	K0
Preset Count	K15
Workspace	V401
Success	C100
Error	C101

Parameter		DL205 Range
CTRIO#	K	K0-255
Output#	K	K0-3
Entry Type	V,K	K0-5; See DL205 V-memory map-Data Words
Pulse Time	V,K	K0-65535; See DL205 V-memory map-Data Words
Preset Count	V,K	K0-2147434528; See DL205 V-memory map
Workspace	V	See DL205 V-memory map-Data Words
Success	X,Y,C,GX,GY,B	See DL205 V-memory map
Error	X,Y,C,GX,GY,B	See DL205 V-memory map

CTRINPT Example

Rung 1: This sets up the CTRIO module in slot 2 of the local base. Each CTRIO module in the system will need a separate CTRIO Config IBox before any CTRxxxx IBoxes can be used. The CTRIO has been configured to use V2000 through V2025 for its input data, and V2030 through V2061 for its output data.



(Example continued on next page)



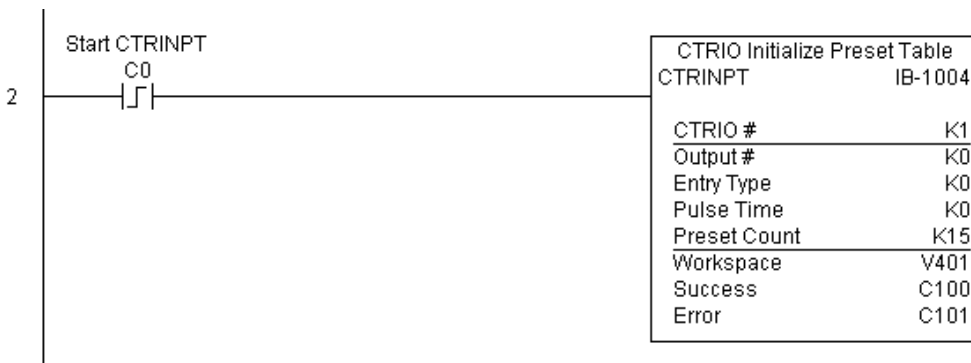
NOTE: The CTRIO Configuration IBox instruction does not require a permissive contact. The top line will be identified in **BOLD italics**, and the instruction name and ID will be in **BOLD** characters.

CTRINPT Example (cont'd)

Rung 2: This rung is a sample method for enabling the CTRINPT command. A C-bit is used to allow the programmer to control the command from Data View for testing purposes.

Turning on C0 will cause the CTRINPT instruction to create a single entry preset table, but not as a file, and use it for the output #0. In this case the single preset will be set at a count of 15 for output #0.

Operating procedure for this example code is to load the CTRINPT_ex1.cwb file to your CTRIO, then enter the code shown here, change to RUN mode, enable output #0 by turning on C2 in Data View, turn encoder on CTRIO to value above 15 and output #0 light will not come on. Now reset the counter with C1, enable C0 to execute CTRINPT command to create a single preset table with a preset to set output#0 at a count of 15, then turn encoder to value of 15+ (output #0 should turn on).



CTRINPT Example (cont'd)

Rung 3: This rung allows the programmer to reset the counter from the ladder logic.



Rung 4: This rung allows the operator to enable output #0 from the ladder code.



CTRIO Initialize Preset Table on Reset (CTRINTR) (IB-1010)

CTRIO Initialize Preset Table on Reset, on a leading edge transition to this IBox, will create a single entry Preset Table in memory but not as a file, on a specific CTRIO Output resource. This IBox will take more than 1 PLC scan to execute. Either the Success or Error bit will turn on when the command is complete. If the Error Bit is on, you can use the CTRIO Read Error Code (CTRRDER) IBox to get extended error information.

✗ 230

✗ 240

✓ 250-1

✓ 260

✓ 262

Entry Type:

K0: Set

K1: Reset

K2: Pulse On (uses Pulse Time)

K3: Pulse Off (uses Pulse Time)

K4: Toggle

K5: Reset Count

DS5	Used
HPP	N/A

Note that the Pulse Time parameter is ignored by some Entry Types.

The Workspace register is for internal use by this IBox instruction and MUST NOT be used anywhere else in your program.

CTRINTR Parameters

- CTRIO#: specifies a specific CTRIO module based on a user defined number (see CTRIO Config Ibox)
- Output#: specifies a CTRIO output to be used by the instruction
- Entry Type: specifies the Entry Type to add during the edit
- Pulse Time: specifies a pulse time in msec for the Pulse On and Pulse Off Entry Types
- Preset Count: specifies an initial count value to begin at after Reset
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the instruction has completed successfully
- Error: specifies a bit that will turn on if the instruction does not complete successfully

CTRIO Initialize Preset Table on Reset

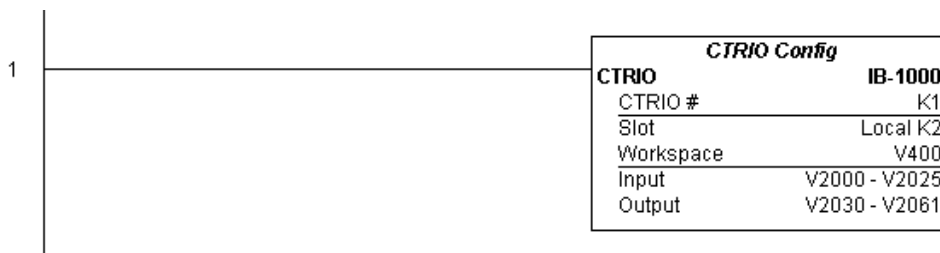
CTRINTR IB-1010

CTRIO #	K1
Output #	K0
Entry Type	K0
Pulse Time	K0
Preset Count	K25
Workspace	V401
Success	C100
Error	C101

Parameter		DL205 Range
CTRIO#	K	K0-255
Output#	K	K0-3
Entry Type	V,K	K0-5; See DL205 V-memory map–Data Words
Pulse Time	V,K	K0-65535; See DL205 V-memory map–Data Words
Preset Count	V,K	K0-2147434528; See DL205 V-memory map
Workspace	V	See DL205 V-memory map–Data Words
Success	X,Y,C,GX,GY,B	See DL205 V-memory map
Error	X,Y,C,GX,GY,B	See DL205 V-memory map

CTRINTR Example

Rung 1: This sets up the CTRIO module in slot 2 of the local base. Each CTRIO module in the system will need a separate CTRIO Config IBox before any CTRxxxx IBoxes can be used. The CTRIO has been configured to use V2000 through V2025 for its input data, and V2030 through V2061 for its output data.



(Example continued on next page)



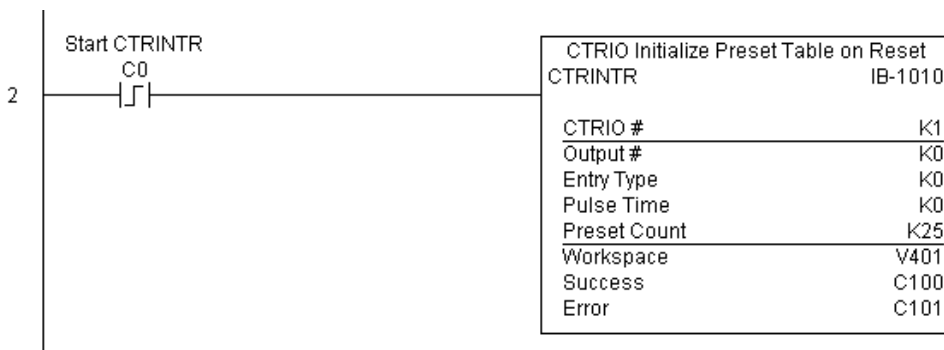
NOTE: The CTRIO Configuration IBox instruction does not require a permissive contact. The top line will be identified in **BOLD italics**, and the instruction name and ID will be in **BOLD** characters.

CTRINTR Example (cont'd)

Rung 2: This rung is a sample method for enabling the CTRINTR command. A C-bit is used to allow the programmer to control the command from Data View for testing purposes.

Turning on C0 will cause the CTRINTR instruction to create a single entry preset table, but not as a file, and use it for output #0, the new preset will be loaded when the current count is reset. In this case the single preset will be a set at a count of 25 for output #0.

Operating procedure for this example code is to load the CTRINTR_ex1.cwb file to your CTRIO, then enter the code shown here, change to RUN mode, enable output #0 by turning on C2 in Data View, turn encoder on CTRIO to value above 10 and output #0 light will come on. Now turn on C0 to execute the CTRINTR command, reset the counter with C1, then turn encoder to value of 25+ (output #0 should turn on).



CTRINTR Example (cont'd)

Rung 3: This rung allows the programmer to reset the counter from the ladder logic.



Rung 4: This rung allows the operator to enable output #0 from the ladder code.



CTRIO Load Profile (CTRLDPR) (IB-1001)

 230

 240

 250-1

 260

 262

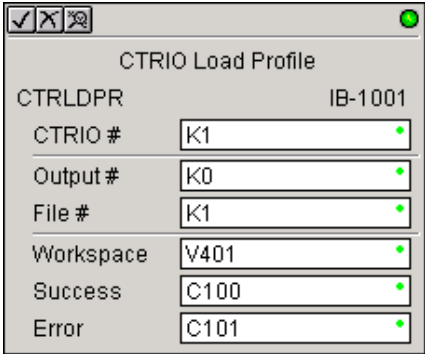
CTRIO Load Profile loads a CTRIO Profile File to a CTRIO Output resource on a leading edge transition to this IBox. This IBox will take more than one PLC scan to execute. Either the Success or Error bit will turn on when the command is complete. If the Error Bit is on, you can use the CTRIO Read Error Code (CTRRDER) IBox to get extended error information.

The Workspace register is for internal use by this IBox instruction and MUST NOT be used anywhere else in your program.

CTRLDPR Parameters

DS5	Used
HPP	N/A

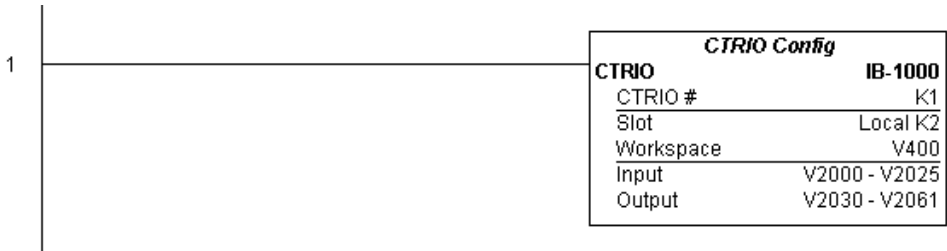
- CTRIO#: specifies a specific CTRIO module based on a user defined number (see CTRIO Config)
- Output#: specifies a CTRIO output to be used by the instruction
- File#: specifies a CTRIO profile File number to be loaded
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the instruction has completed successfully
- Error: specifies a bit that will turn on if the instruction does not complete successfully



Parameter		DL205 Range
CTRIO#	K	K0-255
Output#	K	K0-3
File#	V,K	K0-255; See DL205 V-memory map-Data Words
Workspace	V	See DL205 V-memory map-Data Words
Success	X,Y,C,GX,GY,B	See DL205 V-memory map
Error	X,Y,C,GX,GY,B	See DL205 V-memory map

CTRLDPR Example

Rung 1: This sets up the CTRIO module in slot 2 of the local base. Each CTRIO module in the system will need a separate CTRIO Config IBox before any CTRxxxx IBoxes can be used. The CTRIO has been configured to use V2000 through V2025 for its input data, and V2030 through V2061 for its output data.



NOTE: The CTRIO Configuration IBox instruction does not require a permissive contact. The top line will be identified in **BOLD italics**, and the instruction name and ID will be in **BOLD** characters.

Rung 2: This CTRIO Load Profile IBox will load File #1 into the working memory of Output 0 in CTRIO #1. This example program requires that you load CTRLDPR_IBox.cwb into your Hx-CTRIO(2) module.



(Example continued on next page)

CTRLDPR Example (cont'd)

Rung 3: If the file is loaded successfully, set Profile_Loaded.



CTRIO Read Error (CTRRDER) (IB-1014)

CTRIO Read Error Code, on a leading edge transition to this IBox, will read the decimal error code value (listed below) from the CTRIO module and place it in the specified Error Code register. This instruction is not supported when the CTRIO is used in an ERM/EBC configuration.

230

240

250-1

260

262

Since the Error Code in the CTRIO is only maintained until another CTRIO command is given, you must use this instruction immediately after the CTRIO IBox that reports an error via its Error bit parameter.

DS5	Used
HPP	N/A

The Workspace register is for internal use by this IBox instruction and MUST NOT be used anywhere else in your program.

Error Codes:

0: No Error

100: Specified command code is unknown or unsupported

101: File number not found in the file system

102: File type is incorrect for specified output function

103: Profile type is unknown

104: Specified input is not configured as a limit on this output

105: Specified limit input edge is out of range

106: Specified input function is unconfigured or invalid

107: Specified input function number is out of range

108: Specified preset function is invalid

109: Preset table is full

110: Specified Table entry is out of range

111: Specified register number is out of range

112: Specified register is an unconfigured input or output

2001: Error reading Error Code—cannot access CTRIO via ERM

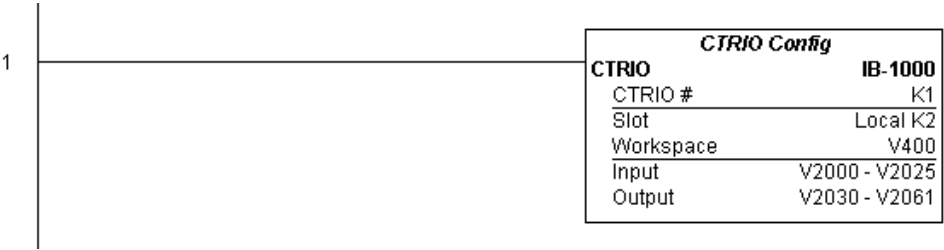
CTRRDER Parameters

- CTRIO#: specifies a specific CTRIO module based on a user-defined number (see CTRIO Config)
- Workspace: specifies a V-memory location that will be used by the instruction
- Error Code: specifies the location where the Error Code will be written

Parameter		DL205 Range
CTRIO#	K	K0-255
Workspace	V	See DL205 V-memory map—Data Words
Error Code	V	See DL205 V-memory map—Data Words

CTRRDER Example

Rung 1: This sets up the CTRIO module in slot 2 of the local base. Each CTRIO module in the system will need a separate CTRIO Config IBox before any CTRxxxx IBoxes can be used. The CTRIO has been configured to use V2000 through V2025 for its input data, and V2030 through V2061 for its output data.



NOTE: The CTRIO Configuration IBox instruction does not require a permissive contact. The top line will be identified in **BOLD italics**, and the instruction name and ID will be in **BOLD** characters.

Rung 2: This CTRIO Read Error Code IBox will read the Extended Error information from CTRIO #1. This example program requires that you load CTRRDER_IBox.cwb into your Hx-CTRIO(2) module.



CTRIO Run to Limit Mode (CTRRTLM) (IB-1011)

CTRIO Run To Limit Mode, on a leading edge transition to this IBox, loads the Run to Limit command and given parameters on a specific Output resource. The CTRIO's Input(s) must be configured as Limit(s) for this function to work.

 230

 240

 250-1

 260

 262

Valid Hexadecimal Limit Values:

K00—Rising Edge of Ch1/C

K10—Falling Edge of Ch1/C

K20—Both Edges of Ch1/C

K01—Rising Edge of Ch1/D

K11—Falling Edge of Ch1/D

K21—Both Edges of Ch1/D

K02—Rising Edge of Ch2/C

K12—Falling Edge of Ch2/C

K22—Both Edges of Ch2/C

K03—Rising Edge of Ch2/D

K13—Falling Edge of Ch2/D

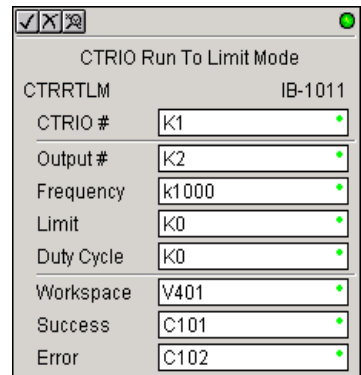
K23—Both Edges of Ch2/D

This IBox will take more than one PLC scan to execute. Either the Success or Error bit will turn on when the command is complete. If the Error Bit is on, you can use the CTRIO Read Error Code (CTRRDER) IBox to get extended error information.

The Workspace register is for internal use by this IBox instruction and MUST NOT be used anywhere else in your program.

CTRRTLM Parameters

- CTRIO#: specifies a specific CTRIO module based on a user-defined number (see CTRIO Config Ibox).
- Output#: specifies a CTRIO output to be used by the instruction.
- Frequency: specifies the output pulse rate (H2-CTRIO: 20Hz–25KHz / H2-CTRIO2: 20Hz–250 KHz).
- Limit: the CTRIO's Input(s) must be configured as Limit(s) for this function to operate.
- Duty Cycle: specifies the % of on time versus off time. This is a hex number. Default of 0 is 50%, also entering 50 will yield 50%. 50% duty cycle is defined as on half the time and off half the time.
- Workspace: specifies a V-memory location that will be used by the instruction.
- Success: specifies a bit that will turn on once the instruction has completed successfully.
- Error: specifies a bit that will turn on if the instruction does not complete successfully.

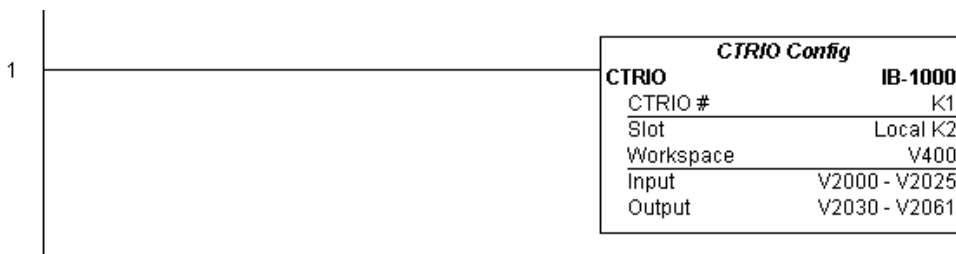


CTRIO Run To Limit Mode	
CTRRTLM	IB-1011
CTRIO #	K1
Output #	K2
Frequency	k1000
Limit	K0
Duty Cycle	K0
Workspace	V401
Success	C101
Error	C102

Parameter		DL205 Range
CTRIO#	K	K0-255
Output#	K	K0-3
Frequency	V,K	K20-20000; See DL205 V-memory map-Data Words
Limit	V,K	K0-FF; See DL205 V-memory map-Data Words
Duty Cycle	V,K	K0-99; See DL205 V-memory map-Data Words
Workspace	V	See DL205 V-memory map-Data Words
Success	X,Y,C,GX,GY,B	See DL205 V-memory map
Error	X,Y,C,GX,GY,B	See DL205 V-memory map

CTRRTLM Example

Rung 1: This sets up the CTRIO module in slot 2 of the local base. Each CTRIO module in the system will need a separate CTRIO Config IBox before any CTRxxx IBoxes can be used. The CTRIO has been configured to use V2000 through V2025 for its input data, and V2030 through V2061 for its output data.



NOTE: The CTRIO Configuration IBox instruction does not require a permissive contact. The top line will be identified in **BOLD** italics, and the instruction name and ID will be in **BOLD** characters.

Rung 2: This CTRIO Run To Limit Mode IBox sets up Output #2 in CTRIO #1 to output pulses at a Frequency of 1000 Hz until Limit #0 comes on. This example program requires that you load CTRRTLM_IBox.cwb into your Hx-CTRIO(2) module.



CTRRTLM Example (cont'd)

Rung 3: If the Run To Limit Mode parameters are OK, set the Direction Bit and Enable the output.



CTRIO Run to Position Mode (CTRRTPM) (IB-1012)

CTRIO Run To Position Mode, on a leading edge transition to this IBox, loads the Run to Position command and given parameters on a specific Output resource.

230

240

Valid Function Values are:

250-1

00: Less Than Ch1/Fn1

260

10: Greater Than Ch1/Fn1

262

01: Less Than Ch1/Fn2

DS5	Used
HPP	N/A

11: Greater Than Ch1/Fn2

02: Less Than Ch2/Fn1

12: Greater Than Ch2/Fn1

03: Less Than Ch2/Fn2

13: Greater Than Ch2/Fn2

This IBox will take more than one PLC scan to execute. Either the Success or Error bit will turn on when the command is complete. If the Error Bit is on, you can use the CTRIO Read Error Code (CTRRDER) IBox to get extended error information.

The Workspace register is for internal use by this IBox instruction and **MUST NOT** be used anywhere else in your program.

CTRRTPM Parameters

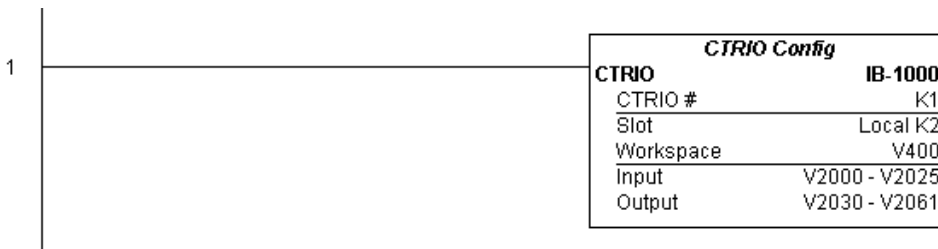
- CTRIO#: specifies a specific CTRIO module based on a user-defined number (see CTRIO Config Ibox).
- Output#: specifies a CTRIO output to be used by the instruction.
- Frequency: specifies the output pulse rate (H2-CTRIO: 20Hz–25KHz / H2-CTRIO2: 20Hz–250 KHz).
- Duty Cycle: specifies the % of on time versus off time. This is a hex number. Default of 0 is 50%, also entering 50 will yield 50%. 50% duty cycle is defined as on half the time and off half the time.
- Position: specifies the count value, as measured on the encoder input, at which the output pulse train will be turned off.
- Workspace: specifies a V-memory location that will be used by the instruction.
- Success: specifies a bit that will turn on once the instruction has completed successfully.
- Error: specifies a bit that will turn on if the instruction does not complete successfully.

CTRIO Run To Position Mode	
CTRRTPM	IB-1012
CTRIO #	K1
Output #	K0
Frequency	K1000
Function	K10
Duty Cycle	K0
Position	K1500
Workspace	V401
Success	C100
Error	C101

Parameter		DL205 Range
CTRIO#	K	K0-255
Output#	K	K0-3
Frequency	V,K	K20-20000; See DL205 V-memory map–Data Words
Duty Cycle	V,K	K0-99; See DL205 V-memory map
Position	V,K	K0-2147434528; See DL205 V-memory map
Workspace	V	See DL205 V-memory map–Data Words
Success	X,Y,C,GX,GY,B	See DL205 V-memory map
Error	X,Y,C,GX,GY,B	See DL205 V-memory map

CTRRTPM Example

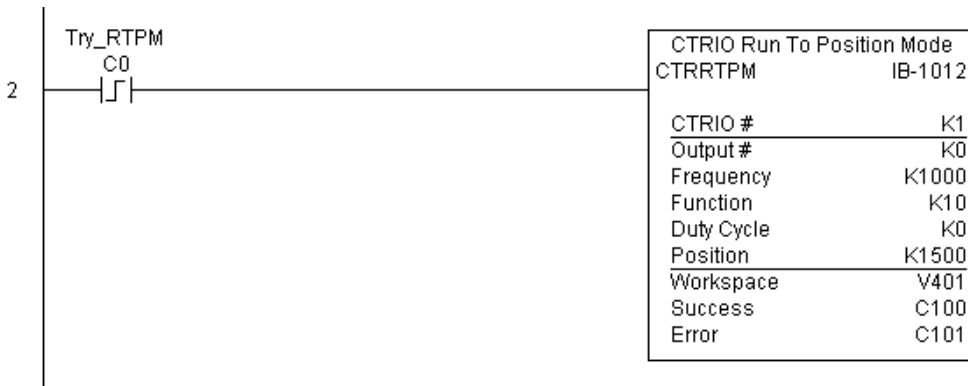
Rung 1: This sets up the CTRIO module in slot 2 of the local base. Each CTRIO module in the system will need a separate CTRIO Config IBox before any CTRxxxx IBoxes can be used. The CTRIO has been configured to use V2000 through V2025 for its input data, and V2030 through V2061 for its output data.



NOTE: The CTRIO Configuration IBox instruction does not require a permissive contact. The top line will be identified in **BOLD** italics, and the instruction name and ID will be in **BOLD** characters.

CTRRTPM Example (cont'd)

Rung 2: This CTRIO Run To Position Mode IBox sets up Output #0 in CTRIO #1 to output pulses at a Frequency of 1000 Hz, use the 'Greater than Ch1/Fn1' comparison operator, until the input position of 1500 is reached. This example program requires that you load CTRRTPM_IBox.cwb into your Hx-CTRIO(2) module.



Rung 3: If the Run To Position Mode parameters are OK, set the Direction Bit and Enable the output.



CTRIO Velocity Mode (CTRVELO) (IB-1013)

CTRIO Velocity Mode loads the Velocity command and given parameters on a specific Output resource on a leading edge transition to this IBox.

230

240

250-1

260

262

This IBox will take more than one PLC scan to execute. Either the Success or Error bit will turn on when the command is complete. If the Error Bit is on, you can use the CTRIO Read Error Code (CTRRDER) IBox to get extended error information.

The Workspace register is for internal use by this IBox instruction and MUST NOT be used anywhere else in your program.

DS5	Used
HPP	N/A

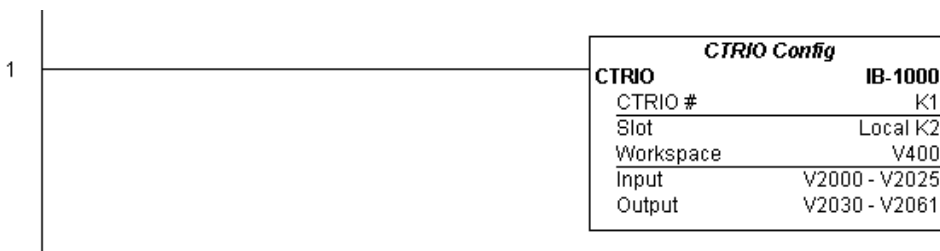
CTRVELO Parameters

- CTRIO#: specifies a specific CTRIO module based on a user defined number (see CTRIO Config Ibox)
- Output#: specifies a CTRIO output to be used by the instruction
- Frequency: specifies the output pulse rate (H2-CTRIO: 20Hz–25KHz / H2-CTRIO2: 20Hz–250 KHz)
- Duty Cycle: specifies the % of on time versus off time. This is a hex number. Default of 0 is 50%, also entering 50 will yield 50%. 50% duty cycle is defined as on half the time and off half the time
- Step Count: This DWORD value specifies the number of pulses to output. A Step Count value of -1 (or 0xFFFFFFFF) causes the CTRIO to output pulses continuously. Negative Step Count values must be V-Memory references.
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the instruction has successfully completed
- Error: specifies a bit that will turn on if the instruction does not complete successfully

Parameter		DL205 Range
CTRIO#	K	K0-255
Output#	K	K0-3
Frequency	V,K	K20-20000; See DL205 V-memory map–Data Words
Duty Cycle	V,K	K0-99; See DL205 V-memory map
Step Count	V,K	K0-2147434528; See DL205 V-memory map
Workspace	V	See DL205 V-memory map–Data Words
Success	X,Y,C,GX,GY,B	See DL205 V-memory map
Error	X,Y,C,GX,GY,B	See DL205 V-memory map

CTRVELO Example

Rung 1: This sets up the CTRIO module in slot 2 of the local base. Each CTRIO module in the system will need a separate CTRIO Config IBox before any CTRxxxx IBoxes can be used. The CTRIO has been configured to use V2000 through V2025 for its input data, and V2030 through V2061 for its output data.



NOTE: The CTRIO Configuration IBox instruction does not require a permissive contact. The top line will be identified in **BOLD italics**, and the instruction name and ID will be in **BOLD characters**.

Rung 2: This CTRIO Velocity Mode IBox sets up Output #0 in CTRIO #1 to output 10,000 pulses at a Frequency of 1000 Hz. This example program requires that you load CTRVELO_IBox.cwb into your Hx-CTRIO(2) module.



CTRVELO Example (cont'd)

Rung 3: If the Velocity Mode parameters are OK, set the Direction Bit and Enable the output.



CTRIO Write File to ROM (CTRWFTR) (IB-1006)

CTRIO Write File to ROM writes the runtime changes made to a loaded CTRIO Preset Table back to Flash ROM on a leading edge transition to this IBox. This IBox will take more than one PLC scan to execute. Either the Success or Error bit will turn on when the command is complete. If the Error Bit is on, you can use the CTRIO Read Error Code (CTRRDER) IBox to get extended error information.

✗ 230

✗ 240

✓ 250-1

✓ 260

✓ 262

The Workspace register is for internal use by this IBox instruction and MUST NOT be used anywhere else in your program.

CTRWFTR Parameters

DS5	Used
HPP	N/A

- CTRIO#: specifies a specific CTRIO module based on a user defined number (see CTRIO Config Ibox)
- Output#: specifies a CTRIO output to be used by the instruction
- Workspace: specifies a V-memory location that will be used by the instruction
- Success: specifies a bit that will turn on once the instruction has completed successfully
- Error: specifies a bit that will turn on if the instruction does not complete successfully

CTRIO Write File to ROM

CTRWFTR IB-1006

CTRIO # K1

Output # K0

Workspace V404

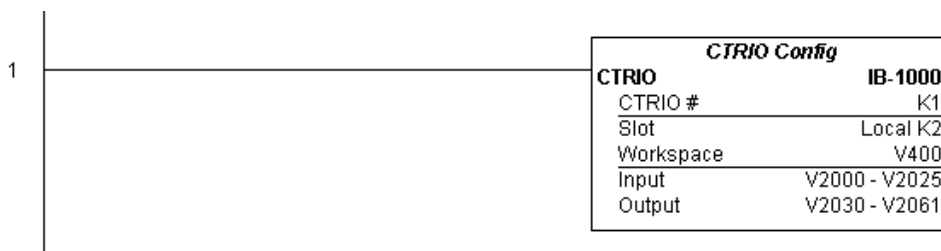
Success C100

Error C101

Parameter		DL205 Range
CTRIO#	K	K0-255
Output#	K	K0-3
Workspace	V	See DL205 V-memory map-Data Words
Success	X,Y,C,GX,GY,B	See DL205 V-memory map
Error	X,Y,C,GX,GY,B	See DL205 V-memory map

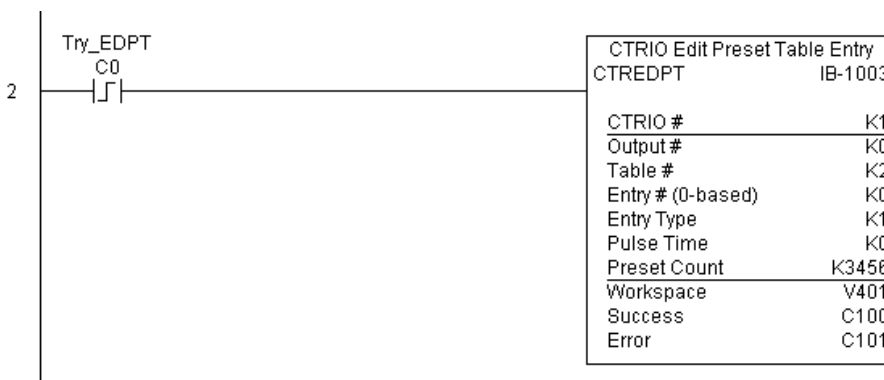
CTRWFTTR Example

Rung 1: This sets up the CTTRIO module in slot 2 of the local base. Each CTTRIO module in the system will need a separate CTTRIO Config IBox before any CTTRxxx IBoxes can be used. The CTTRIO has been configured to use V2000 through V2025 for its input data, and V2030 through V2061 for its output data.



NOTE: The CTTRIO Configuration IBox instruction does not require a permissive contact. The top line will be identified in **BOLD** italics, and the instruction name and ID will be in **BOLD** characters.

Rung 2: This CTTRIO Edit Preset Table Entry IBox will change Entry 0 in Table #2 to be a RESET at Count 3456. This example program requires that you load CTTRWFTR_IBox.cwb into your Hx-CTTRIO(2) module.



(Example continued on next page)

CTRWFTR Example (cont'd)

Rung 3: If the file is successfully edited, use a Write File To ROM IBox to save the edited table back to the CTRIO's ROM, thereby making the changes retentive.

