STANDARD RLL INSTRUCTIONS

In This Chapter...

Introduction

D4-454 PLCs offer a wide variety of instructions to perform many different types of operations. This chapter shows you how to use each standard Relay Ladder Logic (RLL) instruction. In addition to these instructions, you may also need to refer to the Drum instruction in Chapter 6, the Stage programming instructions in Chapter 7, PID in Chapter 10 and programming for analog modules in D4-ANLG-M.

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

- If you know the instruction category (Boolean, Comparative Boolean, etc.), just use the title 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(s) that discusses the instruction.

Using Boolean Instructions

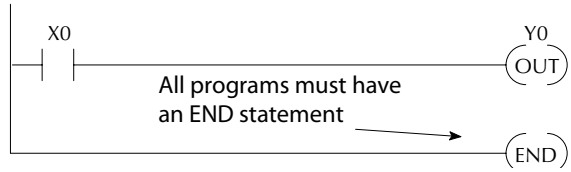
Do you ever wonder why so many PLC manufacturers always quote the scan time for a 1K Boolean program? Simple. Most programs utilize many Boolean instructions. These are typically very simple instructions designed to join input and output contacts in various series and parallel combinations. Our DirectSOFT software 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, however, they can be seen in a Mnemonic View of the program once the DirectSOFT program has been developed and accepted (compiled).

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

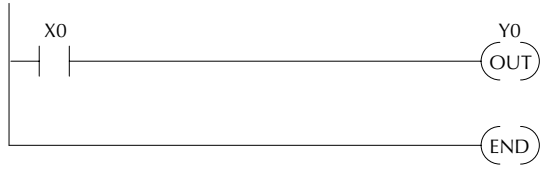
END Statement

D4-454 programs require an END statement as the last instruction. This tells the CPU that this is the end of the program. Normally, any instructions placed after the END statement will not be executed. There are exceptions to this, such as interrupt routines, etc. This chapter will discuss the instruction set in detail.



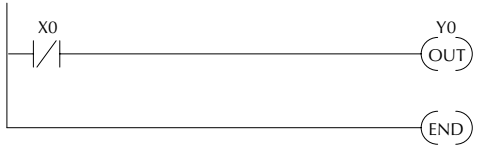
Simple Rungs

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.



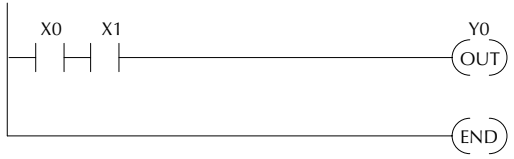
Normally Closed Contact

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.



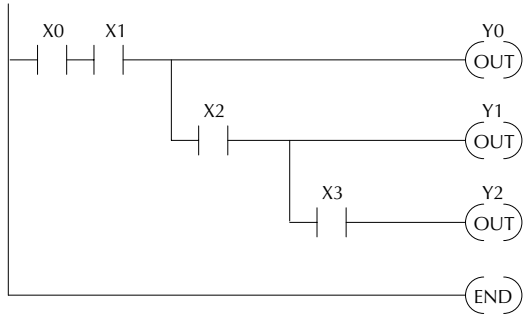
Contacts in Series

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.



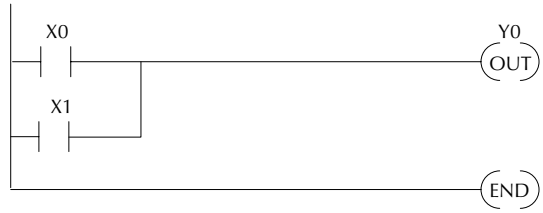
Midline Outputs

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.



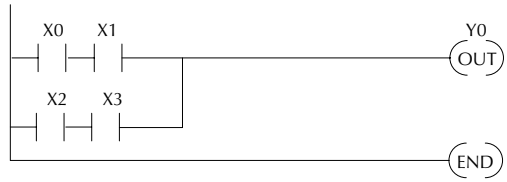
Parallel Elements

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.



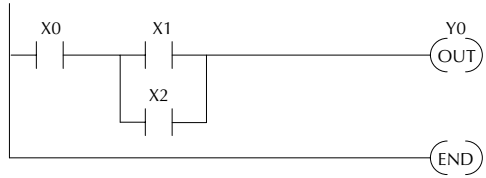
Joining Series Branches in Parallel

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.



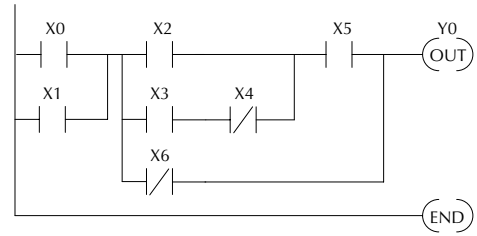
Joining Parallel Branches in Series

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

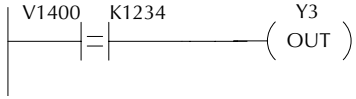
You can combine the various types of series and parallel branches to solve almost any application problem. The following example shows a simple combination network.



Comparative Boolean

The Comparative Boolean provides evaluation of two BCD values using boolean contacts. The valid evaluations are: equal to, not equal to, equal to or greater than, and less than.

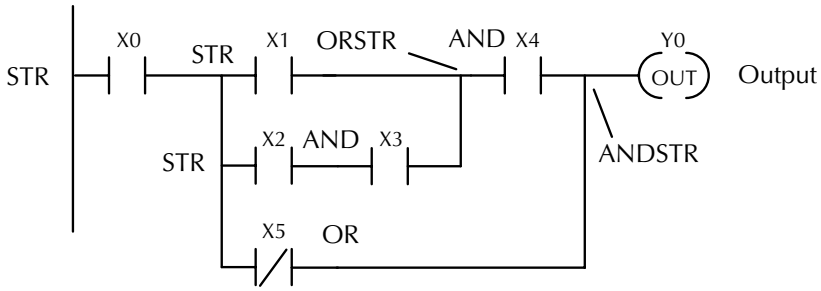
In this example, when the BCD value in V-memory location V1400 is equal to the constant value 1234, Y3 will energize.



Boolean Stack

There are limits to how many elements you can include in a rung. This is because the D4-454 PLCs 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 the program encounters a STR instruction, the instruction is placed on the top of the stack. Any other STR instructions already on the boolean stack are pushed down a level. The ANDSTR, and ORSTR instructions combine levels of the boolean stack when they are encountered. An error will occur during program compilation if the CPU encounters a rung that uses more than the eight levels of the boolean stack.

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



STR X0

1	STR X0
2	
3	
4	

STR X1

1	STR X1
2	STR X0
3	
4	

STR X2

1	STR X2
2	STR X1
3	STR X0
4	

AND X3

1	X2 AND X3
2	STR X1
3	STR X0
4	

ORSTR

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

AND X4

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

ORNOT X5

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

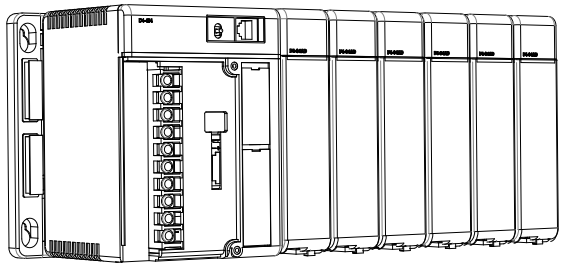
ANDSTR

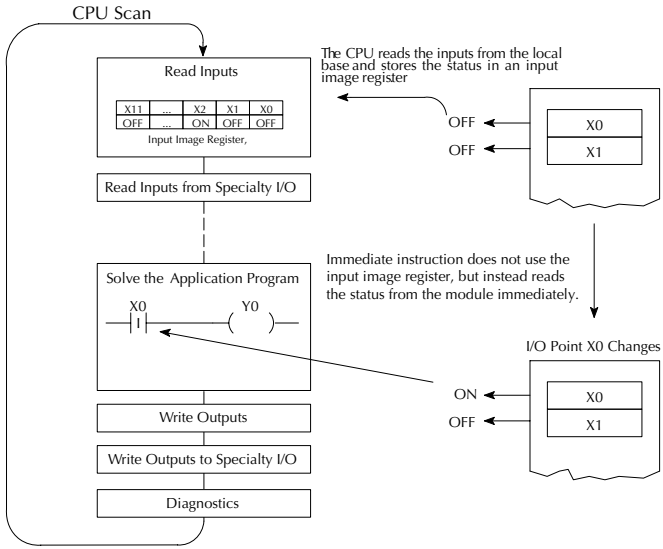
1	XO AND (NOT X5 or X4) AND {X1 or (X2 AND X3)}
2	
3	

Immediate Boolean

The D4-454 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 PLCs offer Immediate input and outputs which are special boolean instructions that allow reading directly from inputs and writing directly to outputs during the program execution portion of the CPU cycle. You may recall that this is normally done during the input or output update portion of the CPU cycle. The immediate instructions take longer to execute because the program execution is interrupted while the CPU reads or writes the I/O point. This function is not normally done until the read inputs or the write outputs portion of the CPU cycle.

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

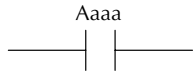




Boolean Instructions

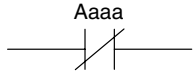
Store (STR)

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



Store Not (STRN)

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



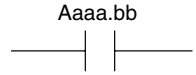
Operand Data Type		D4-454 Range
	A	aaa
Inputs	X	0–1777
Outputs	Y	0–1777
Control Relays	C	0–3777
Stage	S	0–1777
Timer	T	0–377
Counter	CT	0–377
Special Relay	SP	0–777

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

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

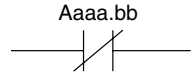
Store Bit-of-Word (STRB)

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



Store Not Bit-of-Word (STRNB)

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



These instructions look like the STR and STRN instructions only the address is different. Take note how the address is set up in the following Store Bit-of-Word example.

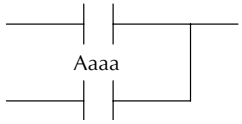
Operand Data Type		D4-454 Range	
	A	aaa	bb
V-memory	B	See memory map	0 to 15
Pointer	PB	See memory map	0 to 15

When bit 12 of V-memory location V1400 is on, output Y2 will energize.

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

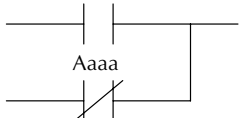
Or (OR)

The Or instruction will logically OR 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 will logically OR a normally closed contact in parallel with another contact in a rung. The status of the contact will be opposite the state of the associated image register point or memory location.



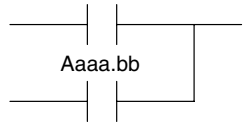
Operand Data Type		D4-454 Range
A		aaa
Inputs	X	0–1777
Outputs	Y	0–1777
Control Relays	C	0–3777
Stage	S	0–1777
Timer	T	0–377
Counter	CT	0–377
Special Relay	SP	0–777

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.

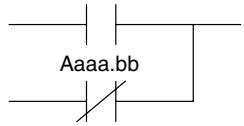
Or Bit-of-Word (OR)

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



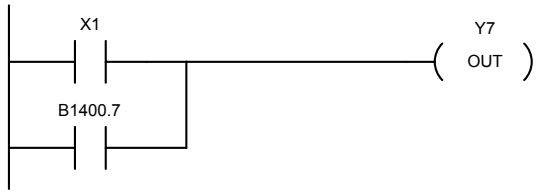
Or Not Bit-of-Word (ORN)

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

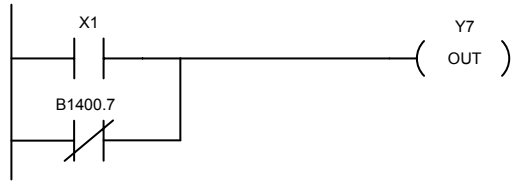


Operand Data Type		D4-454 Range	
	A	aaa	bb
V-memory	B	See memory map	0 to 15
Pointer	PB	See memory map	0 to 15

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



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

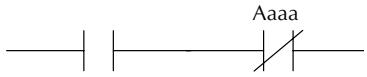


AND (AND)

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.

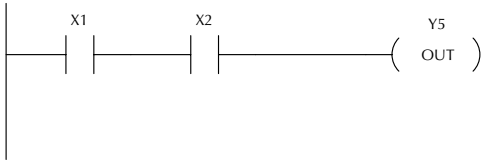
AND NOT (ANDN)

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.

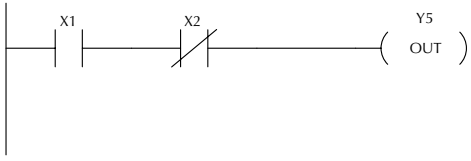


Operand Data Type		D4-454 Range
	A	aaa
Inputs	X	0–1777
Outputs	Y	0–1777
Control Relays	C	0–3777
Stage	S	0–1777
Timer	T	0–377
Counter	CT	0–377
Special Relay	SP	0–777

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

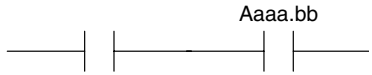


In the following And Not example, when input X1 is on and X2 is off output Y5 will energize.



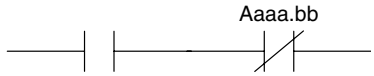
AND Bit-of-Word (AND)

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



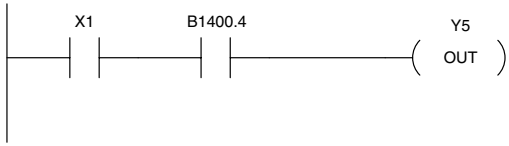
AND Not Bit-of-Word (ANDN)

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

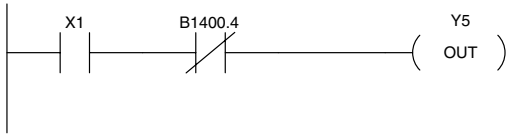


Operand Data Type		D4-454 Range	
	A	aaa	bb
V-memory	B	See memory map	0 to 15
Pointer	PB	See memory map	0 to 15

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

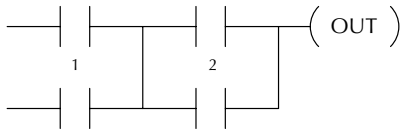


In the following And Not Bit-of-Word example, when input X1 is on and bit 4 of V1400 is off output Y5 will energize.



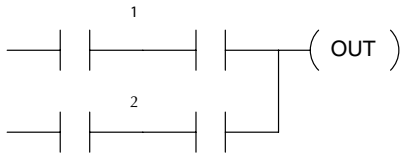
And Store (ANDSTR)

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

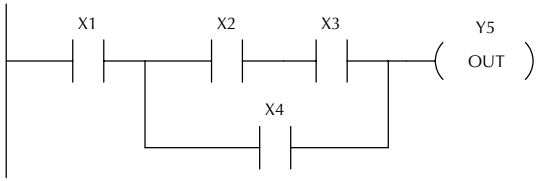


OR Store (ORSTR)

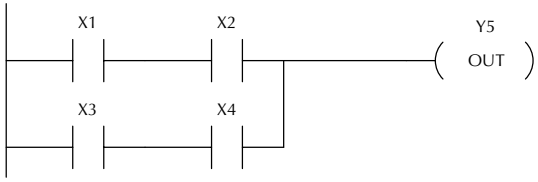
The OR Store instruction logically ORs two branches of a rung in parallel. Both branches must begin with the Store instruction.



In the following And Store example, the branch consisting of contacts X2, X3, and X4 have been anded with the branch consisting of contact X1.

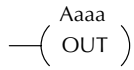


In the following OR Store example, the branch consisting of X1 and X2 have been OR'd with the branch consisting of X3 and X4.



Out (OUT)

The Out instruction reflects the status of the rung (on/off) and outputs the discrete (on/off) state to the specified image register point or memory location.



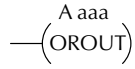
Multiple Out instructions referencing the same discrete location should not be used since only the last Out instruction in the program will control the physical output point. Instead, use the next instruction, the Or Out.

Operand Data Type		D4-454 Range
	A	aaa
Inputs	X	0 – 1777
Outputs	Y	0 – 1777
Control Relays	C	0 – 3777

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

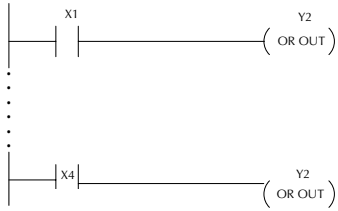
Or Out (OROUT)

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



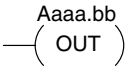
Operand Data Type		D4-454 Range
	A	aaa
Inputs	X	0 – 1777
Outputs	Y	0 – 1777
Control Relays	C	0 – 3777

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



Out Bit-of-Word (OUT)

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.

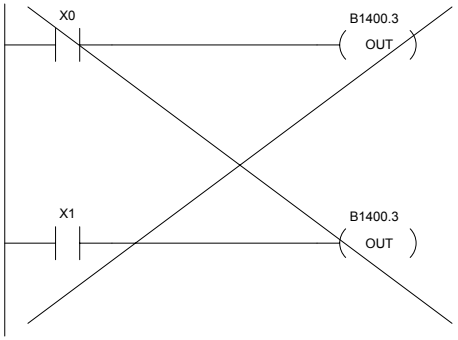


Operand Data Type		D4-454 Range	
	A	aaa	bb
V-memory	B	See memory map	0 to 15
Pointer	PB	See memory map	0 to 15

NOTE: *If the Bit-of-Word is entered as V1400.3 in DirectSOFT, it will be converted to B1400.3. Bit-of-Word can also be entered as B1400.3.*

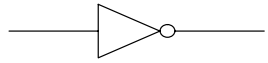
In the following Out Bit-of-Word example, when input X1 is on, bit 3 of V1400 and bit 6 of V1401 will turn on.

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.

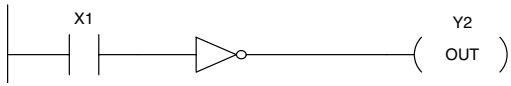


Not (NOT)

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

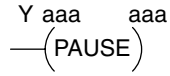


In the following example, when X1 is off, Y2 will energize. This is because the NOT instruction inverts the status of the rung at the NOT instruction.



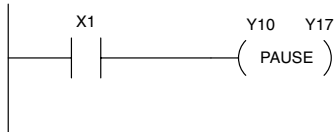
Pause (PAUSE)

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



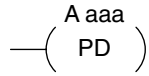
In the following example, when X1 is ON, Y1-Y17 will be turned OFF at the output module. The execution of the ladder program will not be affected.

Operand Data Type		D4-454 Range
	A	aaa
Outputs	Y	0–1777



Positive Differential (PD)

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

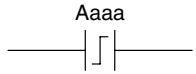


In the following example, every time X1 makes an Off-to-On transition, C0 will energize for one scan.

Operand Data Type		D4-454 Range
	A	aaa
Inputs	X	0–1777
Outputs	Y	0–1777
Control Relays	C	0–3777

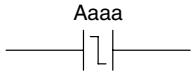
Store Positive Differential (STRPD)

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.



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).



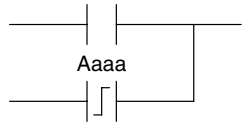
Operand Data Type		D4-454 Range
	A	aaa
Inputs	X	0 – 1777
Outputs	Y	0 – 1777
Control Relays	C	0 – 3777
Stage	S	0 – 1777
Timer	T	0 – 377
Counter	CT	0 – 377

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

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

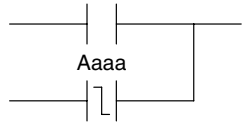
Or Positive Differential (ORPD)

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.



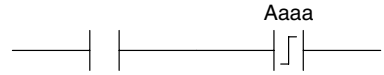
Operand Data Type		D4-454 Range
	A	aaa
Inputs	X	0 – 1777
Outputs	Y	0 – 1777
Control Relays	C	0 – 3777
Stage	S	0 – 1777
Timer	T	0 – 377
Counter	CT	0 – 177

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

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

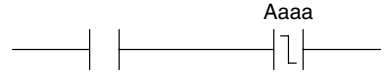
And Positive Differential (ANDPD)

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



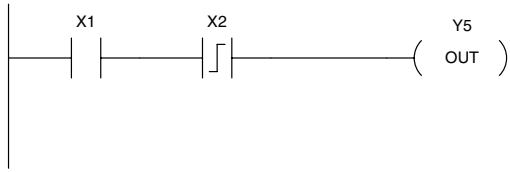
And Negative Differential (ANDND)

The And Negative Differential instruction logically ANDs a normally open contact in series with another contact 5-22 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.

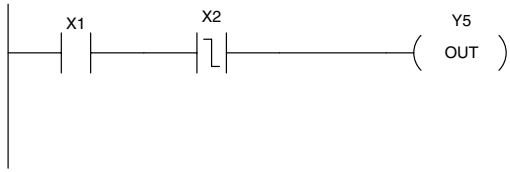


Operand Data Type		D4-454 Range
	A	aaa
Inputs	X	0 – 1777
Outputs	Y	0 – 1777
Control Relays	C	0 – 3777
Stage	S	0 – 1777
Timer	T	0 – 377
Counter	CT	0 – 377

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

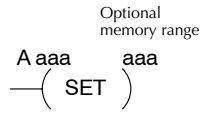


In the following example, Y5 will energize for one CPU scan whenever X1 is on and X2 transitions from On to Off.



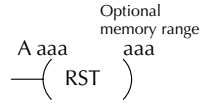
Set (SET)

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



Reset (RST)

The Reset instruction reset,s 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.



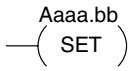
Operand Data Type		D4-454 Range
	A	aaa
Inputs	X	0 – 1777
Outputs	Y	0 – 1777
Control Relays	C	0 – 3777
Stage	S	0 – 1777
Timer	T	0 – 377
Counter	CT	0 – 377

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

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

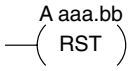
Set Bit-of-Word (SET)

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 (RST)

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		D4-454 Range	
	A	aaa	bb
V-memory	B	See memory map	0 to 15
Pointer	PB	See memory map	0 to 15

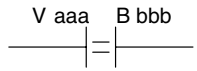
In the following example, when X1 turns on, bit 1 in V1400 is set to the on state.

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

Comparative Boolean

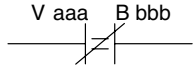
Store If Equal (STRE)

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



Store If Not Equal (STRNE)

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



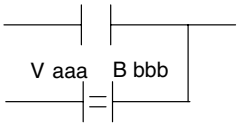
Operand Data Type		D4-454 Range	
	B	aaa	bbb
V-memory	V	See memory map	See memory map
Pointer	P	See memory map	See memory map
Constant	K	– –	0 – 9999

In the following example, when the BCD value in V-memory location V2000 is equal to 4933, Y3 will energize.

In the following example, when the BCD value in V-memory location V2000 does not equal 5060, Y3 will energize.

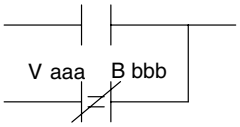
Or If Equal (ORE)

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



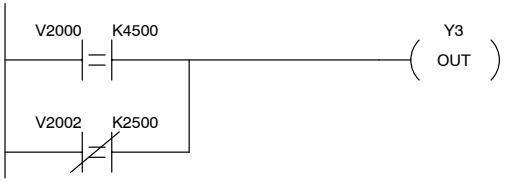
Or If Not Equal (ORNE)

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

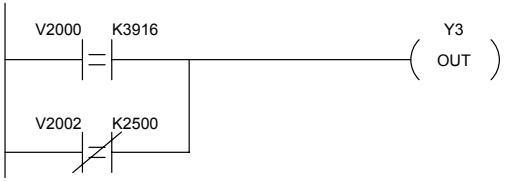


Operand Data Type		D4-454 Range	
	B	aaa	bbb
V-memory	V	See memory map	See memory map
Pointer	P	See memory map	See memory map
Constant	K	– –	0 – 9999

In the following example, when the BCD value in V-memory location V2000 is equal to 4500 or V2002 does not equal 2500, Y3 will energize.

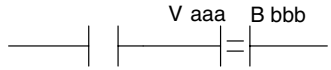


In the following example, when the BCD value in V-memory location V2000 is equal to 3916 or V2002 does not equal 2500, Y3 will energize



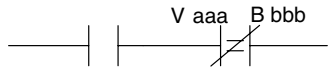
And If Equal (ANDE)

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



And If Not Equal (ANDNE)

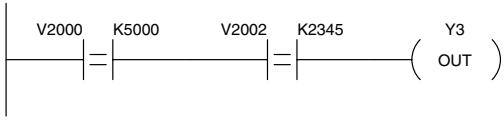
The And If Not Equal instruction connects a normally closed comparative contact in series with another contact. The contact will be on when Vaaa does not equal Bbb.



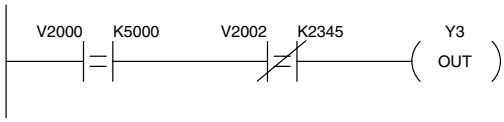
Operand Data Type		D4-454 Range	
	B	aaa	bbb
V-memory	V	See memory map	See memory map
Pointer	P	See memory map	See memory map
Constant	K	– –	0 – 9999

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

In the following example, when the BCD value in V-memory location V2000 is equal to

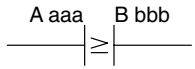


5000 and V2002 does not equal 2345, Y3 will energize.



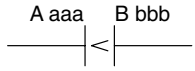
Store (STR)

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



Store Not (STRN)

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



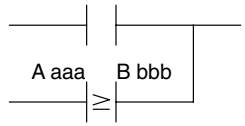
Operand Data Type		D4-454 Range	
	A/B	aaa	bbb
V-memory	V	See memory map	See memory map
Pointer	p	See memory map	See memory map
Constant	K	– –	0 – 9999
Timer	TA	0 – 377	
Counter	CTA	0 – 377	

In the following example, when the BCD value in V-memory location V2000 is equal to or greater than 1000, Y3 will energize.

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

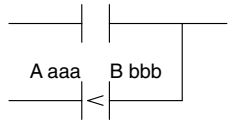
Or (OR)

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



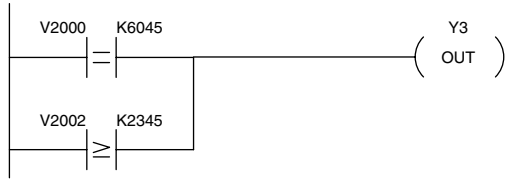
Or Not (ORN)

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

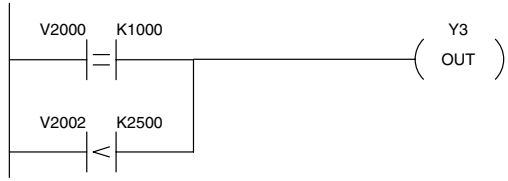


Operand Data Type		D4-454 Range	
	A/B	aaa	bbb
V-memory	V	See memory map	See memory map
Pointer	p	See memory map	See memory map
Constant	K	– –	0 – 9999
Timer	TA	0 – 377	
Counter	CTA	0 – 377	

In the following example, when the BCD value in V-memory location V2000 is equal to 6045 or V2002 is equal to or greater than 2345, Y3 will energize.

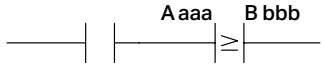


In the following example when the BCD value in V-memory location V2000 is equal to 1000 or V2002 is less than 2500, Y3 will energize.



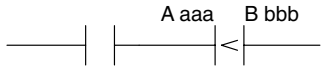
And (AND)

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



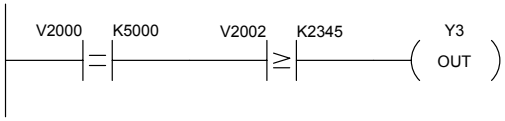
And Not (ANDN)

The Comparative And Not instruction connects a normally closed comparative contact in series with another contact. The contact will be on when Aaaa is less than Bbbb.

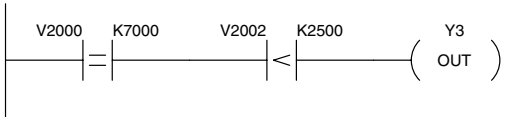


Operand Data Type		D4-454 Range	
	A/B	aaa	bbb
V-memory	V	See memory map	See memory map
Pointer	p	See memory map	See memory map
Constant	K	– –	0 – 9999
Timer	TA	0 – 377	
Counter	CTA	0 – 377	

In the following example, when the value in BCD V-memory location V2000 is equal to 5000, and V2002 is equal to or greater than 2345, Y3 will energize.



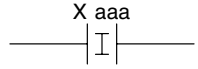
In the following example, when the value in V-memory location V2000 is equal to 7000 and V2002 is less than 2500, Y3 will energize.



Immediate Instructions

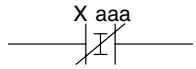
Store Immediate (STRI)

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



Store Not Immediate (STRNI)

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



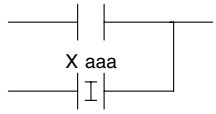
Operand Data Type		D4-454 Range
		aaa
Inputs	X	0–1777

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

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

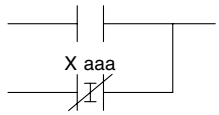
Or Immediate (ORI)

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



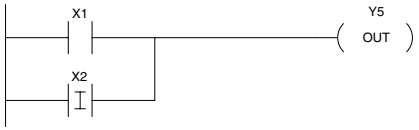
Or Not Immediate (ORNI)

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

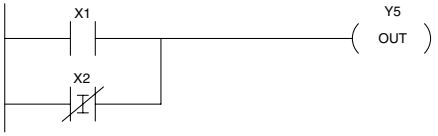


Operand Data Type		D4-454 Range
		aaa
Inputs	X	0–1777

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

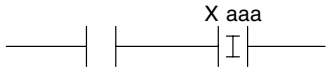


In the following example, when X1 is on or X2 is off, Y5 will energize.



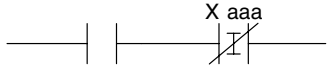
And Immediate (ANDI)

The And Immediate instruction connects two contacts in series. 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.



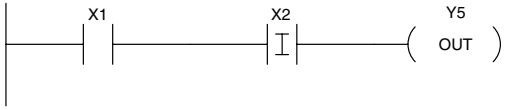
And Not Immediate (ANDNI)

The And Not Immediate instruction 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.

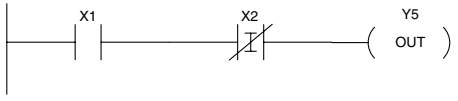


Operand Data Type		D4-454 Range
		aaa
Inputs	X	0–1777

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

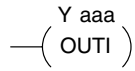


In the following example, when X1 is on and X2 is off, Y5 will energize.



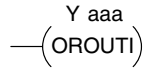
Out Immediate (OUTI)

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



Or Out Immediate (OROUTI)

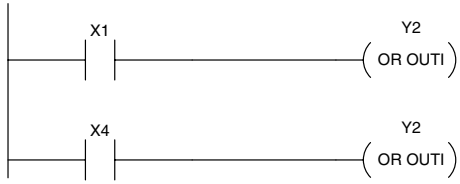
The Or Out Immediate instruction has been designed to use more than 1 rung of discrete logic to control a single output. Multiple Or Out Immediate instructions referencing the same output coil may be used, since all contacts controlling the output are OR'd together. If the status of any rung is on at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



Operand Data Type		D4-454 Range
		aaa
Outputs	Y	0–1777

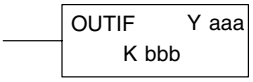
In the following example, when X1 is on, output point Y2 on the output module will turn on. .

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



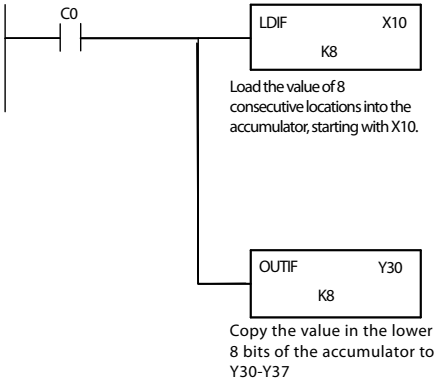
Out Immediate Formatted (OUTIF)

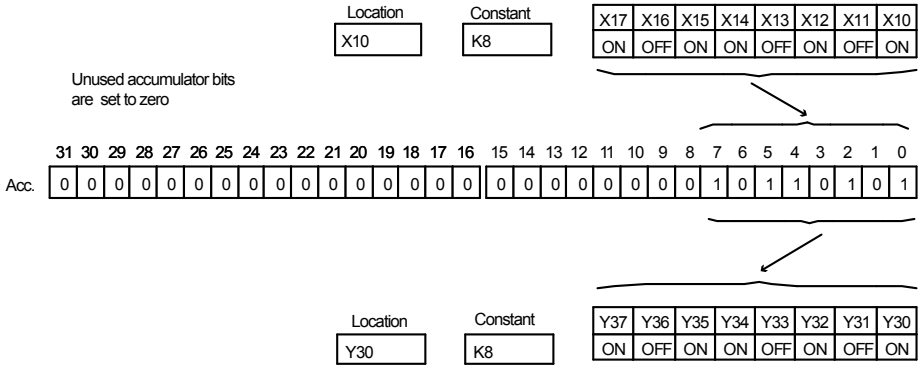
The Out Immediate Formatted instruction outputs a 1–32 bit binary value from the accumulator to specified output points at the time the instruction is executed. Accumulator bits that are not used by the instruction are set to zero.



Operand Data Type		D4-454 Range
		aaa
Outputs	Y	0 – 1777
Constant	K	1 – 32

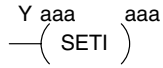
In the following example, when C0 is on,the binary pattern for X10 – X17 is loaded into the accumulator using the Load Immediate Formatted instruction. The binary pattern in the accumulator is written to Y30 – Y37 using the Out Immediate Formatted instruction. This technique is useful to quickly copy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





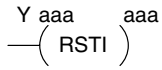
Set Immediate (SETI)

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



Reset Immediate (RSTI)

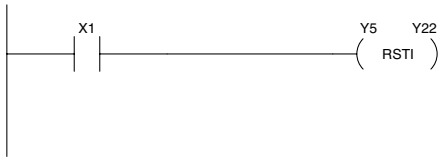
The Reset Immediate instruction immediately resets, or turns off, an output or a range of outputs in the image register and the output point(s) at the time the instruction is executed. The image register is not updated. Once the outputs are reset, it is not necessary for the input to remain on.



Operand Data Type		DL454 Range
		aaa
Outputs	Y	0 – 1777

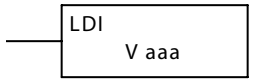
In the following example, when X1 is on, Y2 through Y5 will be set (on) for the corresponding output points.

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



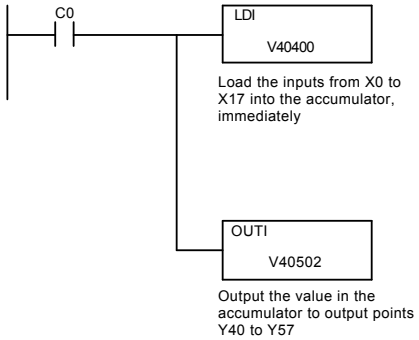
Load Immediate (LDI)

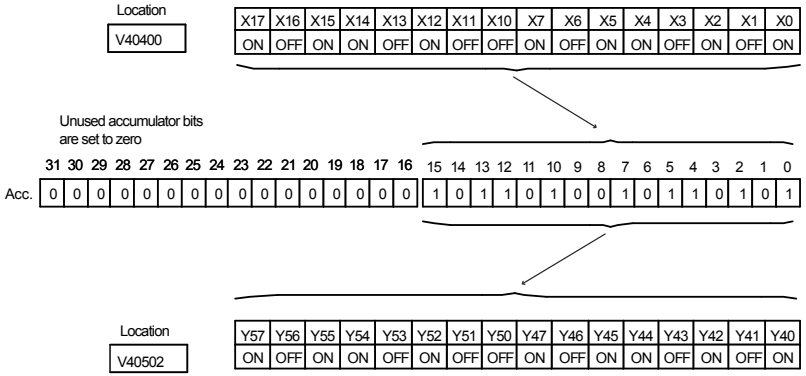
The Load Immediate instruction loads a 16-bit V-memory value into the accumulator. The valid address range includes all input point addresses on the local base. The value reflects the current status of the input points at the time the instruction is executed. This instruction may be used instead of the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



Operand Data Type		D4-454 Range
		aaa
Inputs	V	40400 – 40477

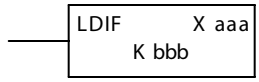
In the following example, when C0 is on, the binary pattern of X0 – X17 will be loaded into the accumulator using the Load Immediate instruction. The Out Immediate instruction could be used to copy the 16 bits in the accumulator to output points, such as Y40 – Y57. This technique is useful to quickly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





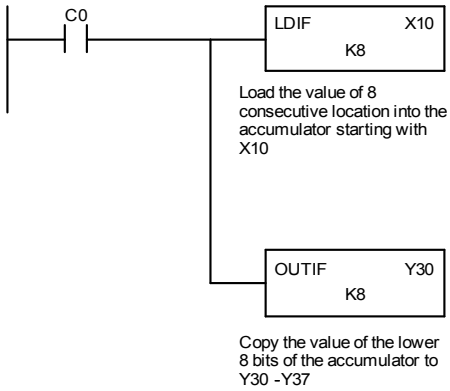
Load Immediate Formatted (LDIF)

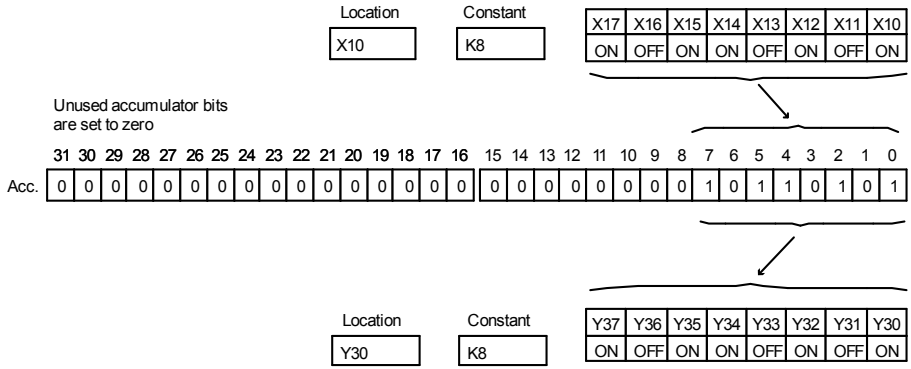
The Load Immediate Formatted instruction loads a 1–32 bit binary value into the accumulator. The value reflects the current status of the input module(s) at the time the instruction is executed. Accumulator bits that are not used by the instruction are set to zero.



Operand Data Type		D4-454 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).

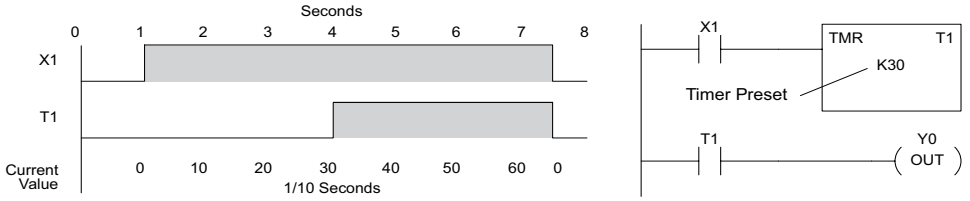




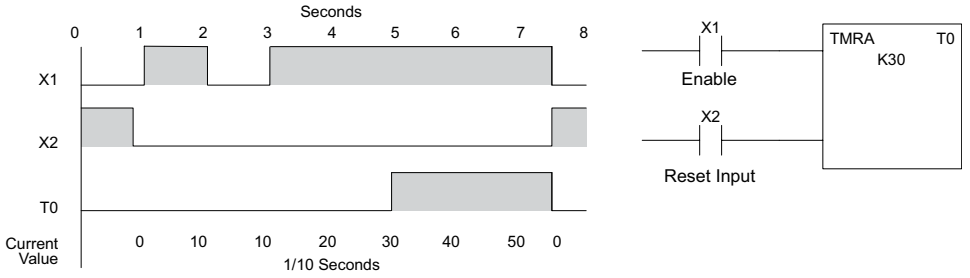
Timer, Counter and Shift Register Instructions

Using Timers

Timers are used to time an event for a desired period. The single input timer will time as long as the input is on. When the input changes from on to off, the timer current value is reset to 0. There is a tenth of a second and a hundredth of a second timer available with a maximum time of 999.9 and 99.99 seconds respectively. There is a discrete bit 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.



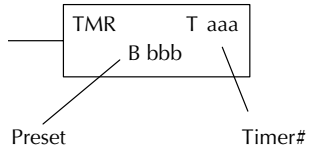
There are those applications that need an accumulating timer, meaning it has the ability to time, stop, and then resume from where it previously stopped. The accumulating timer works similarly to the regular timer, but two inputs are required. The enable input starts and stops the timer. When the timer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



NOTE: *Decimal points are not used in these timers, but the decimal point is implied. The preset and current value for all four timers is in BCD format.*

Timer (TMR) and Timer Fast (TMRF)

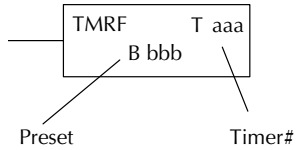
The Timer instruction is a 0.1 second single input timer that times to a maximum of 999.9 seconds. The Timer Fast instruction is a 0.01 second single input timer that times up to a maximum of 99.99 seconds. These timers will be enabled if the input logic is true (on) and will be reset to 0 if the input logic is false (off). Both timers use single word BCD values for the preset and current value. The decimal place is implied.



Instruction Specifications

Timer Reference (Taaa): Specifies the timer number.

Preset Value (Bbbb): Constant value (K) or a V-memory location specified in BCD.



Current Value: Timer current values, in BCD format, 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. Operating as a "timer done bit", 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 is T2.

NOTE: *A V-memory preset is required when the ladder program or an Operator Interface unit is used to change the preset.*

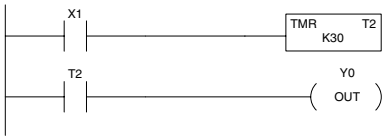
Operand Data Type		D4-454 Range	
	A/B	aaa	bbb
Timers	T	0 – 377	– –
V-memory for preset values	V	– –	1400 – 7377 10000-17777
Pointers (preset only)	P	– –	1400 – 7377 10000-17777
Constants (preset only)	K	– –	0 – 9999
Timer discrete status bits	T/V	0 – 377 or V41100 – 41117	
Timer current values	V /T**	0 – 377	

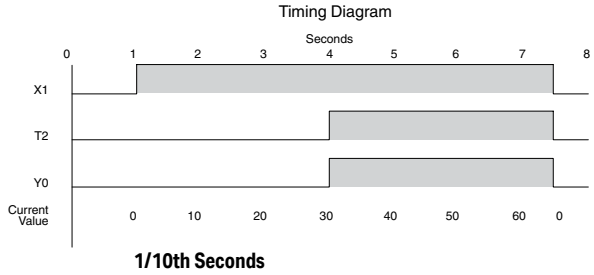
NOTE: *The DirectSOFT programming uses separate references, such as "T2" for discrete status bit for Timer T2, and "TA2" for the current value of Timer T2.*

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

Timer Example Using Discrete Status Bits

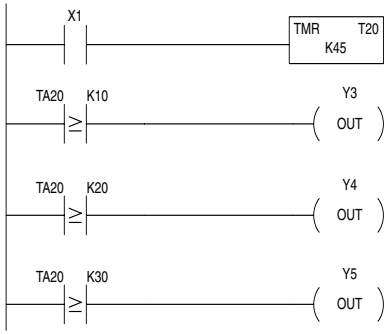
In the following example, a single input timer is used with a preset of 3 seconds. The timer discrete status bit (T2) will turn on when the timer has timed for 3 seconds. The timer is reset when X1 turns off, turning the discrete status bit off and resetting the timer current value to 0.

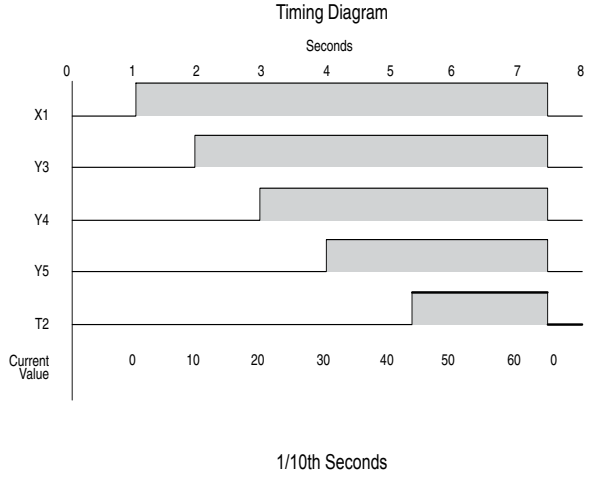




Timer Example Using Comparative Contacts

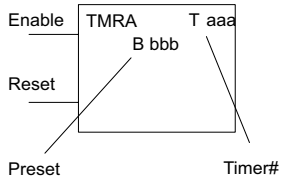
In the following example, a single input timer is used with a preset of 4.5 seconds. Comparative contacts are used to energize Y3, Y4, and Y5 at one second intervals respectively. When X1 is turned off, the timer will be reset to 0 and the comparative contacts will turn off Y3, Y4, and Y5.





Accumulating Timer (TMRA)

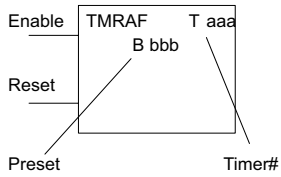
The Accumulating Timer is a 0.1 second two input timer that will time to a maximum of 9999999.9. The TMRA uses two timer registers in V-memory.



Accumulating Fast Timer (TMRAF)

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

Each timer uses two timer registers in V-memory. The preset and current values are in double word BCD format, and the decimal point is implied. These timers have two inputs, an enable and a reset. The timer starts timing when the enable is on and stops when the enable is off (without resetting the count). The reset will reset the timer when on and allow the timer to time when off.



Timer Reference (Taaa): Specifies the timer number.

Preset Value (Bbbb): Constant value (K) or V-memory.

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, V3.

Discrete Status Bit: The discrete status bit is accessed by referencing the associated T memory location. Operating as a "timer done bit," 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.

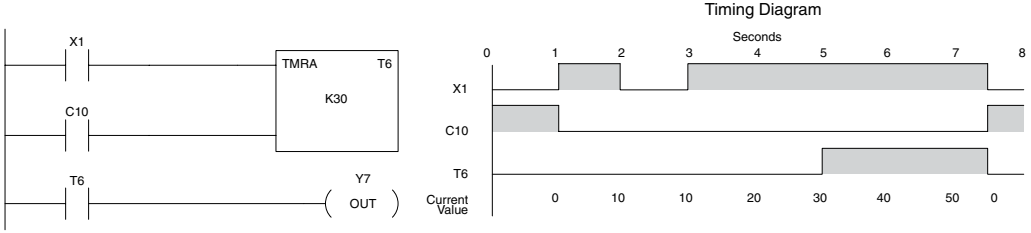
NOTE 1: *The accumulating timer uses two consecutive V-memory locations for the 8-digit value, therefore two consecutive timer locations. For example, if TMRA T1 is used, the next available timer number is T3*
NOTE 2: A V-Memory preset is required if the ladder program or an Operator Interface panel is used to change the preset.

Operand Data Type		D4-454 Range	
	A/B	aaa	bbb
Timers	T	0 – 376	– –
V-memory for preset values	V	– –	1400 – 7377 10000-37777
Pointers (preset only)	P	– –	1400 – 7377 10000-37777
Constants (preset only)	K	– –	0 – 99999999
Timer discrete status bits	T/V	0 – 377 or V41100 – 41117	
Timer current values	V /T**	0 – 377	

NOTE: *The DirectSOFT programming software uses separate references, such as "T2" for discrete status bit for Timer T2, and "TA2" for the current value of Timer T2.*

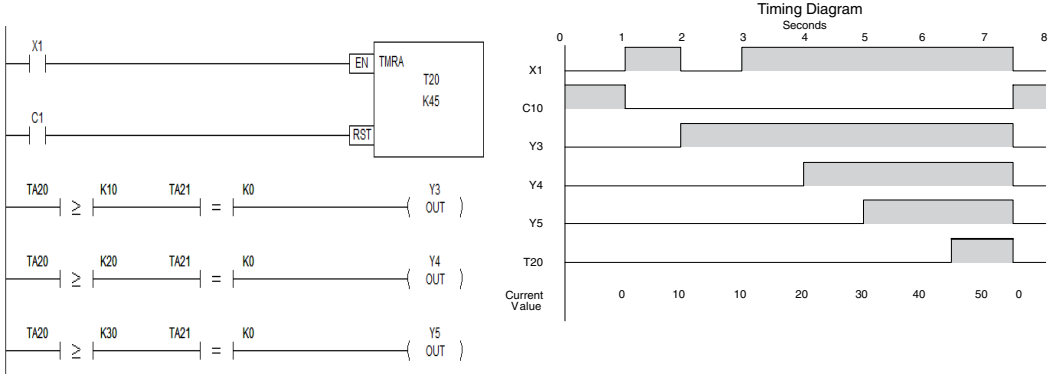
Accumulating Timer Example using Discrete Status Bits

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



Accumulator Timer Example Using Comparative Contacts

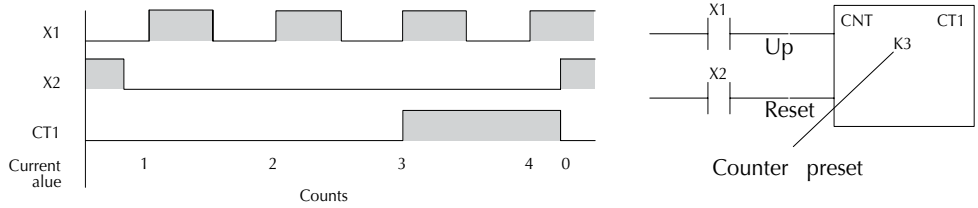
In the following example, a single input timer is used with a preset of 4.5 seconds. Comparative contacts are used to energized Y3, Y4, and Y5 at one second intervals respectively. The comparative contacts will turn off when the timer is reset.



Using Counters

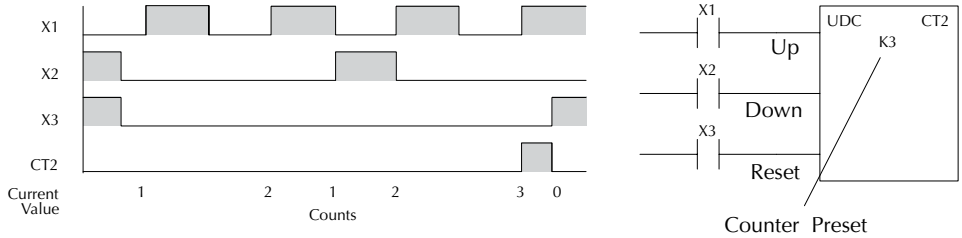
Counters are used to count events . The counters available are up counters, up/down counters, and stage counters (used with RLLPLUS programming).

The up counter (CNT) has two inputs, a count input and a reset input. The maximum count value is 9999. The timing diagram below shows the relationship between the counter input, counter reset, associated discrete bit, current value, and counter preset. The CNT counter preset and current value are both single word BCD values.

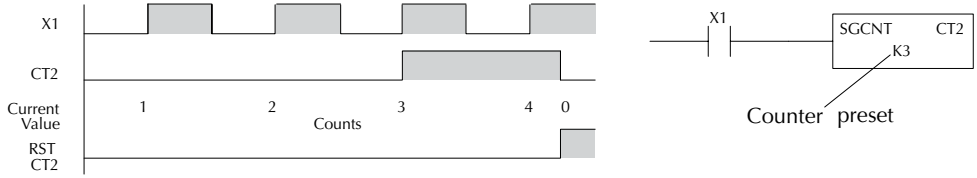


The up down counter (UDC) has three inputs, a count up input, count down input and reset input. The maximum count value is 99999999. The timing diagram below shows the relationship between the counter up and down inputs, counter reset, associated discrete bit, current value, and counter preset. The UDC counter preset and current value are both double word BCD values.

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

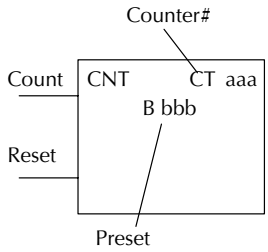


The stage counter (SGCNT) has a count input and is reset by the RST instruction. This instruction is useful when programming using the RLLPLUS structured programming. The maximum count value is 9999. The timing diagram below shows the relationship between the counter input, associated discrete bit, current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



Counter (CNT)

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



Instruction Specifications

Counter Reference (CTaaa): Specifies the counter number.

Preset Value (Bbbb): Constant value (K) or a V-memory location.

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.

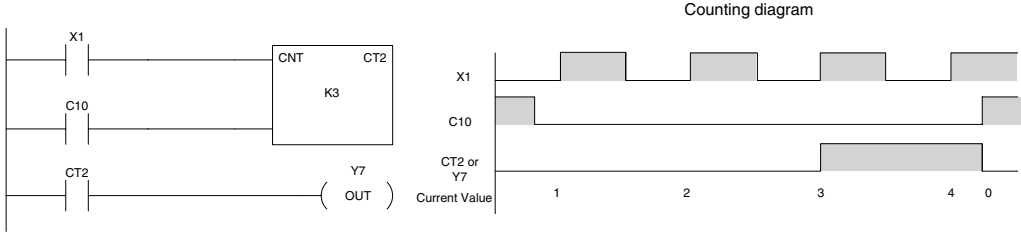
NOTE: *A V-memory preset is required if the ladder program or Operator Interface Panel is used to change the preset.*

Operand Data Type		D4-454 Range	
	A/B	aaa	bbb
Counters	CT	0 – 377	– –
V-memory (preset only)	V	– –	1400 – 7377 10000 – 37777
Pointers (preset only)	P	– –	1400 – 7377 10000 – 37777
Constants (preset only)	K	– –	0 – 9999
Counter discrete status bits	CT/V	0 – 377 or V41140 – V41157	
Counter current values	V /CT**	1000-1377	

NOTE: **May be non-volatile if MOV instruction is used. * DirectSOFT programming software uses separate references, such as "CT2" for discrete status bit for Counter CT2, and "CTA2" for the current value of Counter CT2.*

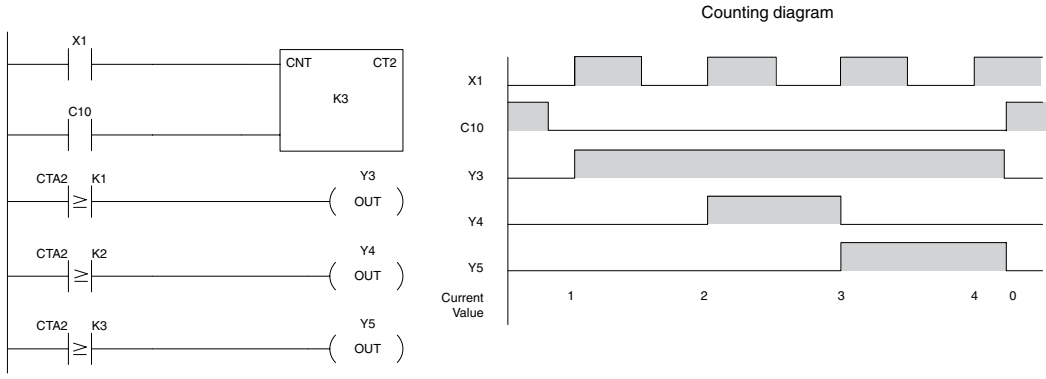
Counter Example Using Discrete Status Bits

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.



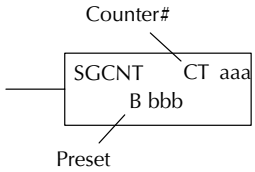
Counter Example Using Comparative Contacts

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.



Stage Counter (SGCNT)

The Stage Counter is a single input counter that increments when the input logic transitions from off to on. This counter differs from other counters since it will hold its current value until reset using the RST instruction. The Stage Counter is designed for use in 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.



Instruction Specifications

Counter Reference (CTaaa): Specifies the counter number.

Preset Value (Bbbb): Constant value (K) or a V-memory location.

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.

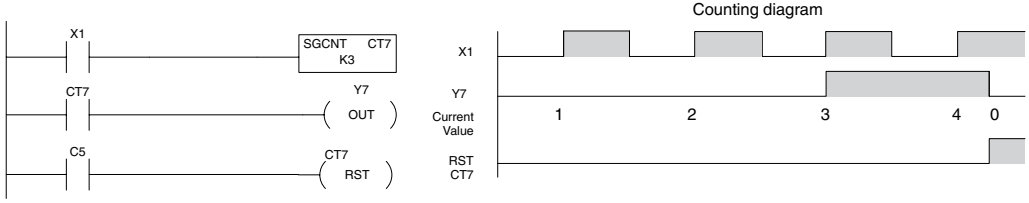
NOTE 1: *In using a counter inside a stage, the stage must be active for one scan before the input to the counter makes a 0-1 transition. Otherwise, there is no real transition and the counter will not count.*
NOTE 2: A V-memory preset is required only if the ladder program or an Operator Interface unit is used to change the preset.

Operand Data Type		D4-454 Range	
	A/B	aaa	bbb
Counters	CT	0 – 377	– –
V-memory (preset only)	V	– –	1400 – 7377 10000 – 37777
Pointers (preset only)	P	– –	1400 – 7377 10000 – 37777
Constants (preset only)	K	– –	0 – 9999
Counter discrete status bits	CT/V	0 – 377 or V41140 – V41157	
Counter current values	V /CT**	1000-1377	

NOTE: *The DirectSOFT programming software uses separate references, such as "CT2" for discrete status bit for Counter CT2, and "CTA2" for the current value of Counter CT2.*

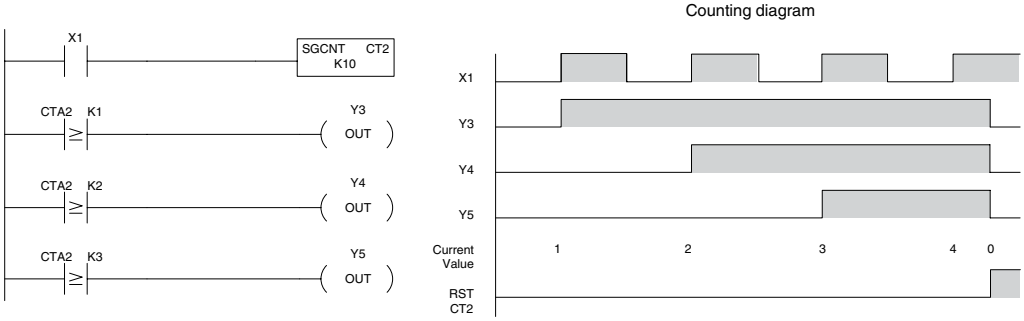
Stage Counter Example Using Discrete Status Bits

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.



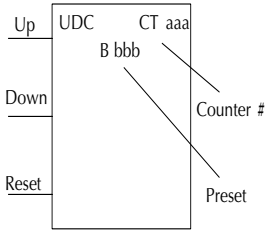
Stage Counter Example Using Comparative Contacts

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 (CTA2).



Up/Down Counter (UDC)

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



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

Instruction Specification

Counter Reference (CTaaa): Specifies the counter number.

Preset Value (Bbbb): Constant value (K) or two consecutive V-memory locations, in BCD.

Current Values: Current count is a double word value accessed by referencing the associated V or CT memory locations in BCD. 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. Operating as a "counter done bit" 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.

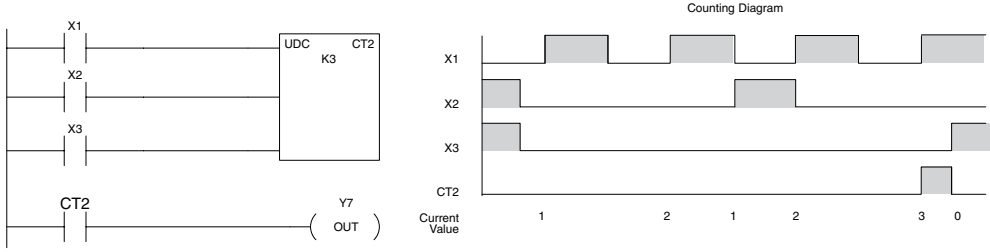
NOTE 1: *The UDC uses two consecutive V-memory locations for the 8-digit value, therefore two consecutive counter locations. For example, if UDC CT1 is used, the next available counter number is CT3.*
NOTE 2: *A V-memory preset is required only if the ladder program or an Operator Interface is used to change the preset.*

Operand Data Type		D4-454 Range	
	A/B	aaa	bbb
Counters	CT	0 – 376	– –
V-memory (preset only)	V	– –	1400 – 7377 10000 – 37777
Pointers (preset only)	P	– –	1400 – 7377 10000 – 37777
Constants (preset only)	K	– –	0 – 99999999
Counter discrete status bits	CT/V	0 – 377 or V41140 – V41157	
Counter current values	V /CT**	1000-1377	

NOTE: *The DirectSOFT programming software uses separate references, such as "CT2" for discrete status bit for Counter CT2, and "CTA2" for the current value of Counter CT2.*

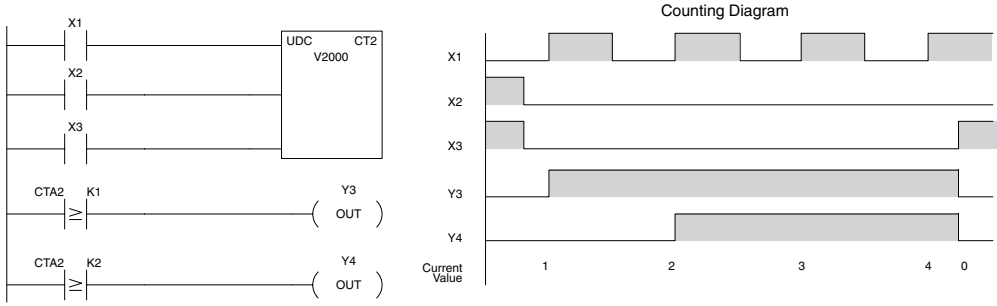
Up / Down Counter Example Using Discrete Status Bits

In the following example, if X2 and X3 are off, the counter will increment by one when X1 toggles from Off to On. 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.



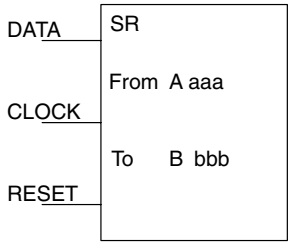
Up / Down Counter Example Using Comparative Contacts

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.



Shift Register (SR)

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

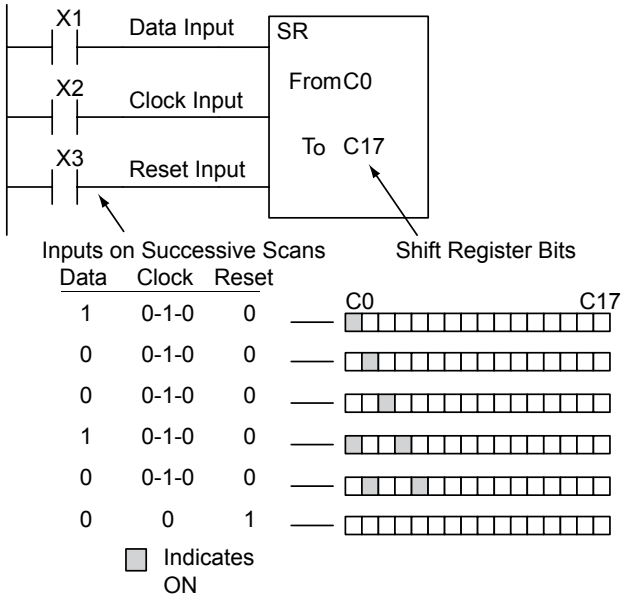


The Shift Register has three contacts.

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

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

Operand Data Type		D4-454 Range	
	A/B	aaa	bbb
Control Relay	C	0 – 3777	0 – 3777



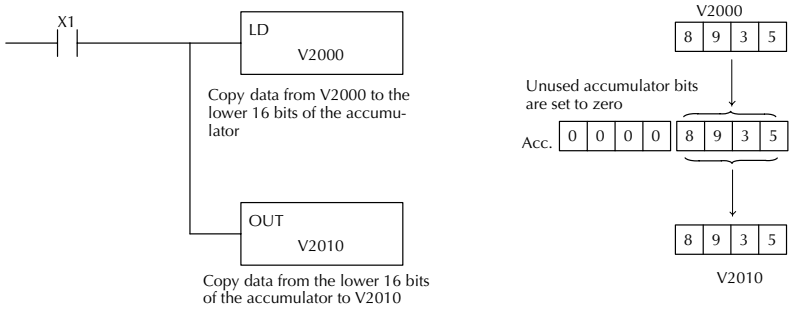
Accumulator/Stack Load and Output Data Instructions

Using the Accumulator

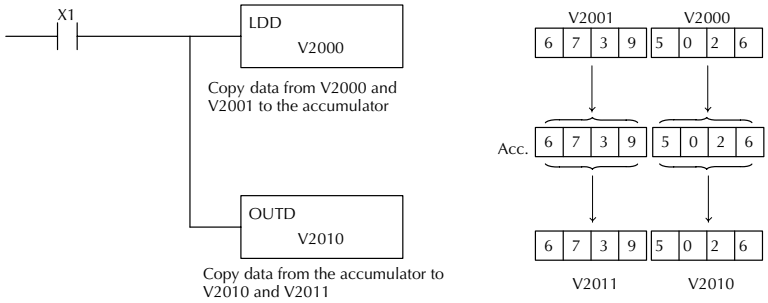
The accumulator in the D4-454 CPUs is a 32-bit register which is used as a temporary storage location for data that is being copied or manipulated in some manner. For example, you have to use the accumulator to perform math operations such as add, subtract, multiply, etc. Since there are 32 bits, you can use up to an 8-digit BCD number. 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 V2000 to V-memory location V2010.

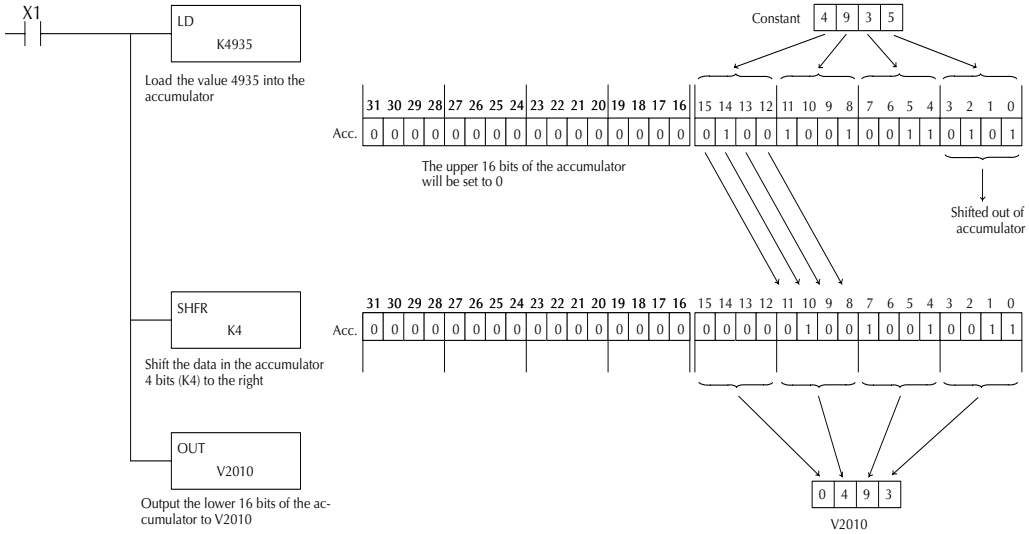


Since the accumulator is 32 bits and V-memory locations are 16 bits, the Load Double and Out Double (or variations thereof) use two consecutive V-memory locations or 8 digit BCD constants to copy data either to the accumulator from a V-memory address or from the accumulator to V-memory. For example, if you wanted to copy data from V2000 and V2001 to V2010 and V2011 the most efficient way to perform this function would be as follows:



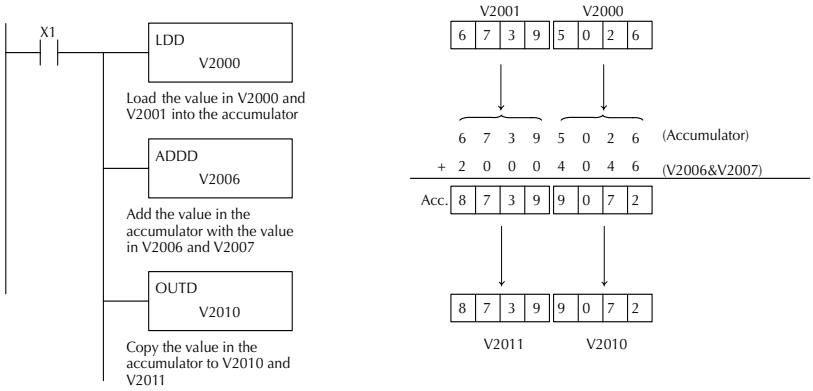
Changing the Accumulator Data

Instructions that manipulate data also use the accumulator. The result of the manipulated data resides in the accumulator. The data that was being manipulated is cleared from the accumulator. The following example loads the constant value 4935 into the accumulator, shifts the data right 4 bits, and outputs the result to V2010.



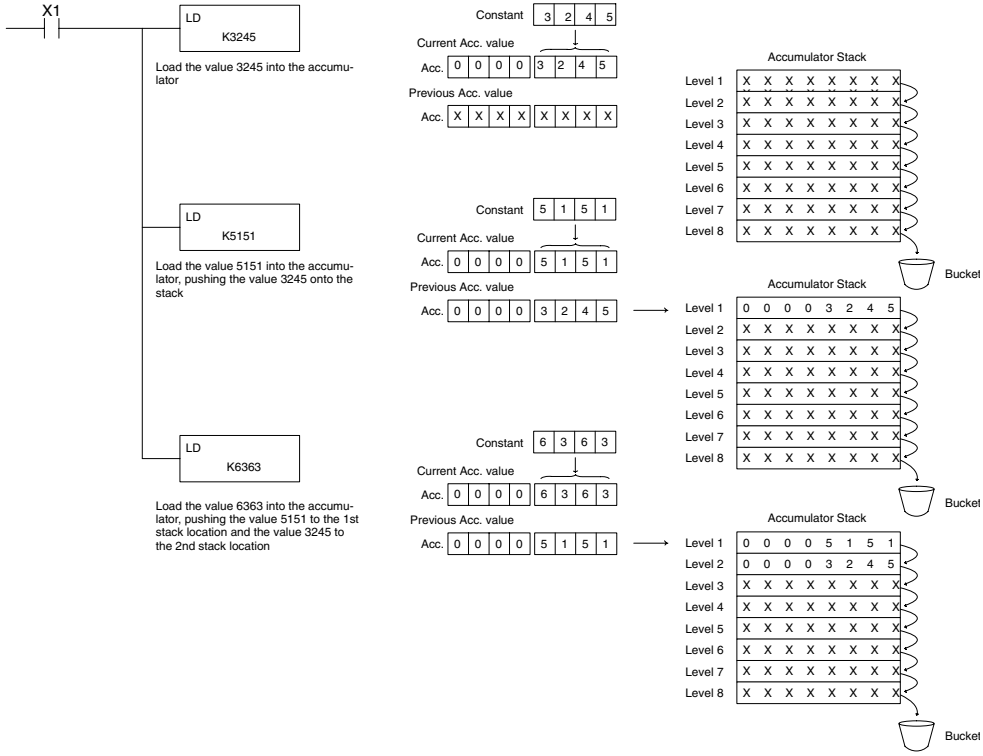
Some of the data manipulation instructions use 32 bits. They use two consecutive V-memory locations or an 8 digit BCD constant to manipulate data in the accumulator.

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

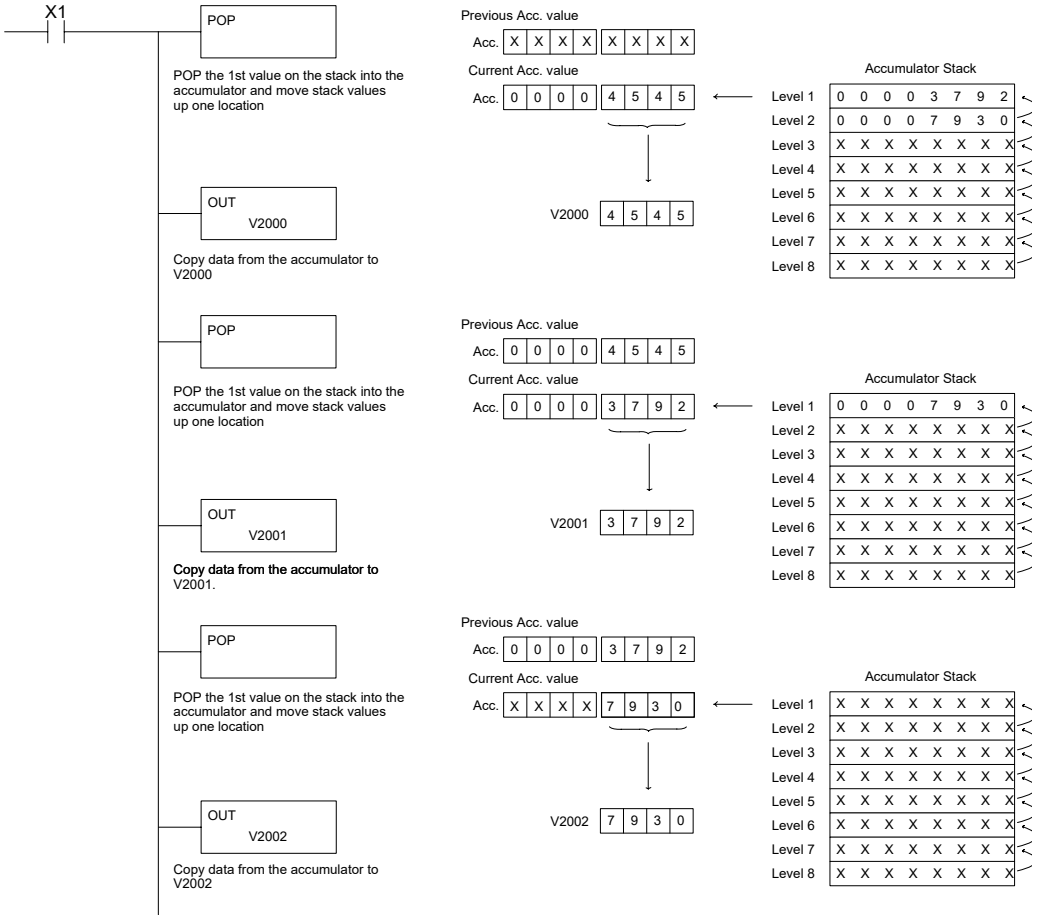


Using the Accumulator Stack

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

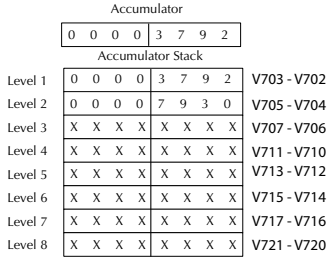


The POP instruction rotates values upward through the stack into the accumulator. When a POP is executed, the value which was in the accumulator is cleared and the value that was on top of the stack is in the accumulator. The values in the stack are shifted up one position in the stack.



Accumulator and Accumulator Stack Memory Locations

There may be times when you want to read a value that been placed onto the accumulator stack without having to pop the stack first. Both the accumulator and the accumulator stack have corresponding V-memory locations that can be accessed by the program. You cannot write to these locations, but you can read them or use them in comparative boolean instructions, etc.

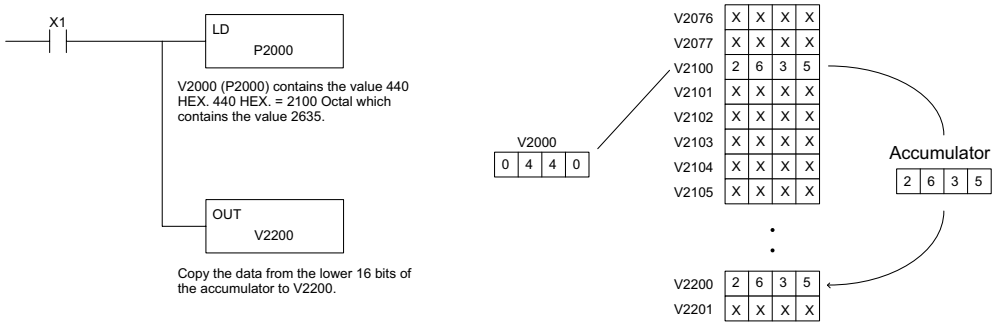


Using Pointers

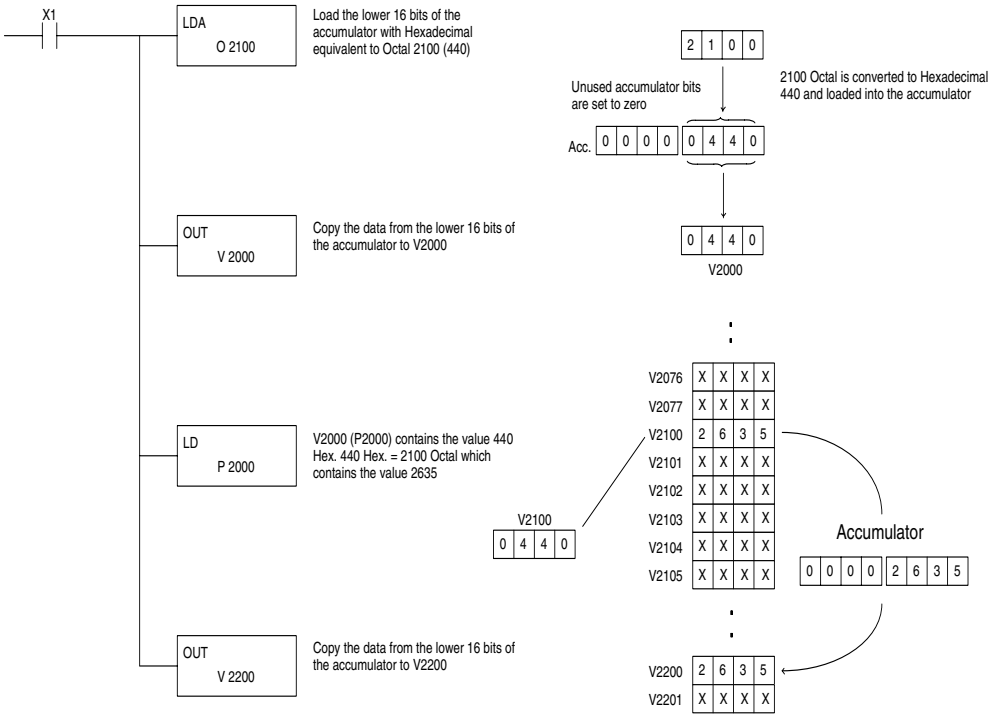
Many of the D4-454 series instructions will allow V-memory pointers as operands commonly known as indirect addressing. Pointers allow instructions to obtain data from V-memory locations referenced by the pointer value.

NOTE: *D4-454 V-memory addressing is in octal. However, the pointers reference a V-memory location with values viewed as HEX. Use the Load Address (LDA) instruction to move an address into the pointer location. This instruction performs the Octal to Hexadecimal conversion automatically.*

In the following example we are using a pointer operand in a Load instruction. V-memory location 2000 is being used as the pointer location. V2000 contains the value 440 which the CPU views as the Hex equivalent of the Octal address V-memory location V2100. The CPU will copy the data from V2100, which (in this example) contains the value 2635, into the lower word of the accumulator.

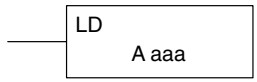


The following example is identical to the one on the previous page, with one exception. The LDA (Load Address) instruction automatically converts the Octal address to Hex.



Load (LD)

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

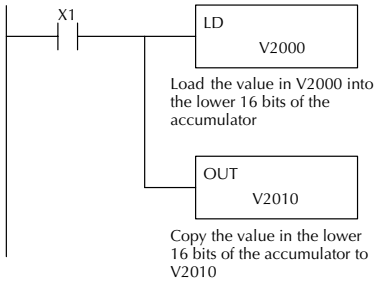


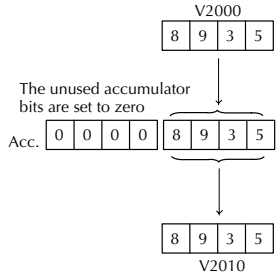
Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	K	0 – FFFF

Discrete Bit Flags	Description
SP70	On anytime the value in the accumulator is negative.
SP76	On when any instruction loads a value of zero into the accumulator.

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

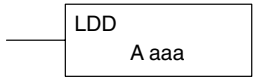
In the following example, when X1 is on, the value in V2000 will be loaded into the accumulator and output to V2010.





Load Double (LDD)

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

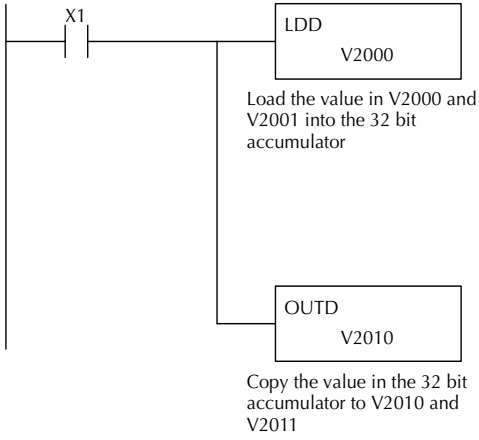


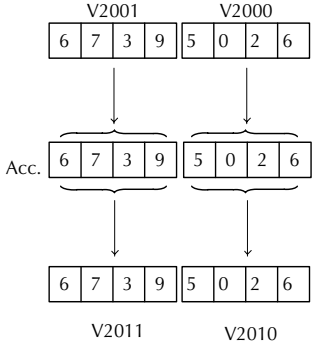
Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	K	0 – FFFFFFFF

Discrete Bit Flags	Description
SP70	On anytime the value in the accumulator is negative.
SP76	On when any instruction loads a value of zero into the accumulator.

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.





Load Formatted (LDF)

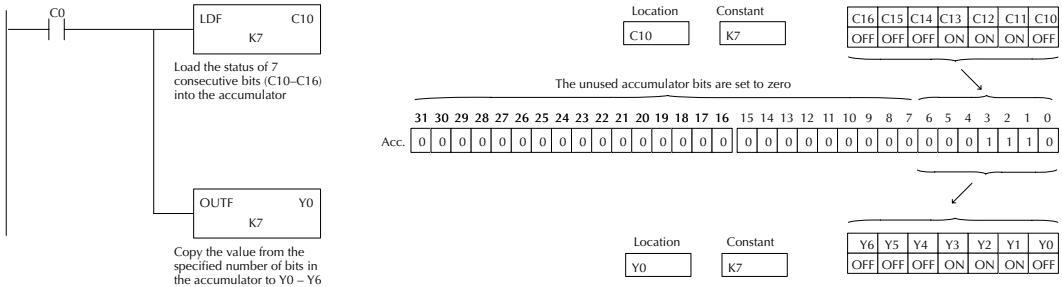
The Load Formatted instruction loads 1–32 consecutive bits from discrete memory locations into the accumulator. The instruction requires a starting location (Aaaa) and the number of bits (Kbbb) to be loaded. Unused accumulator bit locations are set to zero.

Operand Data Type		D4-454 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 Relays	SP	0–777	– –
Global I/O	GX	0–3777	
Constant	K	– –	1–32

Discrete Bit Flags	Description
SP70	On anytime the value in the accumulator is negative.
SP76	On when any instruction loads a value of zero into the accumulator.

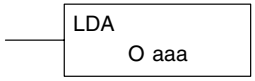
NOTE: *Two consecutive Load instructions will place the value of the first load instruction onto the accumulator stack.*

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



Load Address (LDA)

The Load Address instruction is a 16-bit instruction. It converts any octal value or address to the HEX equivalent value and loads the HEX value into the accumulator. This instruction is useful when an address parameter is required, since all addresses for the D4-454 system are in octal.

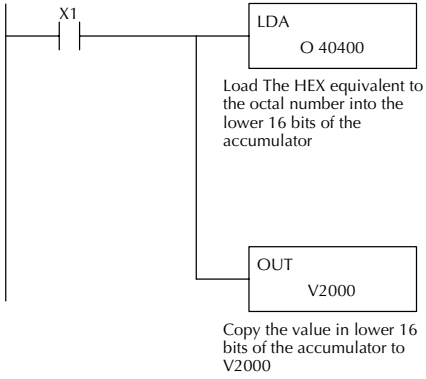


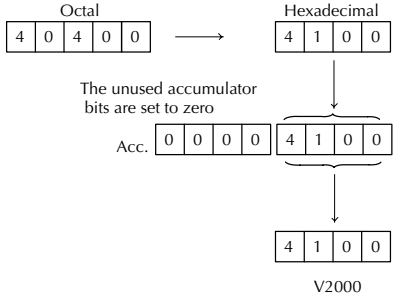
Operand Data Type		D4-454 Range
		aaa
Octal Address	O	1400 – 7377 10000 – 36777

Discrete Bit Flags	Description
SP70	On anytime the value in the accumulator is negative.
SP76	On when any instruction loads a value of zero into the accumulator.

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

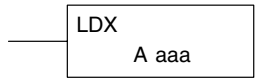
In the following example, when X1 is on, the octal number 40400 will be converted to a HEX 4100 and loaded into the accumulator using the Load Address instruction. The value in the lower 16 bits of the accumulator is copied to V2000 using the Out instruction.





Load Accumulator Indexed (LDX)

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



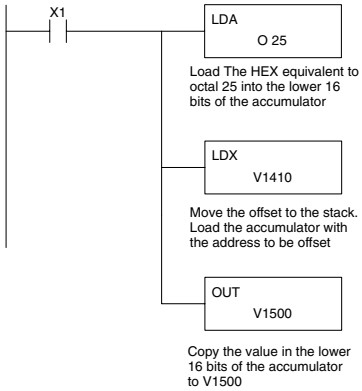
Helpful Hint: The Load Address instruction can be used to convert an octal address to a HEX address and load the value into the accumulator.

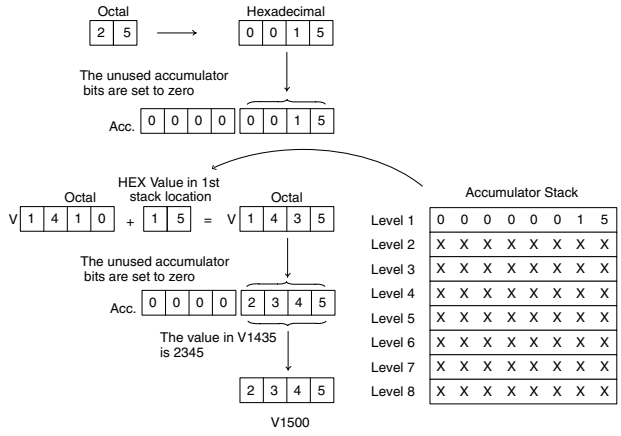
Operand Data Type		D4-454 Range	
A		aaa	aaa
V-memory	V	See memory map	See memory map
Pointer	P	See memory map	See memory map

Discrete Bit Flags	Description
SP70	On anytime the value in the accumulator is negative.
SP76	On when any instruction loads a value of zero into the accumulator.

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

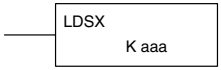
In the following example, when X1 is on, the HEX equivalent for octal 25 will be loaded into the accumulator (this value will be placed on the stack when the Load Accumulator Indexed instruction is executed). V-memory location V1410 will be added to the value in the first level of the stack and the value 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)

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



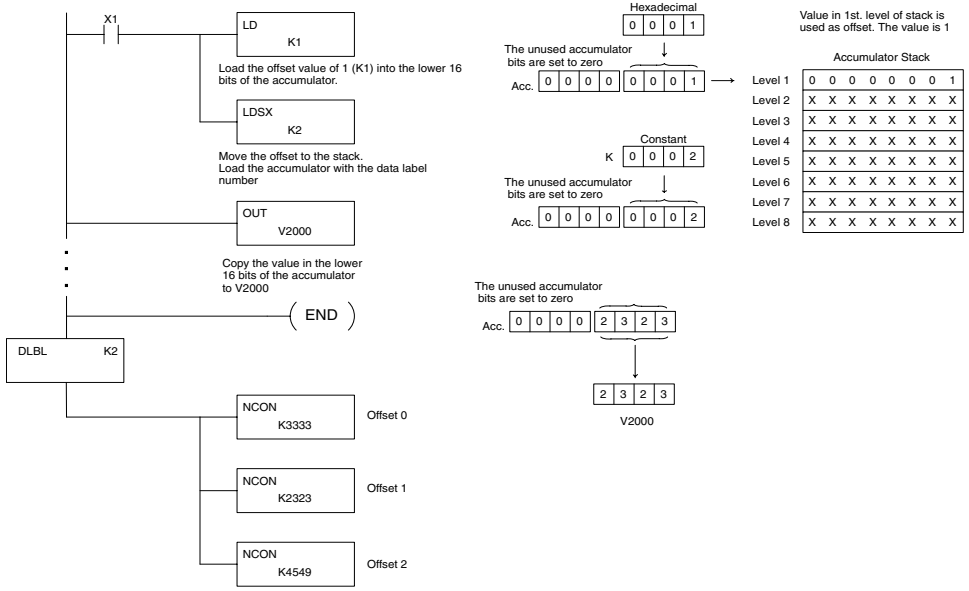
The LDSX instruction uses the value in the first level of the accumulator stack as an offset to determine which numerical or ASCII constant within the Data Label Area will be loaded into the accumulator. The LDSX instruction interprets the value in the first level of the accumulator stack as a HEX value.

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

Operand Data Type		D4-454 Range
		aaa
Constant	K	1-FFFF

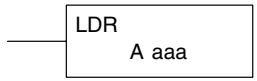
NOTE: *Two consecutive Load instructions will place the value of the first load instruction onto the accumulator stack.*

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



Load Real Number (LDR)

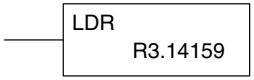
The Load Real Number instruction loads a real number contained in two consecutive V-memory locations, or an 8-digit constant into the accumulator.



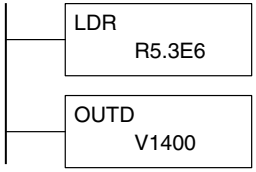
Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Real Constant	R	$-3.402823E^{+38}$ to + $+3.402823E^{+38}$

Discrete Bit Flags	Description
SP70	On anytime the value in the accumulator is negative.
SP76	On when any instruction loads a value of zero into the accumulator.

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".

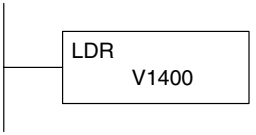


For very large numbers or very small numbers, you can use exponential notation. The number to the right is 5.3 million. The OUTD instruction stores it in V1400 and V1401.



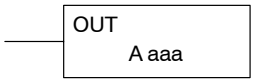
These real numbers are in the IEEE 32-bit floating point format, so they occupy two V-memory locations, regardless of how big or small the number may be! If you view a stored real number in hex, binary, or even BCD, the number shown will be very difficult to decipher. Just like all other number types,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

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



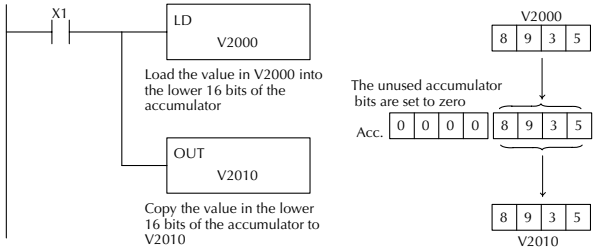
Out (OUT)

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



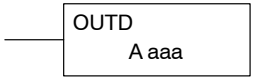
Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map

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.



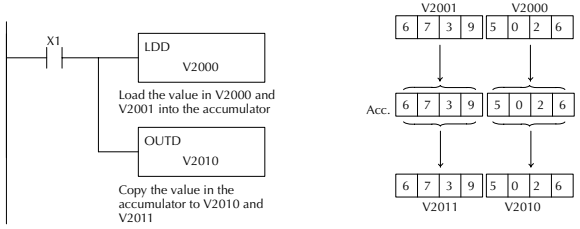
Out Double (OUTD)

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



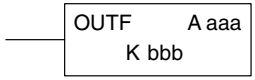
Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map

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 OUTD instruction.



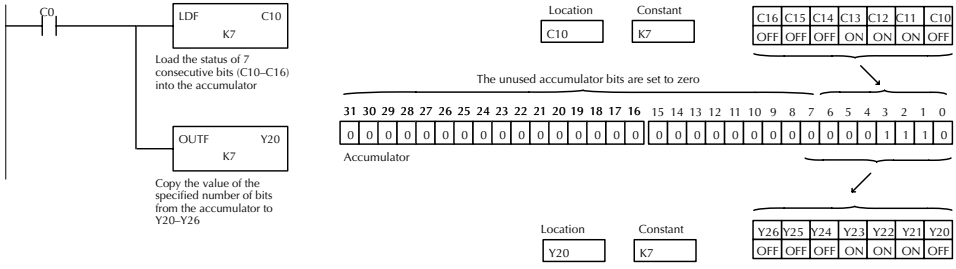
Out Formatted (OUTF)

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



Operand Data Type	D4-454 Range		
	A	aaa	bbb
Inputs	X	0 – 1777	– –
Outputs	Y	0 – 1777	– –
Control Relays	C	0 – 3777	– –
Constant	K	– –	1 – 32

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

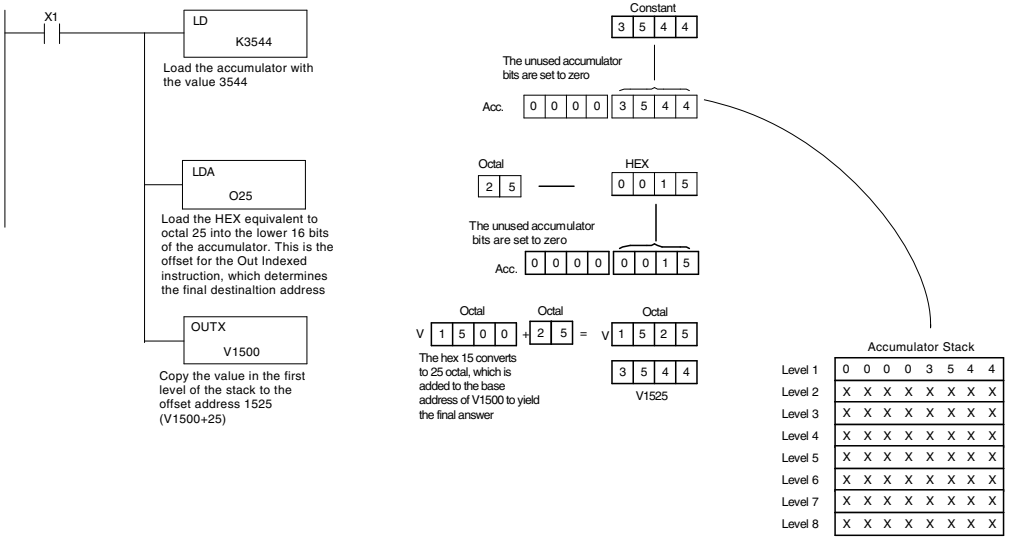


Out Indexed (OUTX)

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

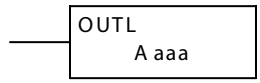
Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map

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



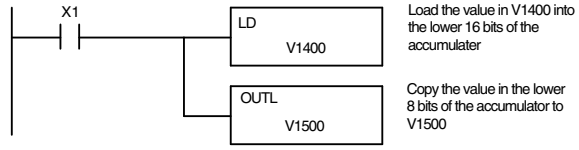
Out Least (OUTL)

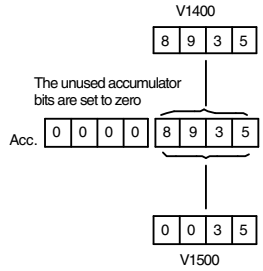
The OUTL instruction copies the value in the lower eight bits of the accumulator to the lower eight bits of the specified V-memory location (i.e., it copies the low byte of the low word of the accumulator).



Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map

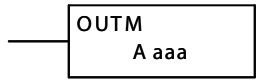
In the following example, when X1 is on, the value in V1400 will be loaded into the lower 16 bits of the accumulator using the LD instruction. The value in the lower 8 bits of the accumulator is copied to V1500 using the OUTL instruction.





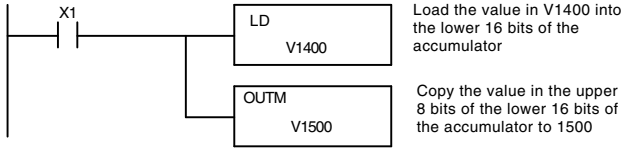
Out Most (OUTM)

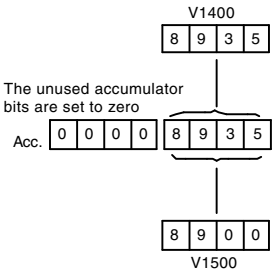
The OUTM instruction copies the value in the upper eight bits of the lower sixteen bits of the accumulator to the upper eight bits of the specified V-memory location (i.e., it copies the high byte of the low word of the accumulator).



Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map

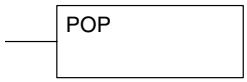
In the following example, when X1 is on, the value in V1400 will be loaded into the lower 16 bits of the accumulator using the LD instruction. The value in the upper 8 bits of the lower 16 bits of the accumulator is copied to V1500 using the OUTM instruction.





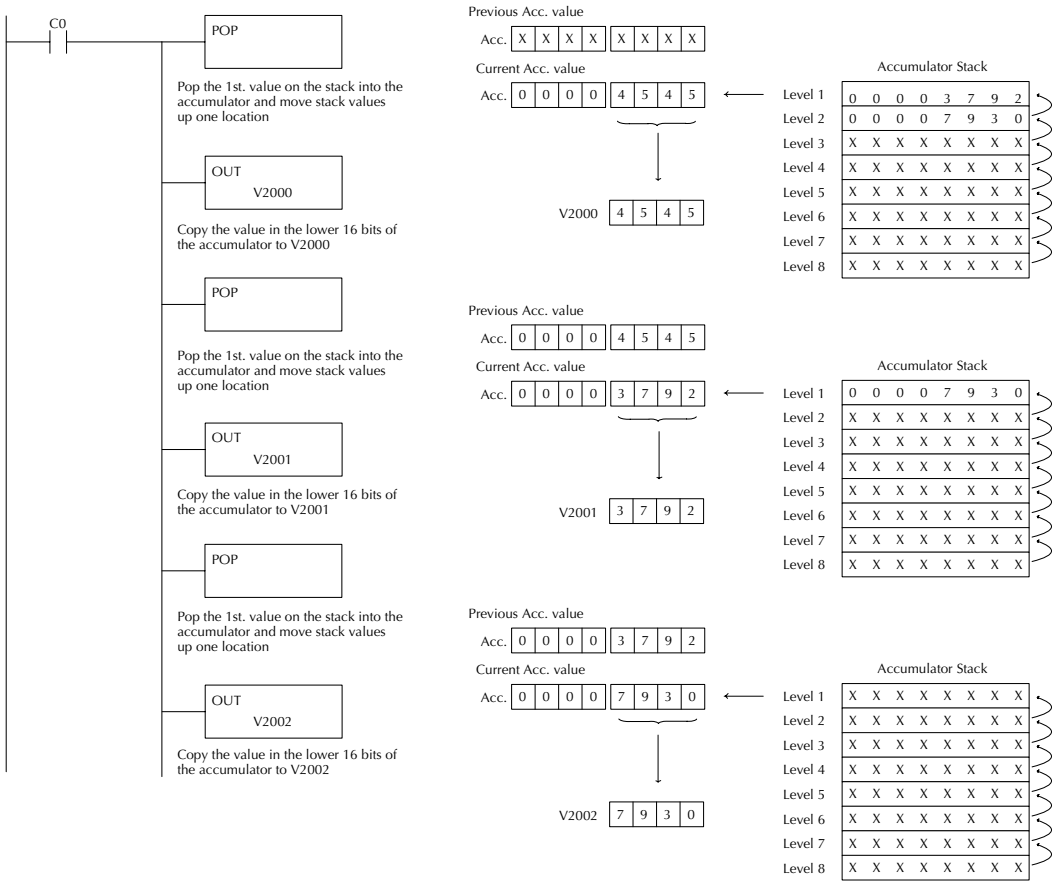
Pop (POP)

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



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

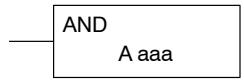
In the example below, when C0 is on, the value 4545 that was on top of the stack is moved into the accumulator using the Pop instruction The value is output to V2000 using the OUT instruction. The next Pop moves the value 3792 into the accumulator and outputs the value to V2001. The last Pop moves the 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 OUTD instruction would be used and 2 V-memory locations for each OUTD must be allocated.



Logical Instructions (Accumulator)

And (AND logical)

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.

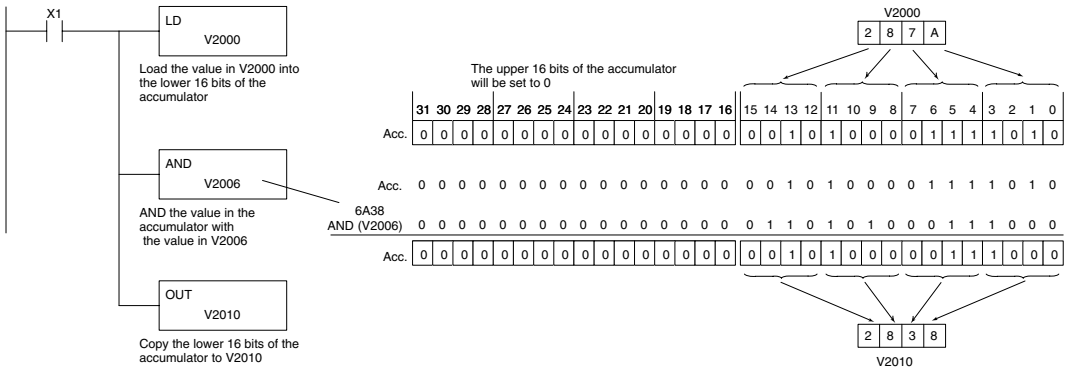


Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map

Discrete Bit Flags	Description
SP63	ON if the result in the accumulator is zero.
SP76	ON when the value loaded into the accumulator by any instruction is zero.

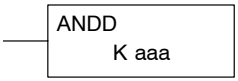
NOTE: *Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V2000 will be loaded into the accumulator using the LD 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.



And Double (ANDD)

ANDD 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 ANDD is zero or a negative number (the most significant bit is on).

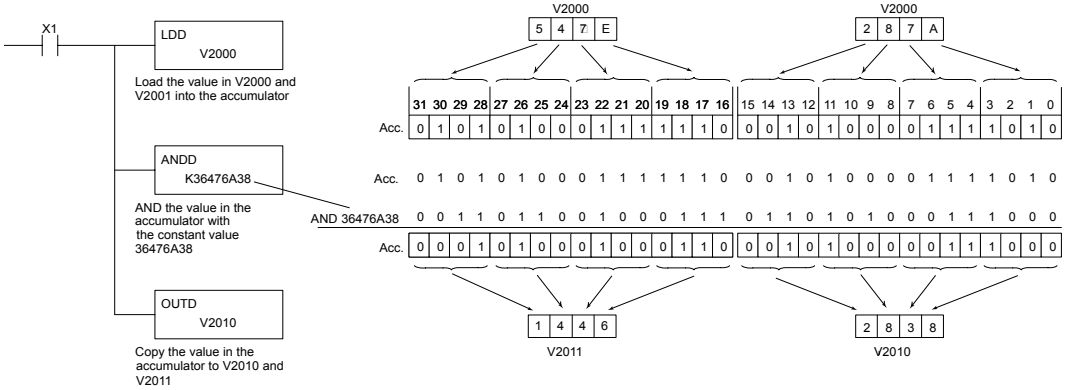


Operand Data Type		D4-454 Range
		aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	K	0 – FFFFFFFF

Discrete Bit Flags	Description
SP63	ON if the result in the accumulator is zero.
SP76	ON when the value loaded into the accumulator by any instruction is zero.

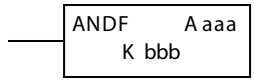
NOTE: *Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V2000 and V2001 will be loaded into the accumulator using the LDD instruction. The value in the accumulator is ANDed with 36476A38 using the ANDD instruction. The value in the accumulator is output to V2010 and V2011 using the OUTD instruction.



And Formatted (ANDF)

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

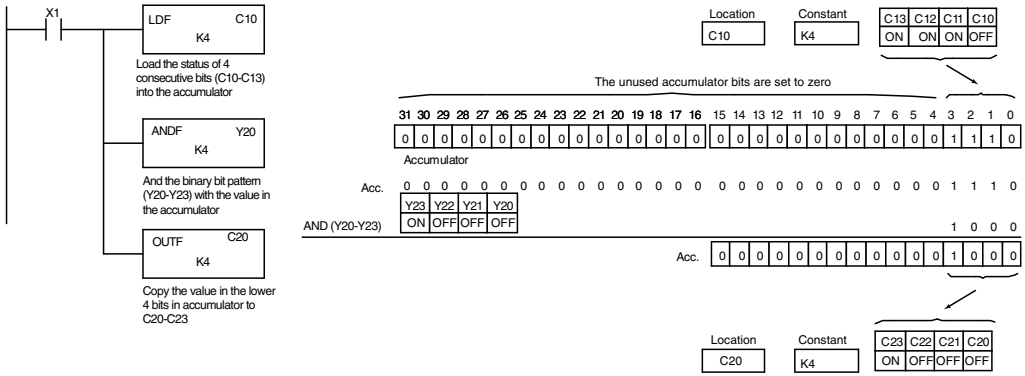


Operand Data Type		D4-454 Range	
	B	aaa	bbb
Inputs	X	0 – 1777	-
Outputs	Y	0 – 1777	-
Control Relays	C	0 – 3777	-
Stage Bits	S	0 – 1777	-
Timer Bits	T	0 – 377	-
Counter Bits	CT	0 – 377	-
Special Relays	SP	0 – 777	-
Constant	K	-	1-32

Discrete Bit Flags	Description
SP63	ON if the result in the accumulator is zero.
SP70	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 LDF instruction loads C10 – C13 (4 binary bits) into the accumulator. The accumulator contents is logically ANDed with the bit pattern from Y20 – Y23 using the ANDF instruction. The OUTF instruction outputs the accumulator's lower four bits to C20 – C23.



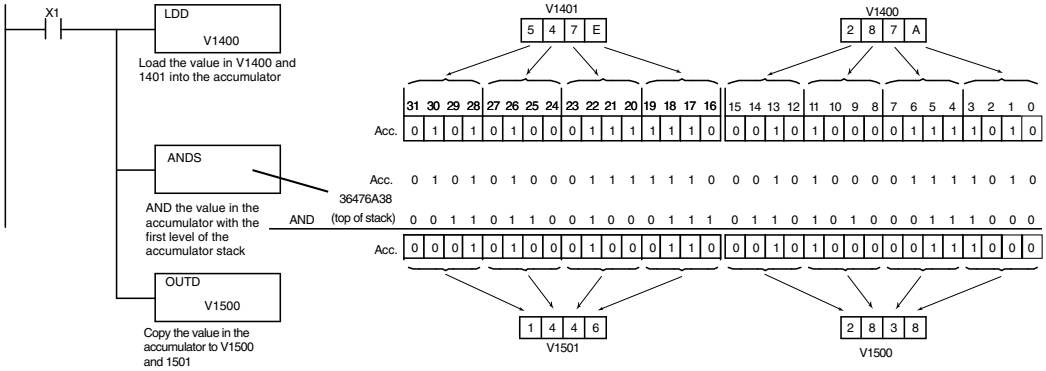
And with Stack (ANDS)

The ANDS 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 ANDS is zero or a negative number (the most significant bit is on

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

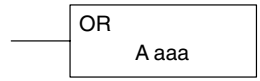
NOTE: *Status flags are valid only until another instruction uses the same flag.*

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 or the accumulator stack. The result resides in the accumulator. The 32-bit value is then output to V1500 and V1501.



Or (OR)

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

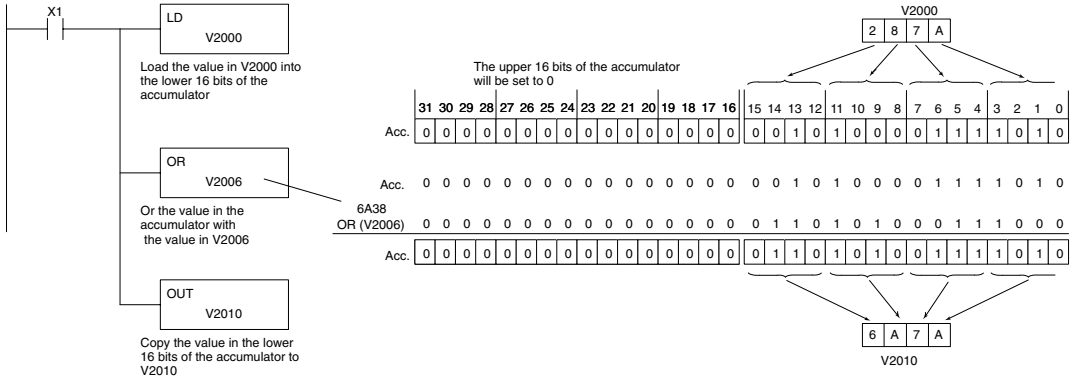


Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map

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

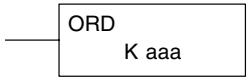
NOTE: *Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in 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 is output to V2010 using the Out instruction.



Or Double (ORD)

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

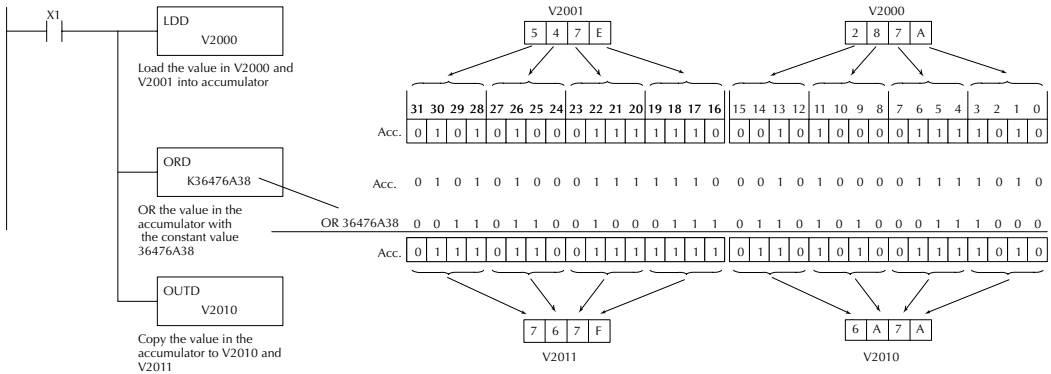


Operand Data Type		D4-454 Range
		aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	K	0 – FFFFFFFF

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

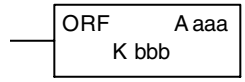
NOTE: *Status flags are valid only until another instruction uses the same flag.*

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



Or Formatted (ORF)

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

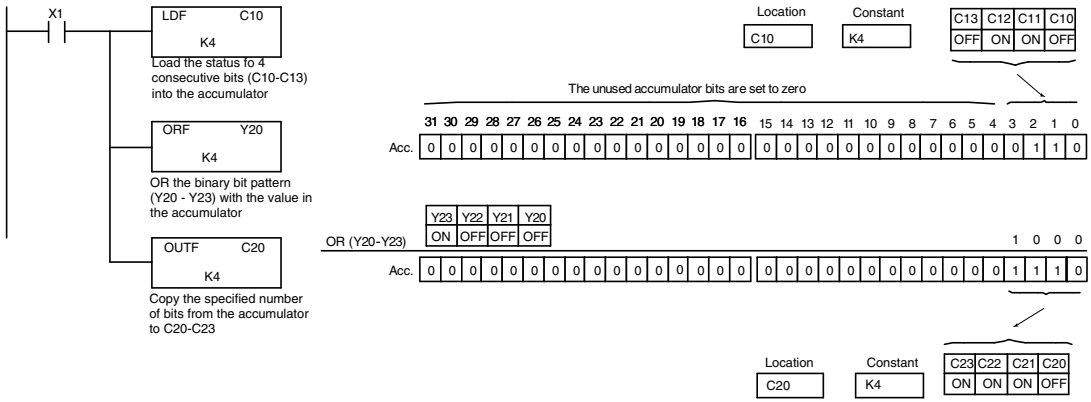


Operand Data Type		D4-454 Range	
	A/B	aaa	bbb
Inputs	X	0 – 1777	--
Outputs	Y	0 – 1777	--
Control Relays	C	0 – 3777	--
Stage Bits	S	0 – 1777	--
Timer Bits	T	0 – 377	--
Counter Bits	CT	0 – 377	--
Special Relays	SP	0 – 777	--
Constant	K	-	1-32

Discrete Bit Flags	Description
SP63	ON if the result in the accumulator is zero.
SP70	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 LDF instruction loads C10 – C13 (4 binary bits) into the accumulator. The ORF instruction logically ORs the accumulator contents with Y20 – Y23 bit pattern. The ORF instruction outputs the accumulator's lower four bits to C20 – C23.

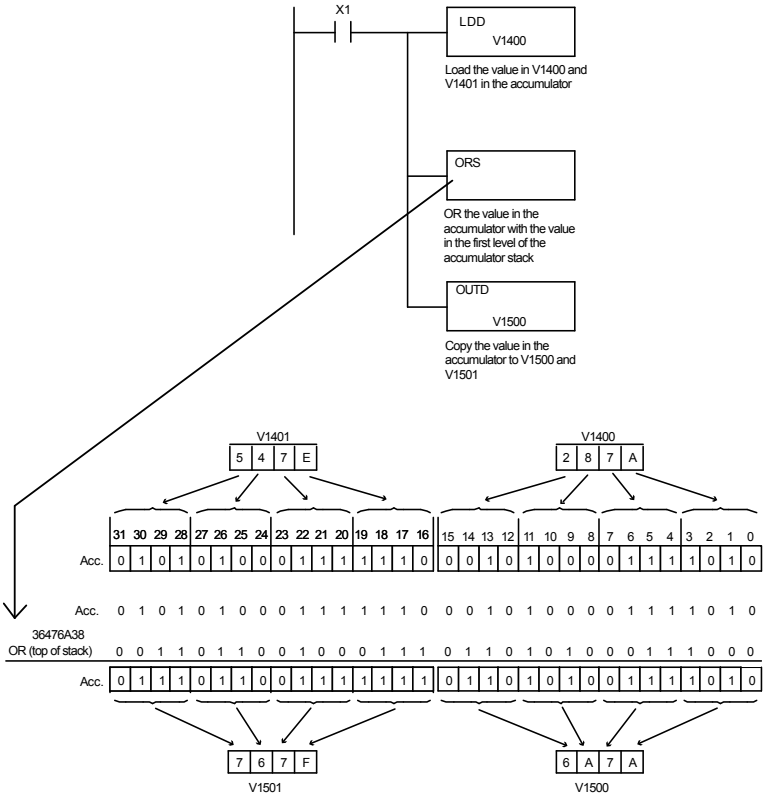


Or with Stack (ORS)

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

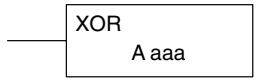
Discrete Bit Flags	Description
SP63	ON if the result in the accumulator is zero.
SP70	ON if the result in the accumulator is negative.

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



Exclusive Or (XOR)

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

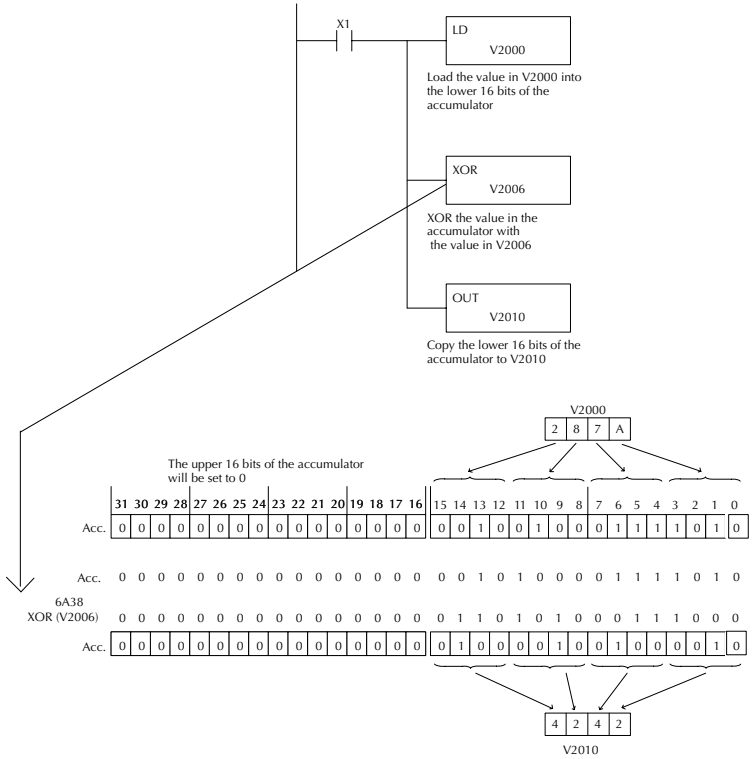


Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map

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

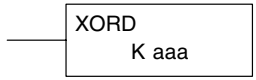
NOTE: *Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V2000 will be loaded into the accumulator using the LD instruction. The value in the accumulator is exclusive ORed with V2006 using the XOR instruction. The value in the lower 16 bits of the accumulator is output to V2010 using the OUT instruction.



Exclusive Or Double (XORD)

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

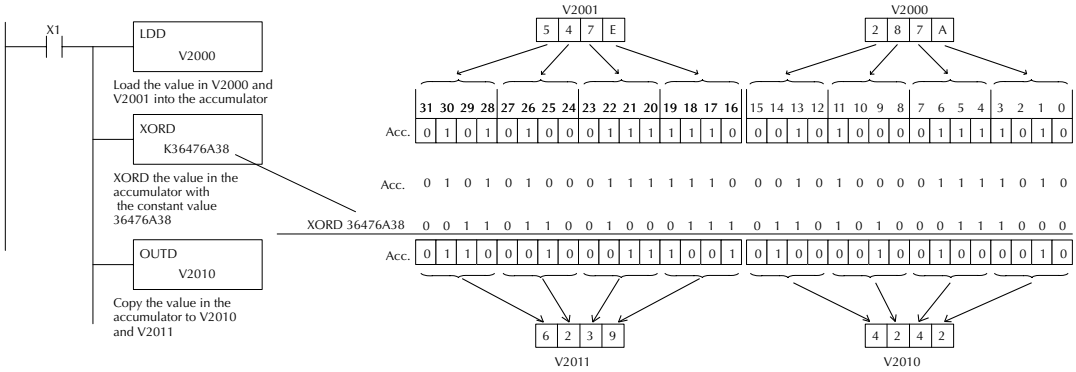


Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	K	0 – FFFFFFFF

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

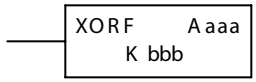
NOTE: *Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V2000 and V2001 will be loaded into the accumulator using the LDD instruction. The value in the accumulator is exclusively ORed with 36476A38 using the XORD instruction. The value in the accumulator is output to V2010 and V2011 using the OUTD instruction.



Exclusive Or Formatted (XORF)

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



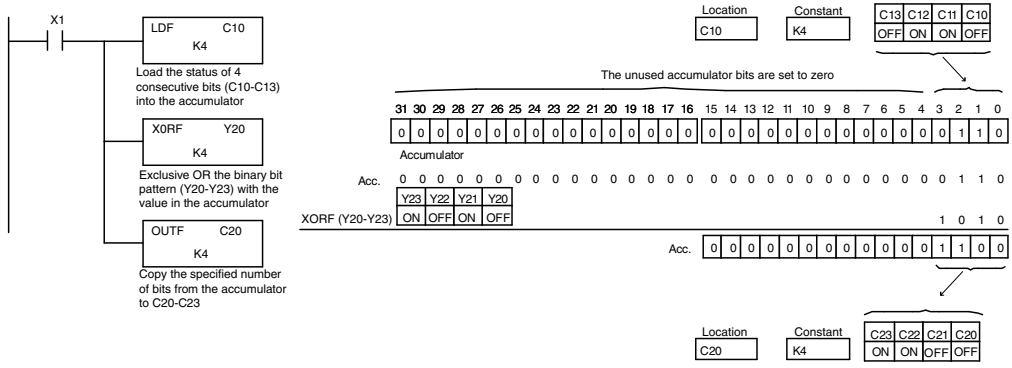
The instruction requires a starting location (Aaaa) and the number of bits (Bbbb) to be exclusive ORed. Discrete status flags indicate if the result of the XORF is zero or negative (the most significant bit =1).

Operand Data Type	A/B	D4-454 Range	
		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 Relays	SP	0 – 777	-
Global I/O GX		0 – 3777	
Constant	K	-	1-32

Discrete Bit Flags	Description
SP63	ON if the result in the accumulator is zero.
SP70	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 LDF instruction. The value in the accumulator will be logically exclusive ORed with the bit pattern from Y20 – Y23 using the XORF instruction. The value in the lower 4 bits of the accumulator is output to C20 – C23 using the OUTF instruction.



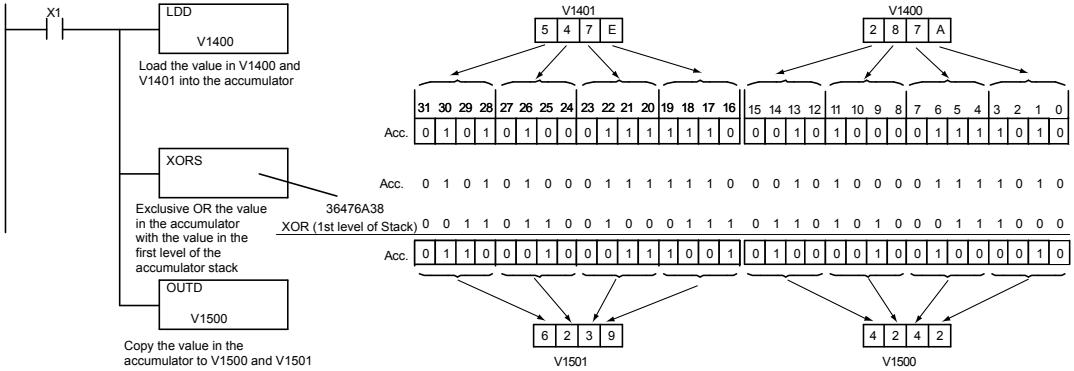
Exclusive Or with Stack (XORS)

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

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

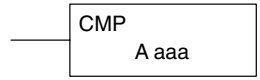
NOTE: *Status flags are valid only until another instruction uses the same flag.*

In the following example, when X1 is on, the value in V1400 and V1401 will be loaded into the accumulator using the LDD instruction. The binary value in the accumulator will be exclusively OR'd with 36476A38 using the XORS instruction. The value in the accumulator is output to V1500 and V1501 using the OUTD instruction.



Compare (CMP)

The CMP 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. The data format for this instruction is BCD/Hex, Decimal and Binary.

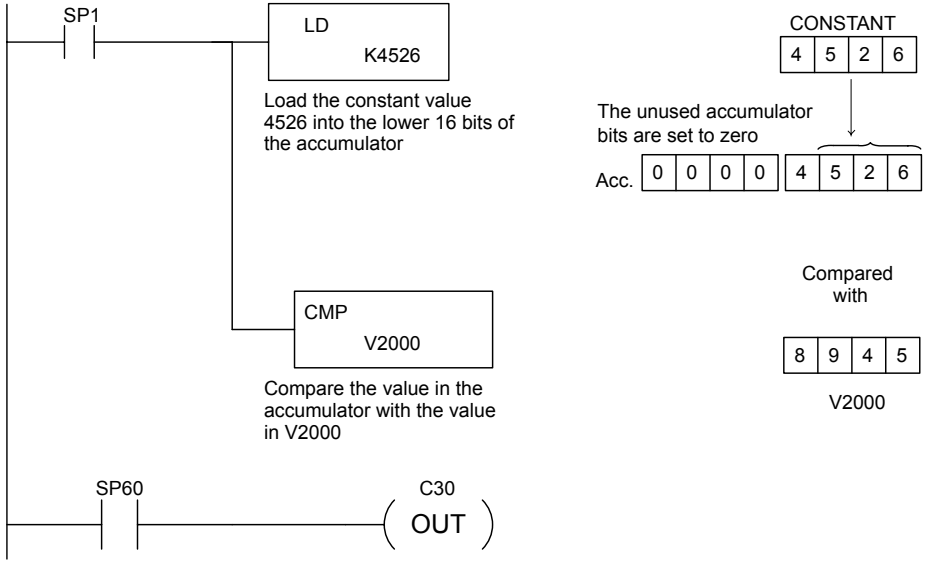


Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map

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.

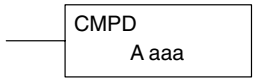
NOTE: *Status flags are valid only until another instruction uses the same flag.*

In the following example when SP1 is on, the constant 4526 will be loaded into the lower 16 bits of the accumulator using the LD instruction. The value in the accumulator is compared with the value in V2000 using the CMP 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 CMP instruction, SP60 will turn on, energizing C30.



Compare Double (CMPD)

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

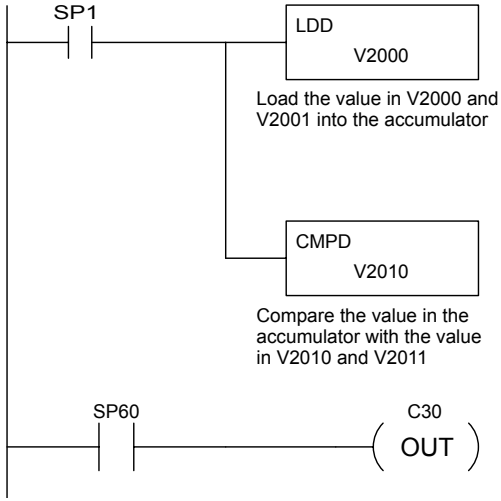


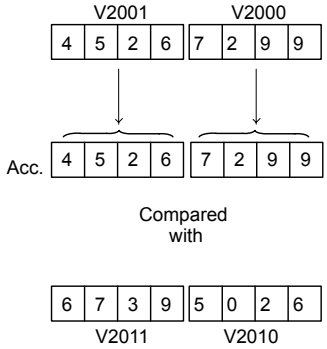
Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	K	0 – FFFFFFFF

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

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

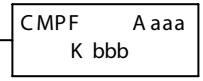
In the following example, when SP1 is on, the value in V2000 and V2001 will be loaded into the accumulator using the Load Double instruction. The value in the accumulator is compared with the value in V2010 and V2011 using the CMPD instruction. The corresponding discrete status flag will be turned on, indicating the result of the comparison. In this example, if the value in the accumulator is less than the value specified in the Compare instruction, SP60 will turn on energizing C30.





Compare Formatted (CMPF)

The Compare Formatted instruction 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. The data format for this instruction is BCD/Hex, Decimal and Binary.

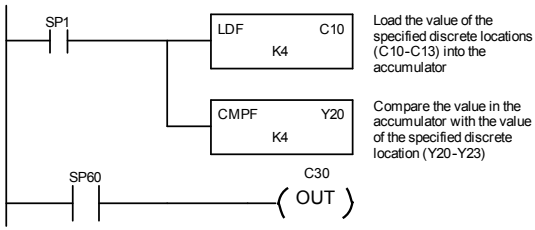


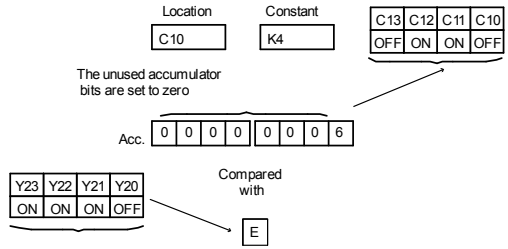
Operand Data Type		D4-454 Range	
	A/B	aaa	bbb
Inputs	X	0 – 1777	-
Outputs	Y	0 – 1777	-
Control Relays	C	0 – 3777	-
Stage Bits	S	0 – 1777	-
Timer Bits	T	0 – 377	-
Counter Bits	CT	0 – 377	-
Special Relays	SP	0 – 777	-
Global I/O	GX	0 – 3777	
Constant	K	-	1-32

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

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

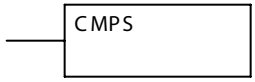
In the following example, when SP1 is on, the Load Formatted instruction loads the binary value (6) from C10 – C13 into the accumulator. The CMPF instruction compares the value in the accumulator to the value in Y20 – Y23 (E hex). The corresponding discrete status flag will be turned on, indicating the 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)

The Compare with Stack instruction is a 32-bit instruction that compares the value in the accumulator with the value in the first level of the accumulator stack. The data format for this instruction is BCD/Hex, Decimal and Binary.

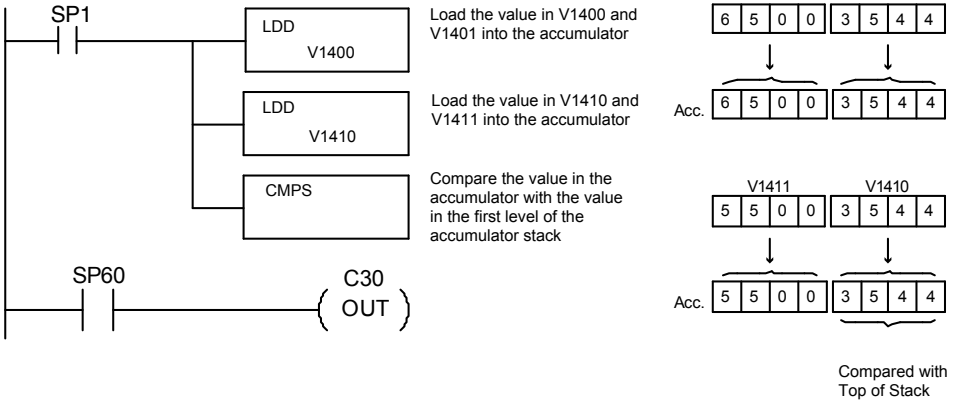


The corresponding status flag will be turned on, indicating the result of the comparison. This does not affect the value in the accumulator.

Discrete Bit Flags	Description
SP60	On when the value in the accumulator is less than the 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

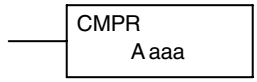
NOTE: *Status flags are valid only until another instruction uses the same flag.*

In the following example, when SP1 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.



Compare Real Number (CMPR)

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

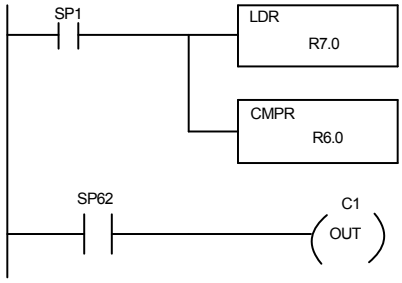


Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	R	-3.402823E+38 to + +3.402823E+38

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

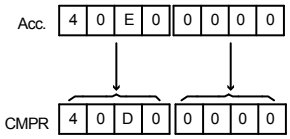
NOTE: *Status flags are valid only until another instruction uses the same flag.*

In the following example, when SP1 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), turning on control relay C1.



Load the real number representation for decimal 7 into the accumulator

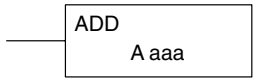
Compare the value with the real number representation for decimal 6



Math Instructions

Add (ADD)

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

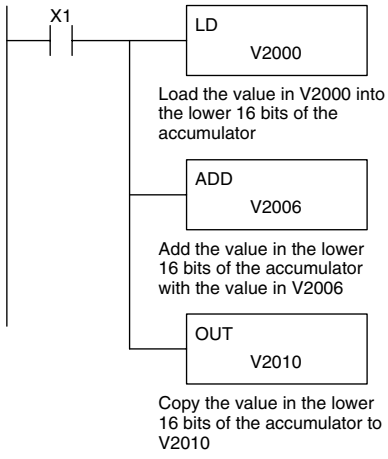


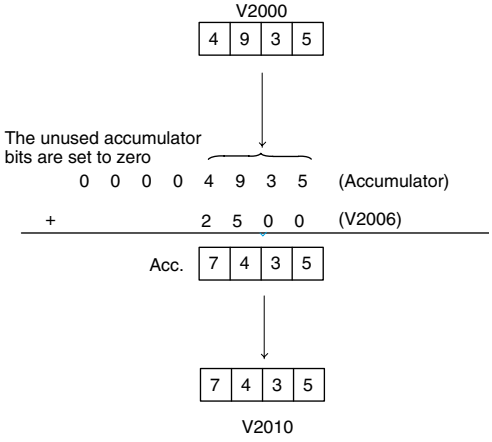
Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map

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 was encountered.

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

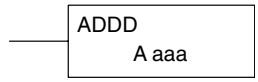
In the following example, when X1 is on, the value in V2000 will be loaded into the accumulator using the Load instruction. The value in the lower 16 bits of the accumulator is added to the value in V2006 using the Add instruction. The value in the accumulator is copied to V2010 using the Out instruction.





Add Double (ADDD)

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

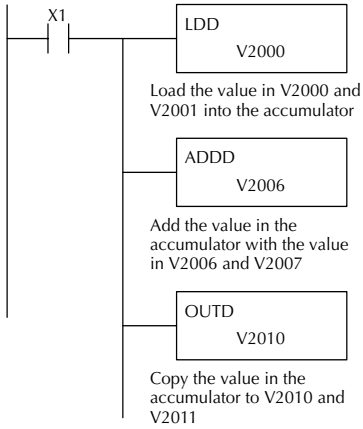


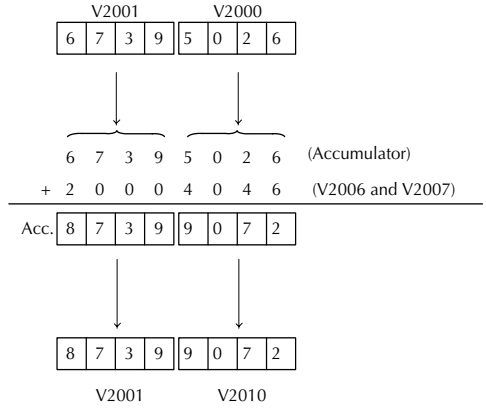
Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	K	0 – 99999999

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

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

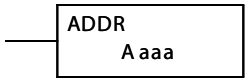
In the following example, when X1 is on, the value in V2000 and V2001 will be loaded into the accumulator using the Load Double instruction. The value in the accumulator is added with the value in V2006 and V2007 using the Add Double instruction. The value in the accumulator is copied to V2010 and V2011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





Add Real (ADDR)

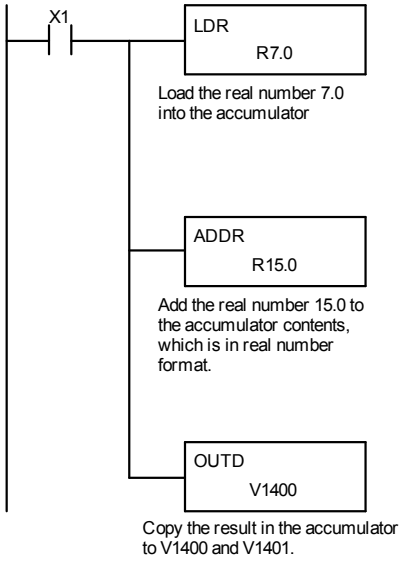
The Add Real instruction adds a real number in the accumulator with either a real constant or a real number occupying two consecutive V-memory locations. The result resides in the accumulator. Both numbers must be Real data type (IEEE floating point format).



Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	R	$-3.402823E^{+38}$ to + $+3.402823E^{+38}$

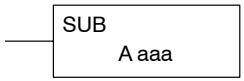
Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.
SP71	On anytime the V-memory specified by a pointer (P) is not valid.
SP72	On anytime the value in the accumulator is an 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.

NOTE 1: *Status flags are valid only until another instruction uses the same flag.*
NOTE 2: The current HPP does not support real number entry with automatic conversion to the 32-bit IEEE format. You must use DirectSOFT programming software for this feature.



Subtract (SUB)

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

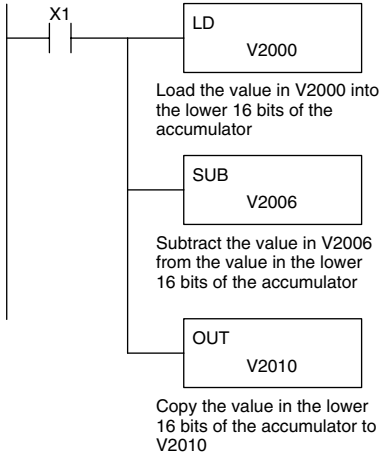


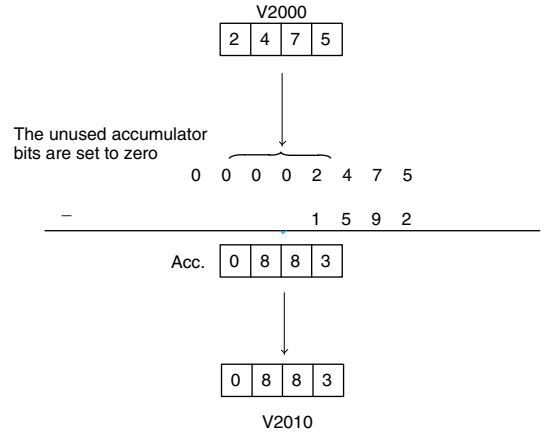
Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map

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
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.*

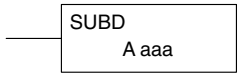
In the following example, when X1 is on, the value in V2000 will be loaded into the accumulator using the Load instruction. The value in V2006 is subtracted from the value in the accumulator using the Subtract instruction. The value in the accumulator is copied to V2010 using the Out instruction.





Subtract Double (SUBD)

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

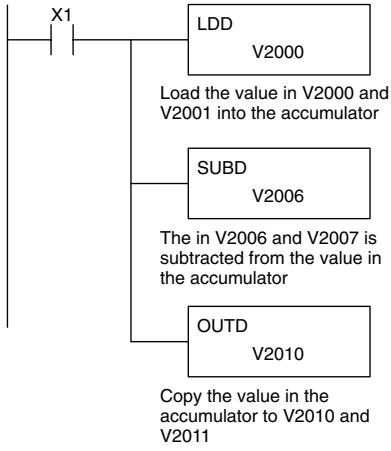


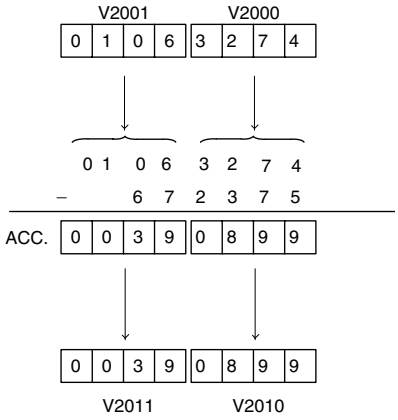
Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	K	0 – 99999999

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

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

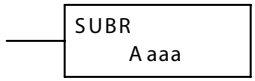
In the following example, when X1 is on, the value in V2000 and V2001 will be loaded into the accumulator using the Load Double instruction. The value in V2006 and V2007 is subtracted from the value in the accumulator. The value in the accumulator is copied to V2010 and V2011 using the Out Double instruction.





Subtract Real (SUBR)

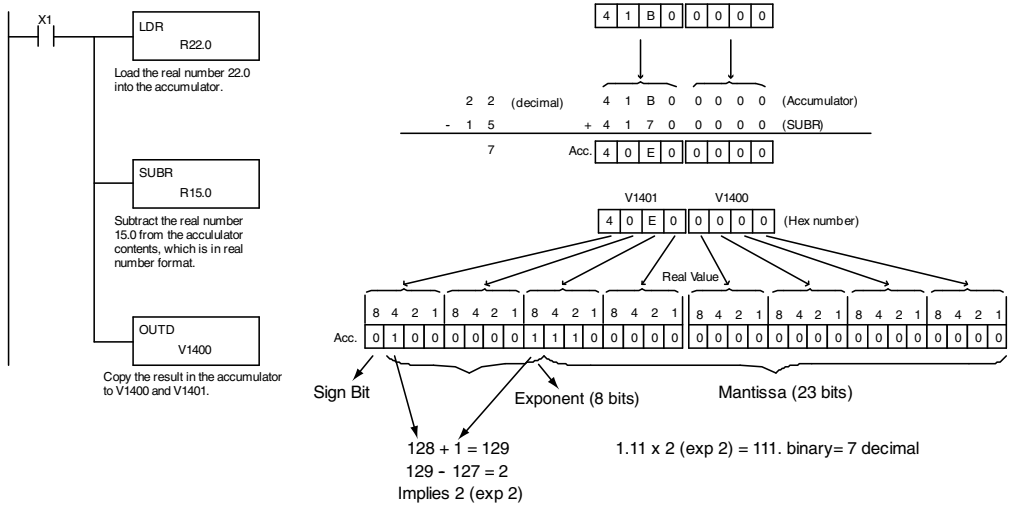
The Subtract Real is a 32-bit instruction that subtracts a real number, which is either two consecutive V-memory locations or a 32-bit constant, from a real number in the accumulator. The result is a 32-bit real number that resides in the accumulator. Both numbers must be Real data type (IEEE floating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



Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	R	-3.402823E^{+38} to + +3.402823E^{+38}

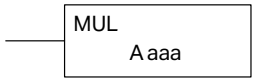
Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.
SP71	On anytime the V-memory specified by a pointer (P) is not valid.
SP72	On anytime the value in the accumulator is an 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.

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



Multiply (MUL)

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

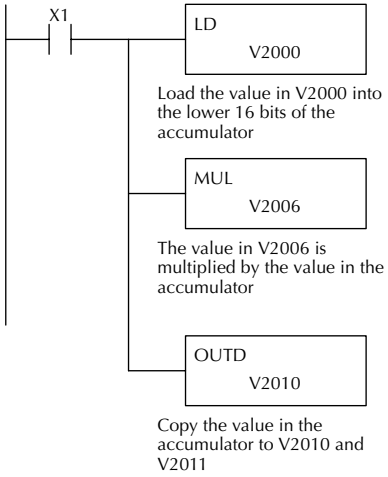


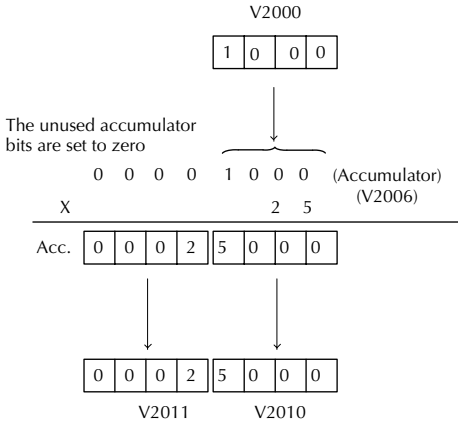
Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	K	0 – 9999

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

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

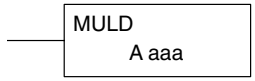
In the following example, when X1 is on, the value in V2000 will be loaded into the accumulator using the Load instruction. The value in V2006 is multiplied by the value in the accumulator using the Multiply instruction. The value in the accumulator is copied to V2010 and V2011 using the Out Double instruction.





Multiply Double (MULD)

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

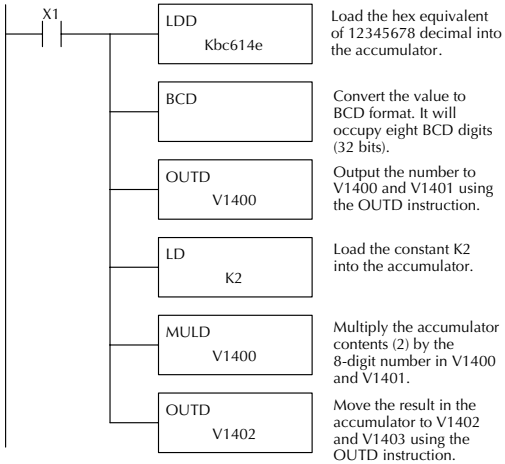


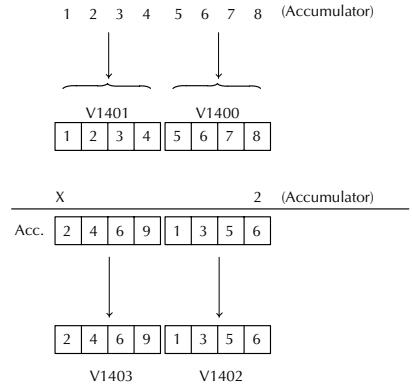
Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map

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

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

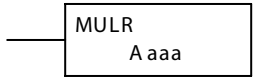
In the following example, when X1 is on, the constant Kbc614e hex will be loaded into the accumulator. When converted to BCD the number is "12345678". That number is stored in V1400 and V1401. After loading the constant K2 into the accumulator, we multiply it times 12345678, which gives us 24691356.





Multiply Real (MULR)

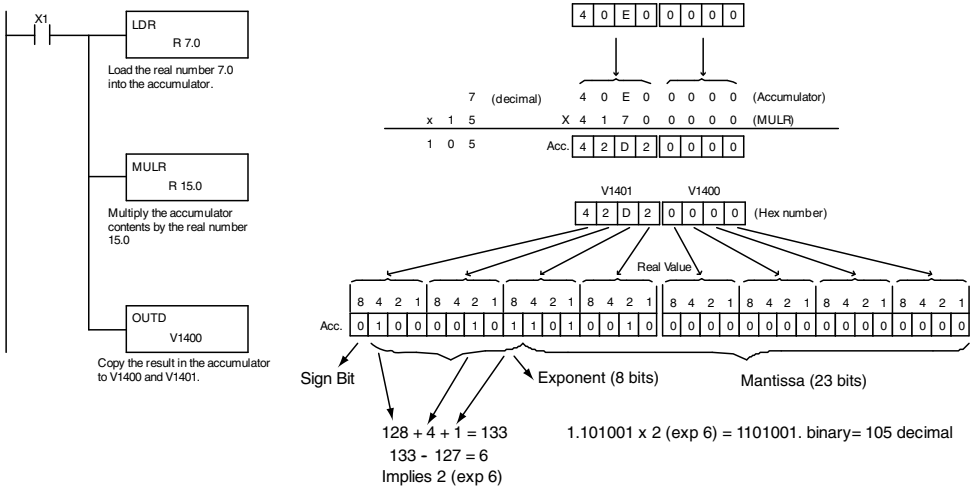
The Multiply Real instruction multiplies a real number in the accumulator with either a real constant or a real number occupying two consecutive V-memory locations. The result resides in the accumulator. Both numbers must be Real data type (IEEE floating point format).



Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Real Constant	R	$-3.402823E^{+38}$ to + $+3.402823E^{+38}$

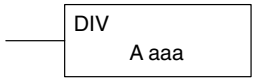
Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.
SP71	On anytime the V-memory specified by a pointer (P) is not valid.
SP72	On anytime the value in the accumulator is an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

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



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.

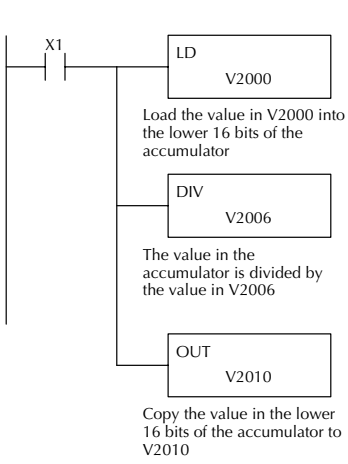


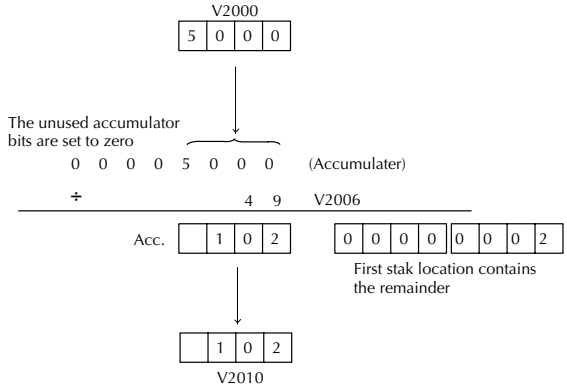
Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	K	0 – 9999

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

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

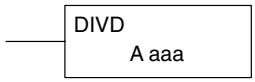
In the following example, when X1 is on, the value in V2000 will be loaded into the accumulator using the Load instruction. The value in the accumulator will be divided by the value in V2006 using the Divide instruction. The value in the accumulator is copied to V2010 using the Out instruction.





Divide Double (DIVD)

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

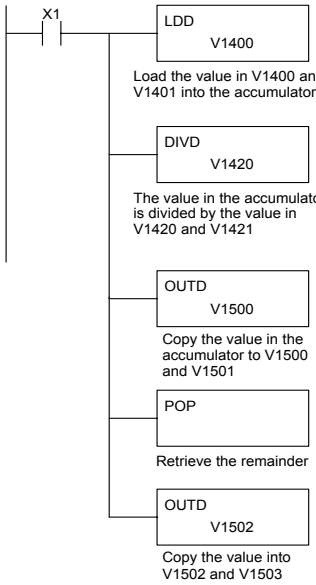


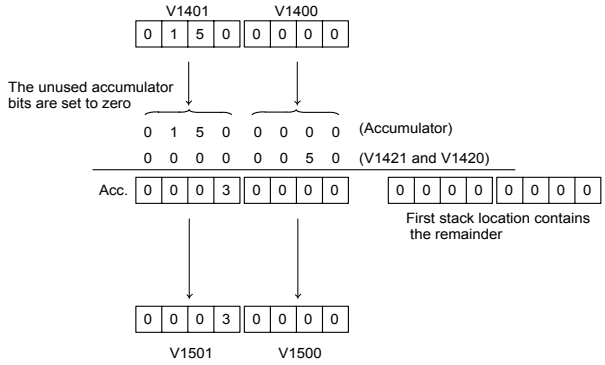
Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map

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.*

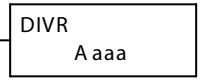
In the following example, when X1 is on, the value in V1400 and V1401 will be loaded into the accumulator using the Load Double instruction. The value in the accumulator is divided by the value in V1420 and V1421 using the Divide Double instruction. The first part of the quotient resides in the accumulator 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)

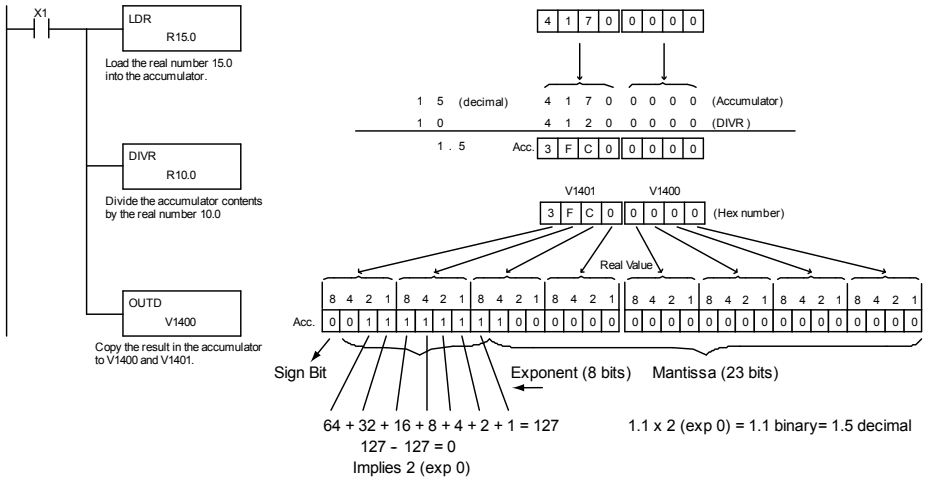
The Divide Real instruction divides a real number in the accumulator by either a real constant or a real number occupying two consecutive V-memory locations. The result resides in the accumulator. Both numbers must be Real data type (IEEE floating point format).



Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Real Constant	R	-3.402823E^{+38} to + +3.402823E^{+38}

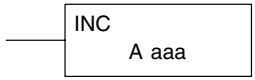
Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.
SP71	On anytime the V-memory specified by a pointer (P) is not valid.
SP72	On anytime the value in the accumulator is an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

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



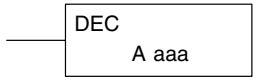
Increment (INC)

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



Decrement (DEC)

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

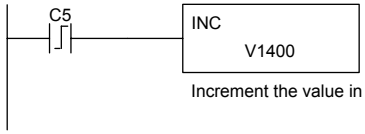


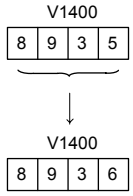
Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map

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

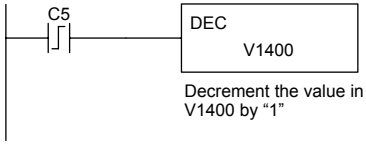
NOTE: *Status flags are valid only until another instruction uses the same flag.*

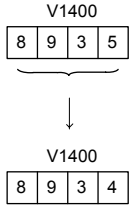
In the following increment example, when C5 makes an Off-to-On transition the value in V1400 increases by one.





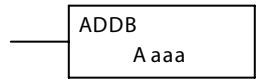
In the following decrement example, when C5 makes an Off-to-On transition the value in V1400 is decreased by one.





Add Binary (ADDB)

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

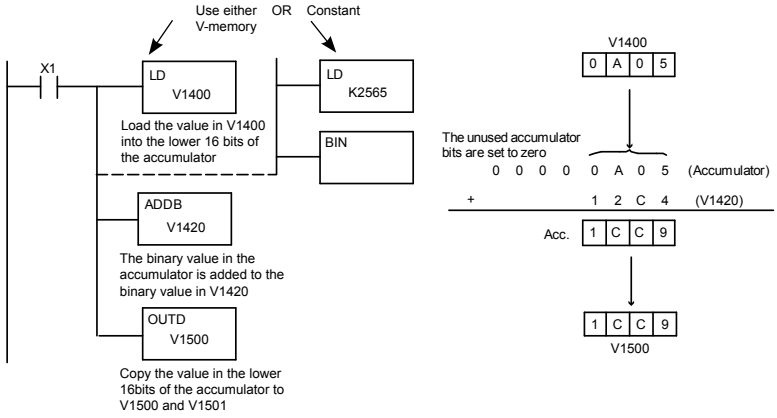


Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	K	0-FFFF

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

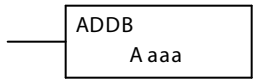
NOTE: *Status flags are valid only until another instruction uses the same flag.*

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 the accumulator will be added to the binary value in V1420 using the Add Binary instruction. The value in the accumulator is copied to V1500 and V1501 using the Out Double instruction.



Add Binary Double (ADDBD)

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.

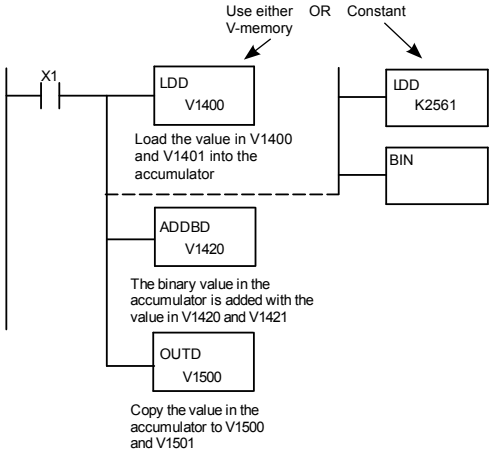


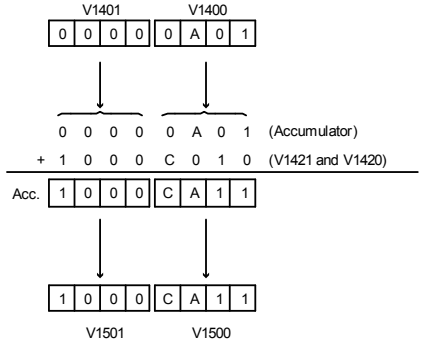
Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	K	0-FFFFFFFF

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

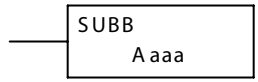
In the following example, when X1 is on, the value in V1400 and V1401 will be loaded into the accumulator using the Load Double instruction. The binary value in the accumulator is added with the binary value in V1420 and V1421 using the Add Binary Double instruction. The value in the accumulator is copied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





Subtract Binary (SUBB)

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

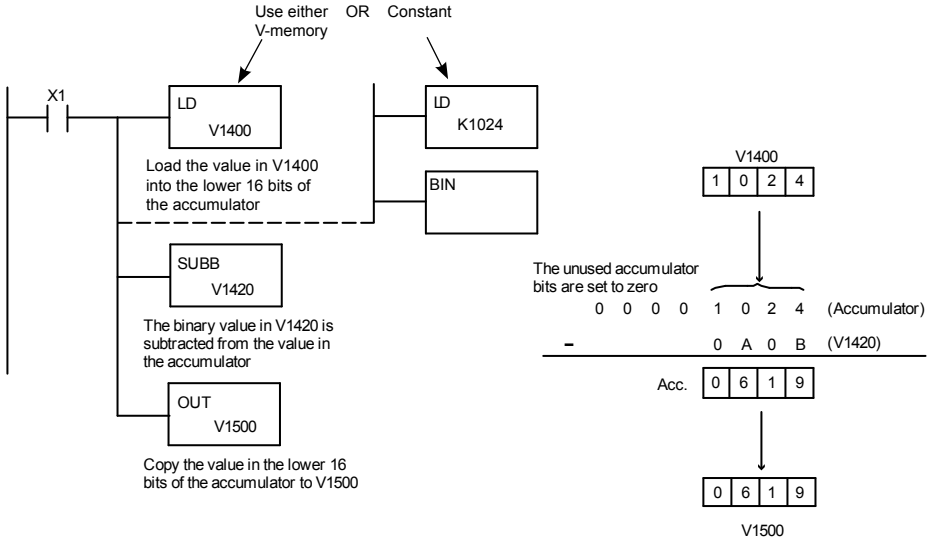


Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	K	0-FFFF

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.
SP70	On anytime the value in the accumulator is negative.

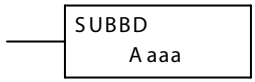
NOTE: *Status flags are valid only until another instruction uses the same flag.*

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



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.

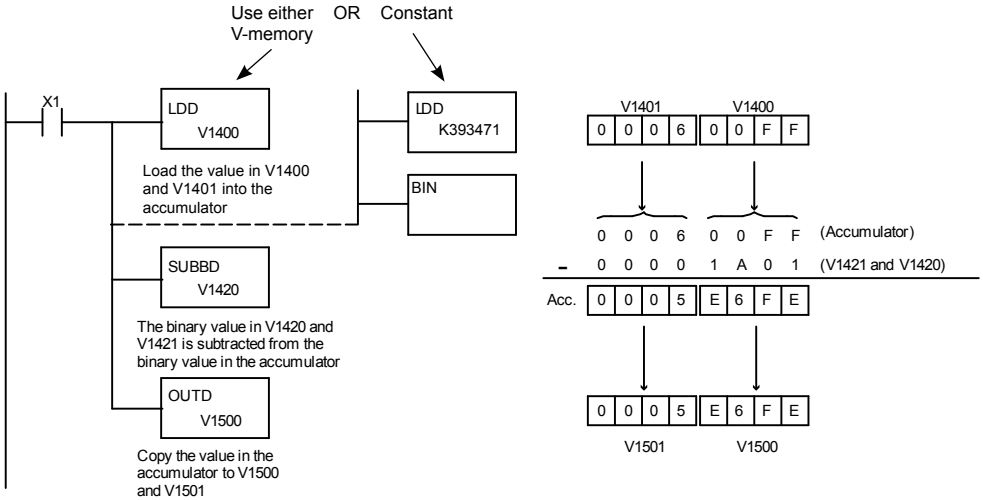


Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	K	0-FFFFFFFFF

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

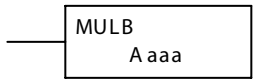
NOTE: *Status flags are valid only until another instruction uses the same flag.*

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



Multiply Binary (MULB)

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

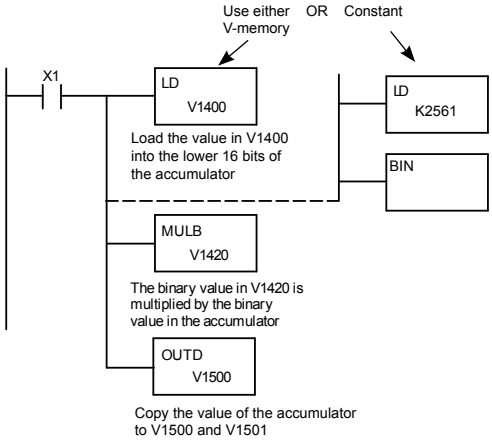


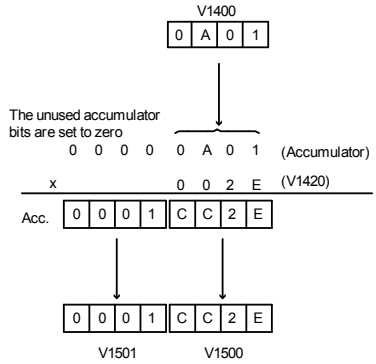
Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	K	0-FFFF

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

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

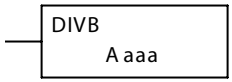
In the following example, when X1 is on, the value in V1400 will be loaded into the accumulator using the Load instruction. The binary value in V1420 is multiplied by the binary value in the accumulator using the Multiply Binary instruction. The value in the accumulator is copied to V1500 using the Out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





Divide Binary (DIVB)

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

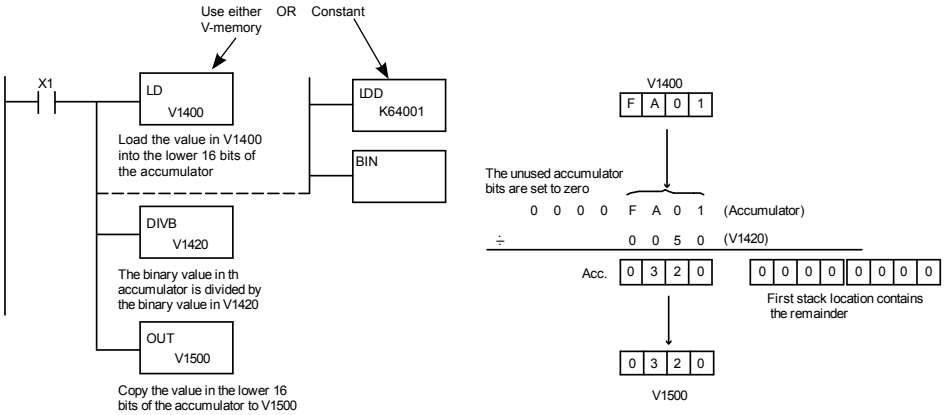


Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	K	0-FFFF

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

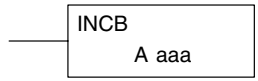
NOTE: *Status flags are valid only until another instruction uses the same flag.*

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



Increment Binary (INCB)

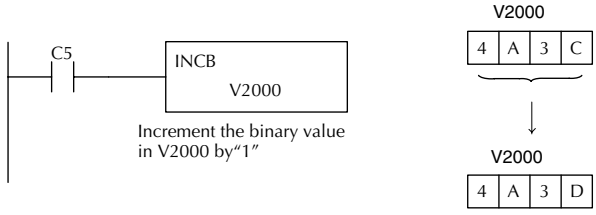
The Increment Binary instruction increments a binary value in a specified V-memory location by "1" each time the instruction is executed.



Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map

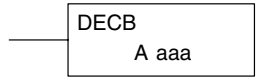
Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.

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



Decrement Binary (DECB)

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

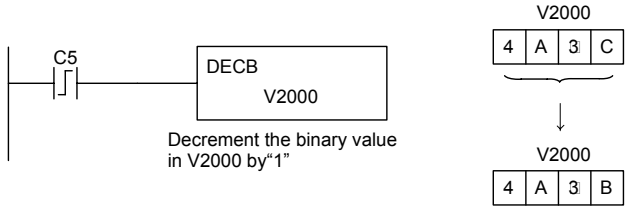


Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map

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

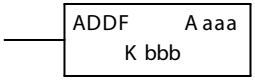
NOTE: *Status flags are valid only until another instruction uses the same flag.*

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



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 a range of discrete bits. The specified range (Kbbb) can be 1 to 32 consecutive bits. The result resides in the accumulator.

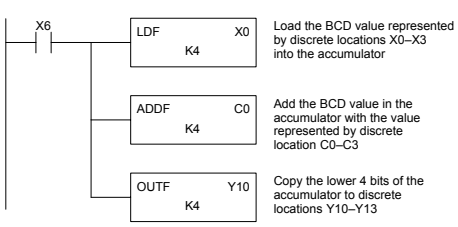


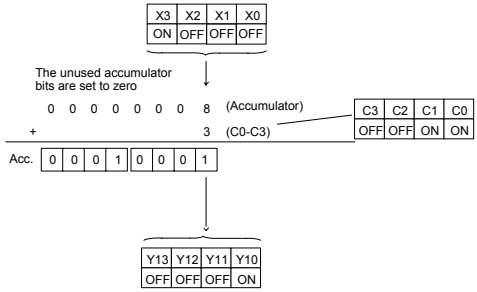
Operand Data Type		D4-454 Range	
	A	aaa	bbb
Inputs	X	0 – 1777	– –
Outputs	Y	0 – 1777	– –
Control Relays	C	0 – 3777	– –
Stage Bits	S	0 – 1777	– –
Timer Bits	T	0 – 377	– –
Counter Bits	CT	0 – 377	– –
Special Relays	SP	0 – 777	– –
Global I/O	GX	0 – 3777	– –
Constant	K	– –	1 – 32

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

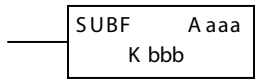
In the following example, when X6 is on, the BCD value formed by discrete locations X0 – X3 is loaded into the accumulator using the LDF instruction. The BCD value formed by discrete locations C0 – C3 is added to the value in the accumulator using the ADDF instruction. The value in the lower four bits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

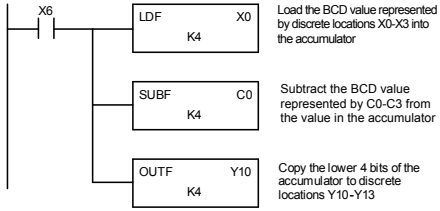


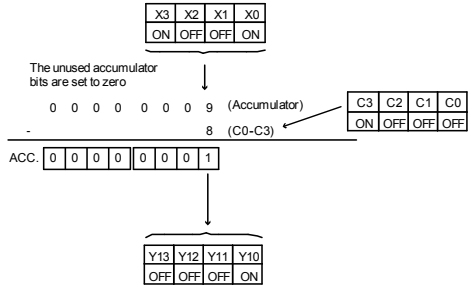
Operand Data Type		D4-454 Range	
	A	aaa	bbb
Inputs	X	0 – 1777	– –
Outputs	Y	0 – 1777	– –
Control Relays	C	0 – 3777	– –
Stage Bits	S	0 – 1777	– –
Timer Bits	T	0 – 377	– –
Counter Bits	CT	0 – 377	– –
Special Relays	SP	0 – 777	– –
Global I/O	GX	0 – 3777	– –
Constant	K	– –	1 – 32

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP65	On when the 32 bit subtraction instruction results in a borrow
SP70	On any time the value in the accumulator is negative.
SP73	On when a signed addition or subtraction results in an incorrect sign bit.
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.*

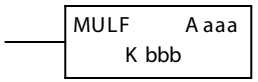
In the following example, when X6 is on, the BCD value formed by discrete locations X0 – X3 is loaded into the accumulator using the LDF instruction. The BCD value formed by discrete location C0 – C3 is subtracted from the BCD value in the accumulator using the SUBF instruction. The value in the lower four bits of the accumulator is copied to Y10 – Y13 using the OUTF instruction.





Multiply Formatted (MULF)

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

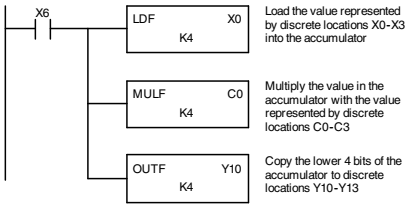


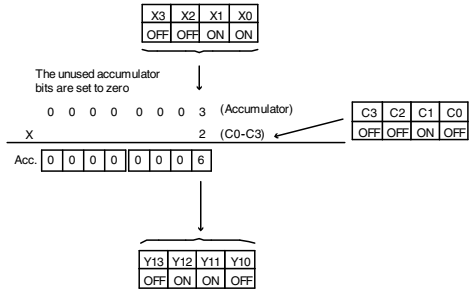
Operand Data Type		D4-454 Range	
	A	aaa	bbb
Inputs	X	0 – 1777	– –
Outputs	Y	0 – 1777	– –
Control Relays	C	0 – 3777	– –
Stage Bits	S	0 – 1777	– –
Timer Bits	T	0 – 377	– –
Counter Bits	CT	0 – 377	– –
Special Relays	SP	0 – 777	– –
Global I/O	GX	0-3777	– –
Constant	K	– –	1 – 16

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On any time 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.*

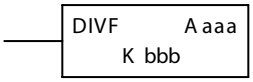
In the following example, when X6 is on, the value formed by discrete locations X0 – X3 is loaded into the accumulator using the Load Formatted instruction. The value formed by discrete locations C0 – C3 is multiplied by the value in the accumulator using the Multiply Formatted instruction. The value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





Divide Formatted (DIVF)

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

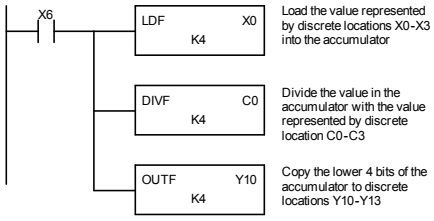


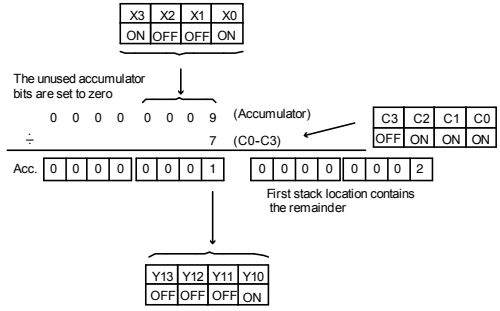
Operand Data Type		D4-454 Range	
	A	aaa	bbb
Inputs	X	0 – 1777	– –
Outputs	Y	0 – 1777	– –
Control Relays	C	0 – 3777	– –
Stage Bits	S	0 – 1777	– –
Timer Bits	T	0 – 377	– –
Counter Bits	CT	0 – 377	– –
Special Relays	P	0 – 777	– –
Global I/O	X	0 – 3777	– –
Constant	K	– –	1 – 16

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

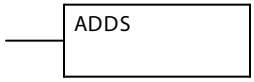
In the following example, when X6 is on, the value formed by discrete locations X0 – X3 is loaded into the accumulator using the Load Formatted instruction. The value in the accumulator is divided by the value formed by discrete location C0 – C3 using the Divide Formatted instruction. The value in the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





Add Top of Stack (ADDS)

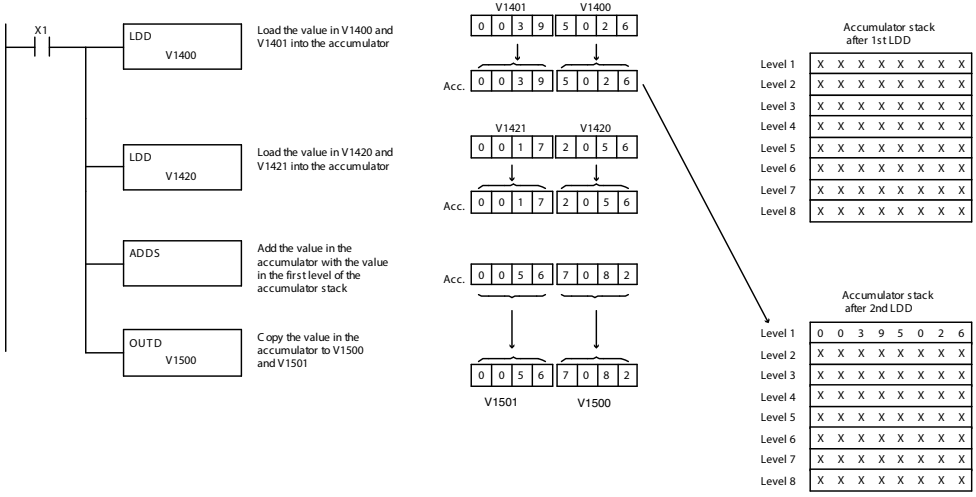
Add Top of Stack is a 32-bit instruction that adds the BCD value in the accumulator with the BCD value in the first level of the accumulator stack. The result resides in the accumulator. The value in the first level of the accumulator stack is removed and all stack values are moved up one level.



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

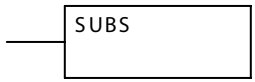
NOTE: *Status flags are valid only until another instruction uses the same flag.*

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)

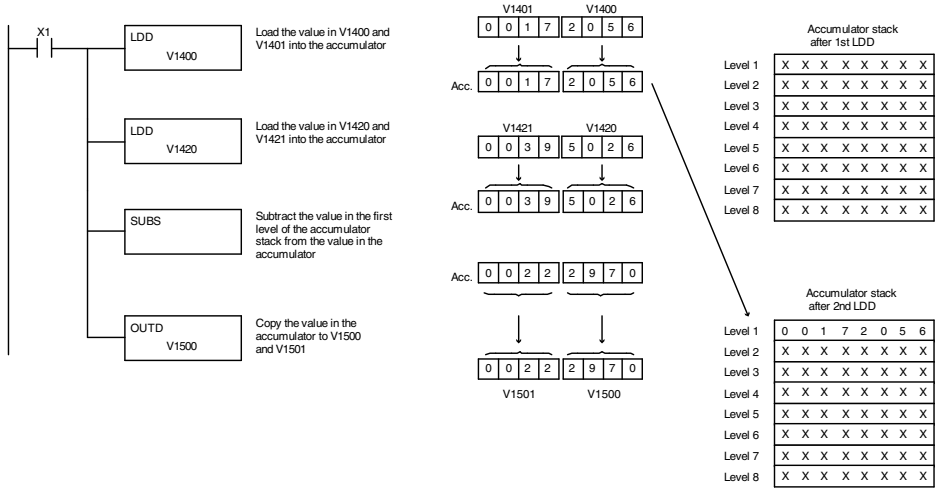
Subtract Top of Stack is a 32-bit instruction that subtracts the BCD value in the first level of the accumulator stack from the BCD value in the accumulator. The result resides in the accumulator. The value in the first level of the accumulator stack is removed and all stack values are moved up one level.



Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP65	On when the 32 bit subtraction instruction results in a borrow.
SP70	On anytime the value in the accumulator is negative.
SP73	On when a signed addition or subtraction results in an incorrect sign bit.
SP75	On when a BCD instruction is executed and a NON-BCD number was encountered.

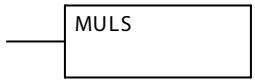
NOTE: *Status flags are valid only until another instruction uses the same flag.*

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



Multiply Top of Stack (MULS)

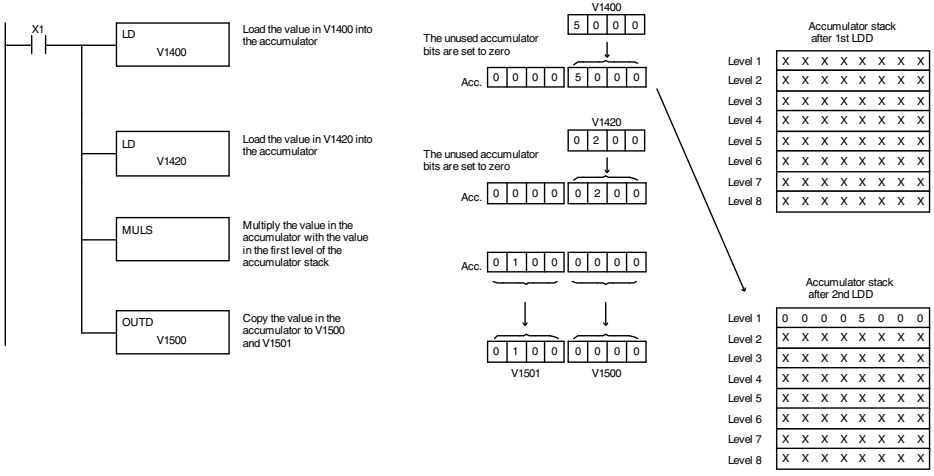
Multiply Top of Stack is a 16-bit instruction that multiplies a 4-digit BCD value in the first level of the accumulator stack by a 4-digit BCD value in the accumulator. The result resides in the accumulator. The value in the first level of the accumulator stack is removed and all stack values are moved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



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

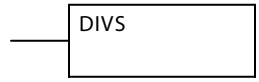
NOTE: *Status flags are valid only until another instruction uses the same flag.*

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.



Divide by Top of Stack (DIVS)

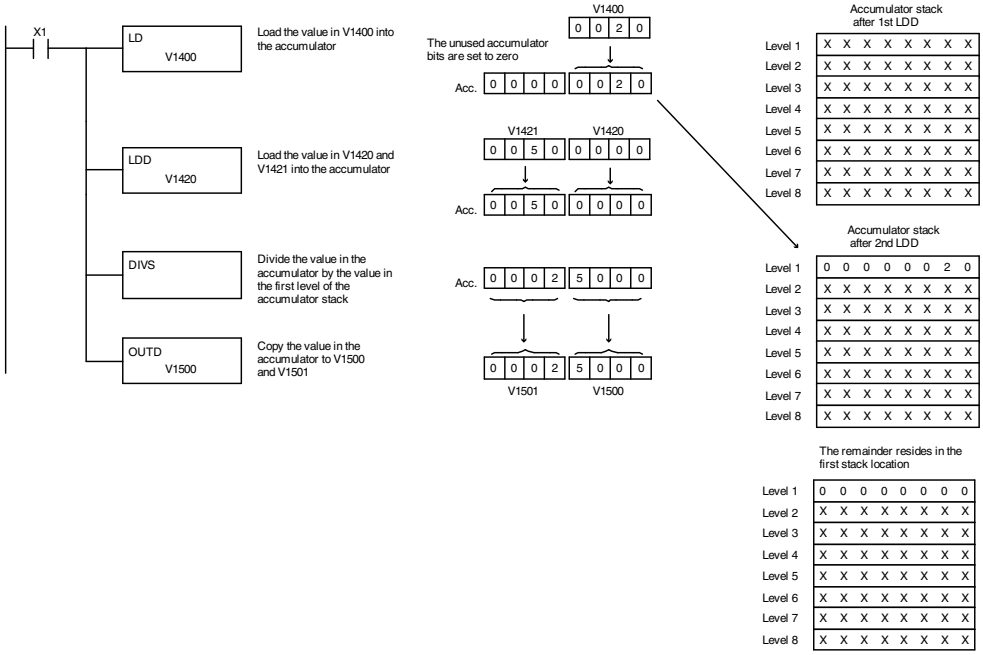
Divide Top of Stack is a 32-bit instruction that divides the 8-digit BCD value in the accumulator by a 4-digit BCD value in the first level of the accumulator stack. The result resides in the accumulator and the remainder resides in the first level of the accumulator stack



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

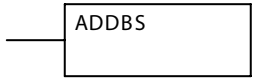
NOTE: *Status flags are valid only until another instruction uses the same flag.*

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



Add Binary Top of Stack (ADDBS)

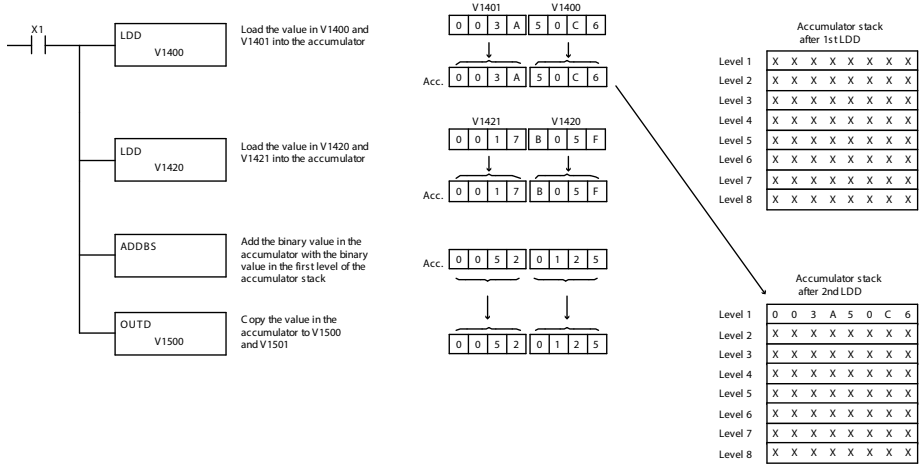
Add Binary Top of Stack instruction is a 32-bit instruction that adds the binary value in the accumulator with the binary value in the first level of the accumulator stack. The result resides in the accumulator. The value in the first level of the accumulator stack is removed and all stack values are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



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

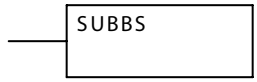
NOTE: *Status flags are valid only until another instruction uses the same flag.*

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 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)

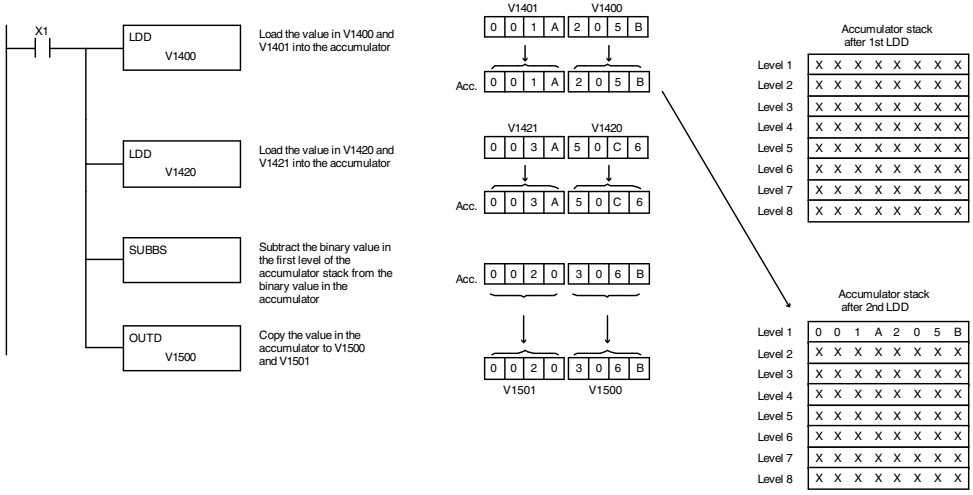
Subtract Binary Top of Stack is a 32-bit instruction that subtracts the binary value in the first level of the accumulator stack from the binary value in the accumulator. The result resides in the accumulator. The value in the first level of the accumulator stack is removed and all stack locations are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



Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP65	On when the 32-bit subtraction instruction results in a borrow.
SP70	On any time the value in the accumulator is negative.
SP73	On when a signed addition or subtraction results in an incorrect sign bit.

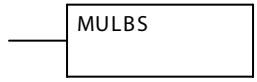
NOTE: *Status flags are valid only until another instruction uses the same flag.*

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 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)

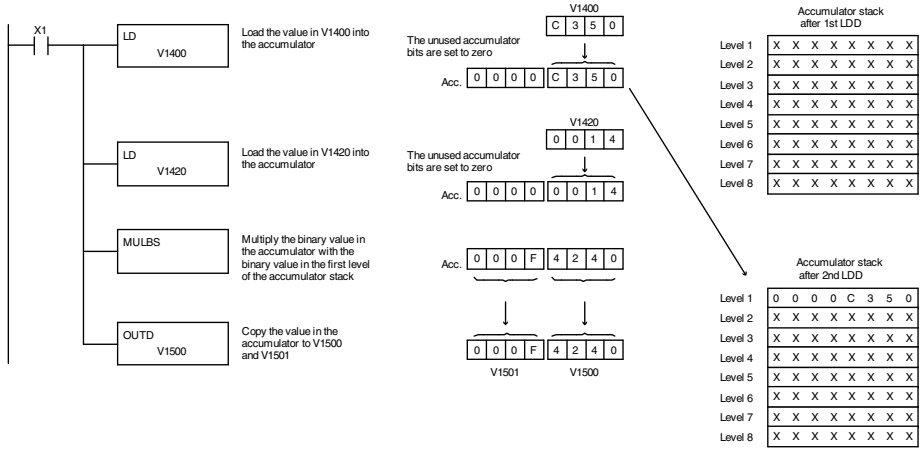
Multiply Binary Top of Stack is a 16-bit instruction that multiplies the 16-bit binary value in the first level of the accumulator stack by the 16-bit binary value in the accumulator. The result resides in the accumulator and can be 32 bits (8 digits max.). The value in the first level of the accumulator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



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

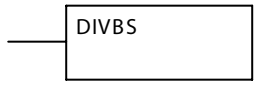
NOTE: *Status flags are valid only until another instruction uses the same flag.*

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



Divide Binary by Top OF Stack (DIVBS)

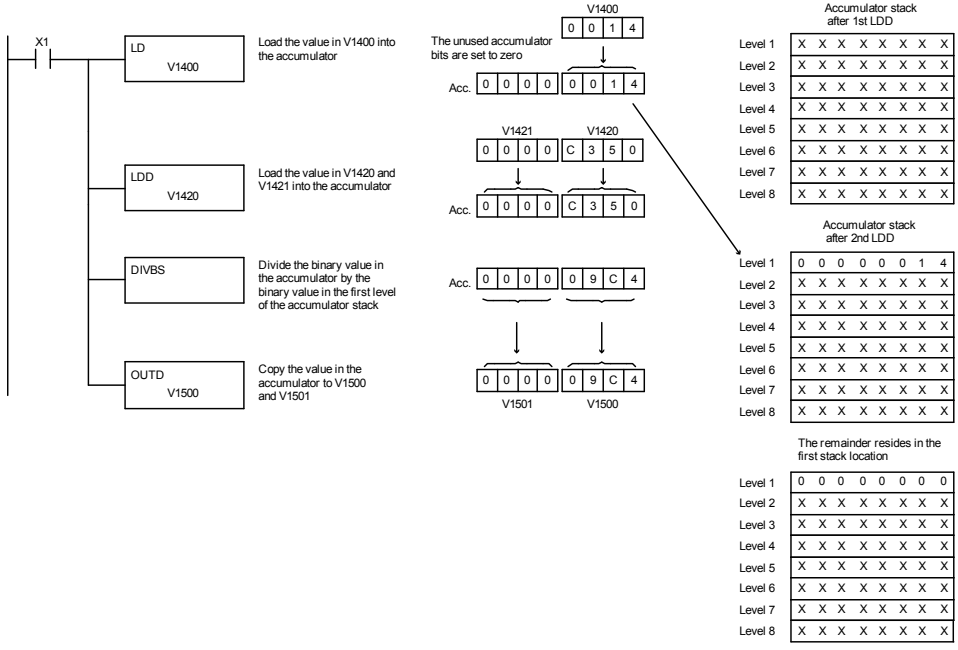
Divide Binary Top of Stack is a 32-bit instruction that divides the 32-bit binary value in the accumulator by the 16-bit binary value in the first level of the accumulator stack. The result resides in the accumulator and the remainder resides in the first level of the accumulator stack.



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

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

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



Transcendental Functions

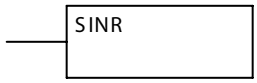
The D4-454 CPU features special numerical functions to complement its real number capability. The transcendental functions include the trigonometric sine, cosine, and tangent, and also their inverses (arc sine, arc cosine, and arc tangent). The square root function is also grouped with these other functions.

The transcendental math instructions operate on a real number in the accumulator (it cannot be BCD or binary). The real number result resides in the accumulator. The square root function operates on the full range of positive real numbers. The sine, cosine and tangent functions require numbers expressed in radians. You can work with angles expressed in degrees by first converting them to radians with the Radian (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 an invalid floating point number
SP73	On when a signed addition or subtraction results in an incorrect sign bit.
SP75	On when a real number instruction is executed and a non-real number encountered.

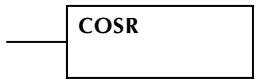
Sine Real (SINR)

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 must be Real data type (IEEE floating point format).



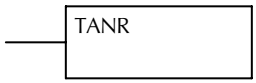
Cosine Real (COSR)

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 must be Real data type (IEEE floating point format)..



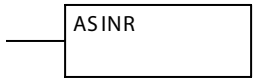
Tangent Real (TANR)

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 must be Real data type (IEEE floating point 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 must be Real data type (IEEE floating point format).

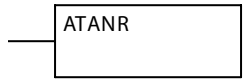


Arc Cosine Real (ACOSR)

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 must be Real data type (IEEE floating point format).

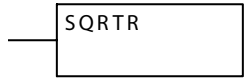
Arc Tangent Real (ATANR)

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 must be Real data type (IEEE floating point format).



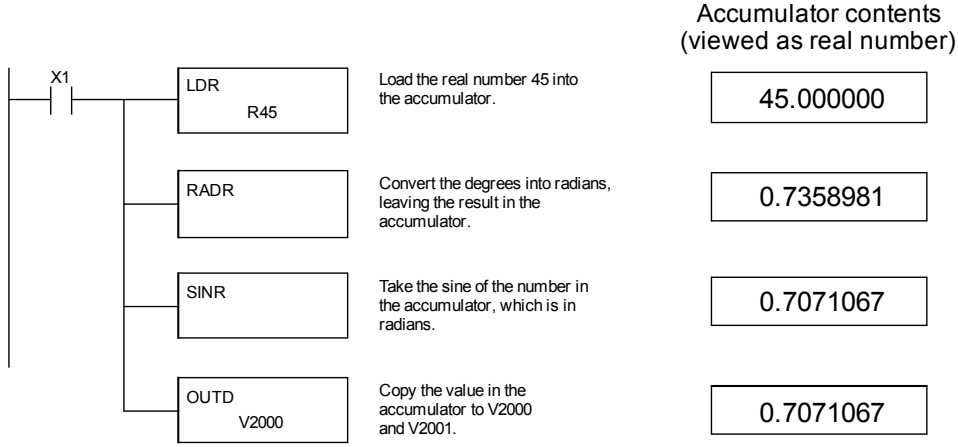
Square Root Real (SQRTR)

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 must be Real data type (IEEE floating point format).



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

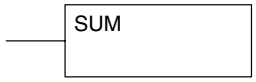
The following example takes the sine of 45 degrees. Since these transcendental functions operate only on real numbers, we do an LDR (load real) 45. The trig functions operate only in radians, so we must convert the degrees to radians by using the RADR command. After using the SINR (Sine Real) instruction, 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.



Bit Operation Instructions

Sum (SUM)

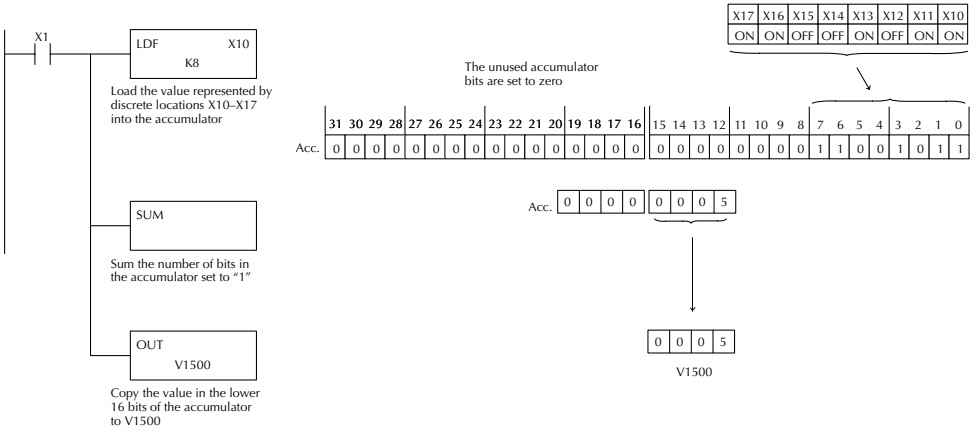
The Sum instruction counts number of bits that are set to "1" in the accumulator. The HEX result resides in the accumulator.



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

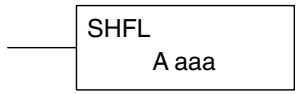
In the following example, when X1 is on, the value formed by discrete locations X10 – X17 is loaded into the accumulator using the Load Formatted instruction. The number of bits in the accumulator set to "1" is counted using the Sum instruction. The value in the accumulator is copied to V1500 using the Out instruction.

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



Shift Left (SHFL)

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

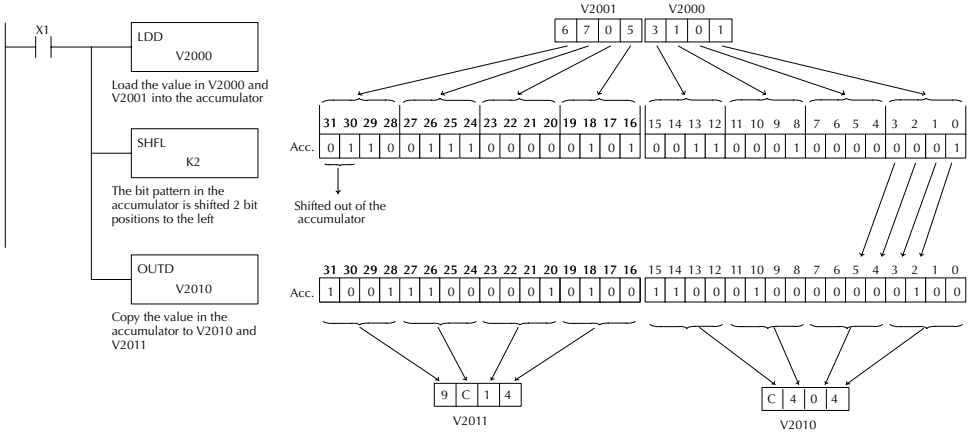


Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map
Constant	K	1-32

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

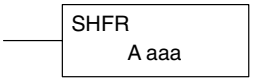
In the following example, when X1 is on, the value in V2000 and V2001 will be loaded into the accumulator using the Load Double instruction. The bit pattern in the accumulator is shifted 2 bits to the left using the Shift Left instruction. The value in the accumulator is copied to V2010 and V2011 using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

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



Shift Right (SHFR)

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

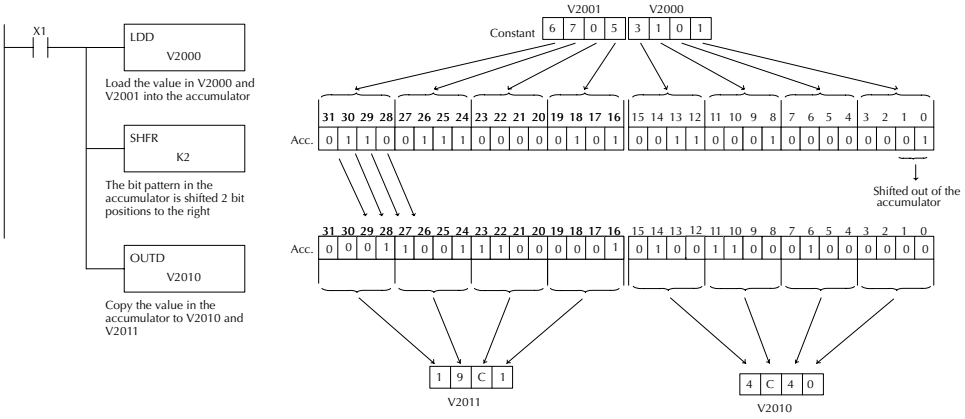


Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map
Constant	K	1-32

Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value 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 bit pattern in the accumulator is shifted 2 bits to the right using the Shift Right instruction. The value in the accumulator is copied to V2010 and V2011 using the Out Double instruction.

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



Rotate Left (ROTL)

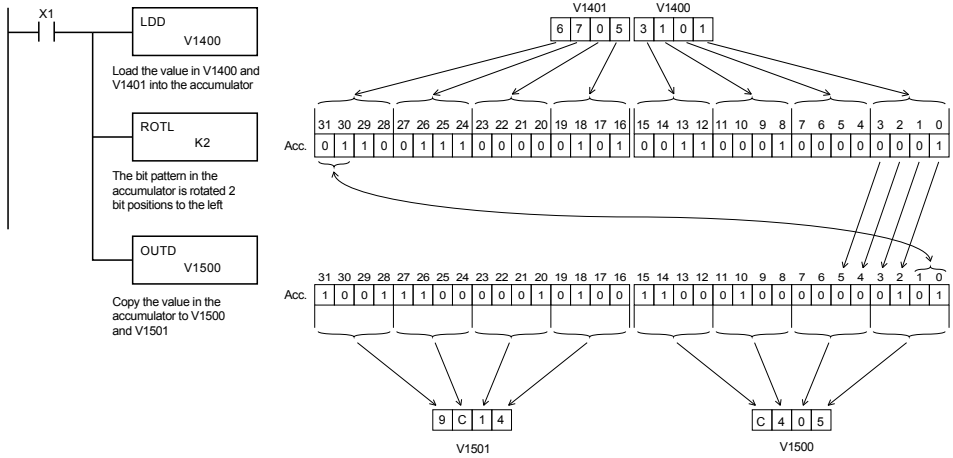
Rotate Left is a 32-bit instruction that rotates the bits in the accumulator a specified number (Aaaa) of places to the left.

Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map
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.
SP70	On anytime the value in the accumulator is negative.

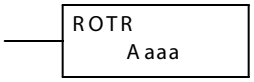
In the following example, when X1 is on, the value in V1400 and V1401 will be loaded into the accumulator using the Load Double instruction. The bit pattern in the accumulator is rotated 2 bit positions to the left using the Rotate Left instruction. The value in the accumulator is copied to V1500 and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

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



Rotate Right (ROTR)

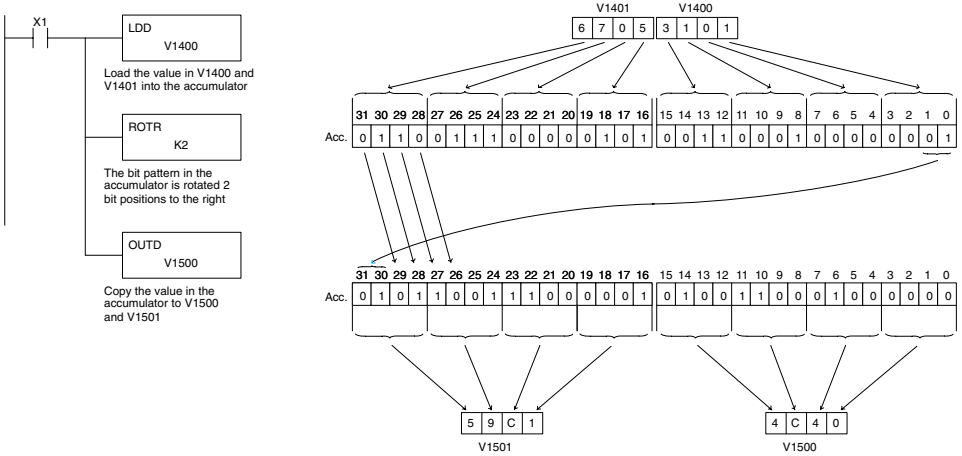
Rotate Right is a 32-bit instruction that rotates the bits in the accumulator a specified number (Aaaa) of places to the right.



Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map
Constant	K	1-32

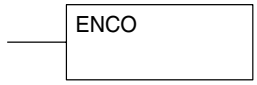
Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.

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



Encode (ENCO)

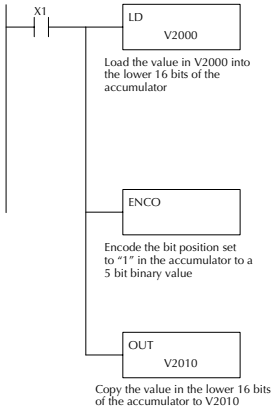
The Encode instruction encodes the bit position in the accumulator having a value of 1, and returns the appropriate binary representation. If the most significant bit is set to 1 (Bit 31), the Encode instruction would place the value HEX 1F (decimal 31) in the accumulator. If the value to be encoded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

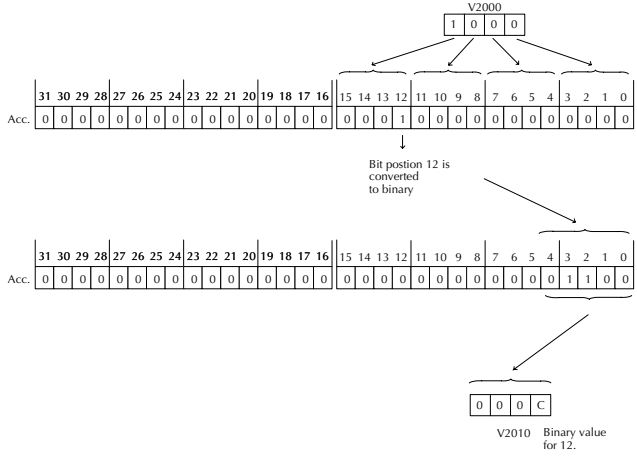


Discrete Bit Flags	Description
SP53	On when the value of the operand is larger than the accumulator can work with.

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

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

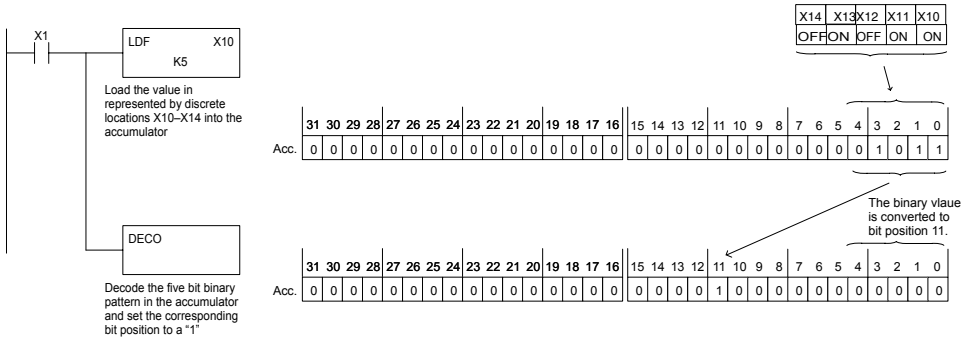




Decode (DECO)

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

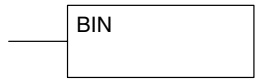
In the following example when X1 is on, the value formed by discrete locations X10 – X14 is loaded into the accumulator using the Load Formatted instruction. The 5- bit binary pattern in the accumulator is decoded by setting the corresponding bit position to a "1" using the Decode instruction.



Number Conversion Instructions (Accumulator)

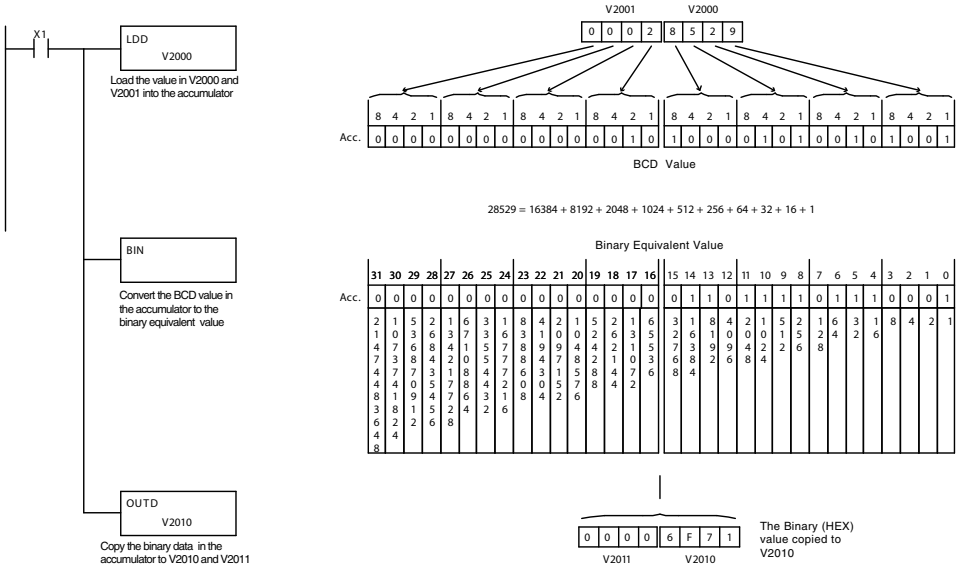
Binary (BIN)

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



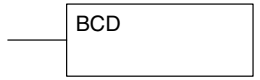
Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.
SP75	On when a BCD instruction is executed and a NON – BCD number was encountered.

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



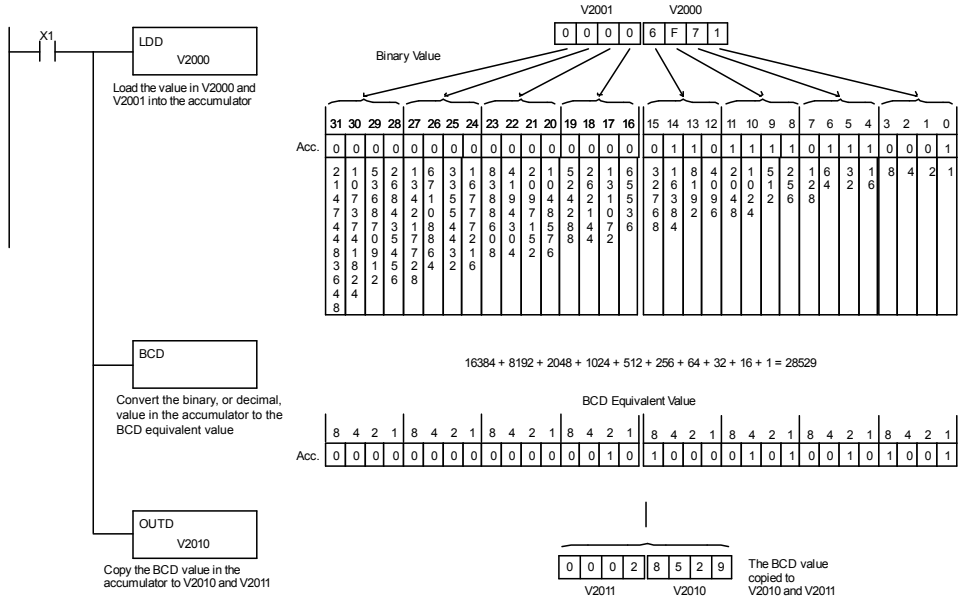
Binary Coded Decimal (BCD)

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



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

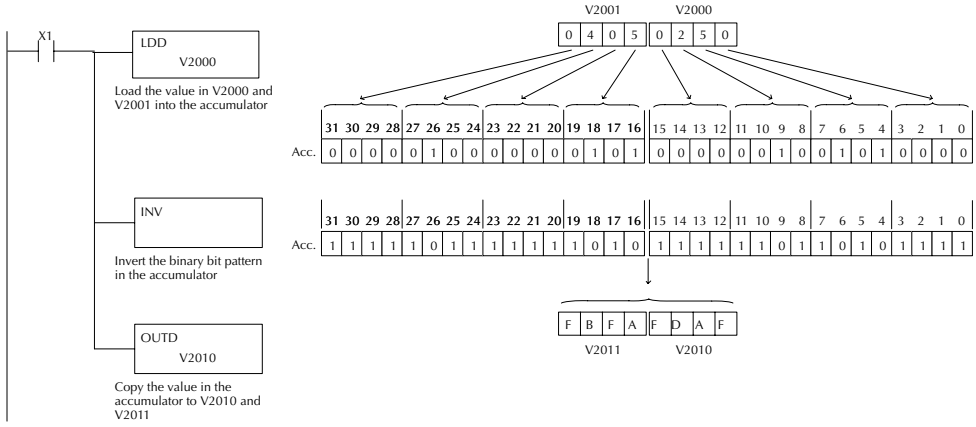
In the following example, when X1 is on, the binary, or decimal, value in V2000 and V2001 is loaded into the accumulator using the Load Double instruction. The value in the accumulator is converted to the BCD equivalent value using the BCD instruction. The BCD value in the accumulator is copied to V2010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



Invert (INV)

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

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

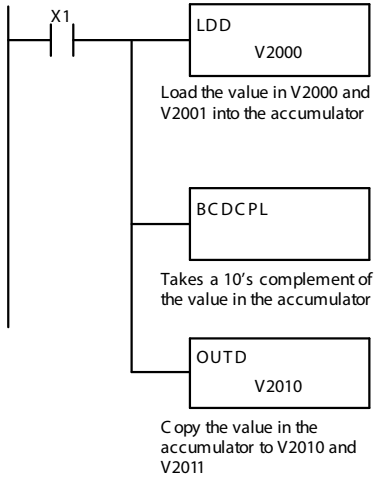


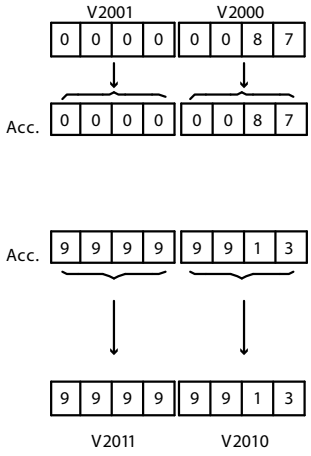
Ten's Complement (BCDCPL)

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

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

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





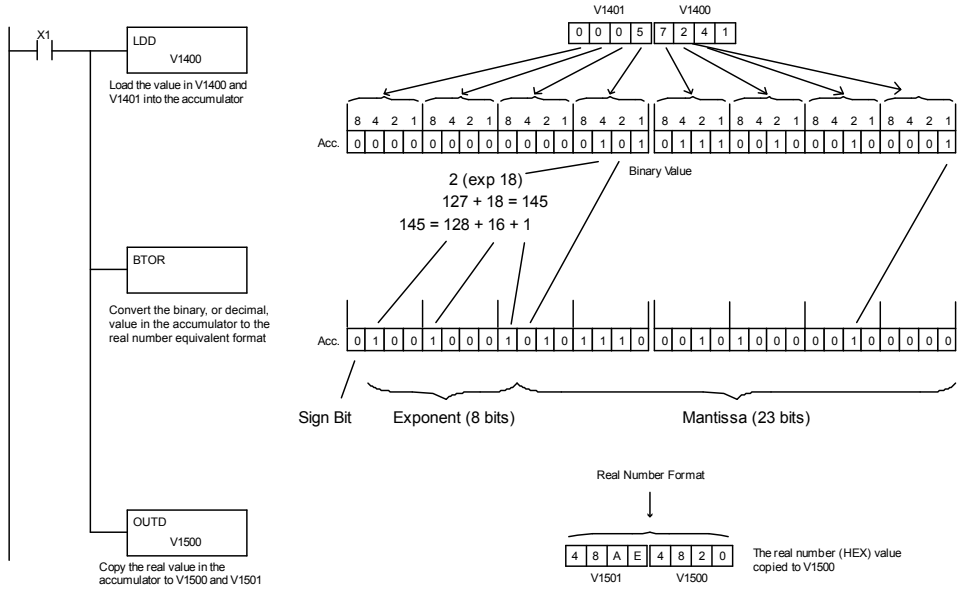
Binary to Real Conversion (BTOR)

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

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

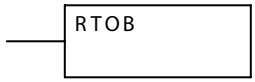
Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.

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, or decimal, 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.



Real to Binary Conversion (RTOB)

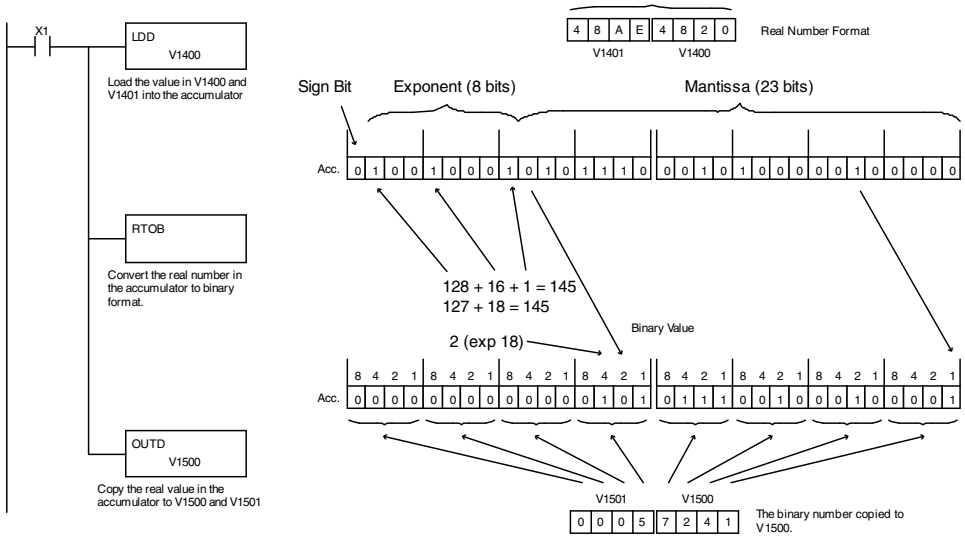
The Real-to-Binary instruction converts the real number in the accumulator to a binary value. The result resides in the accumulator. Both the binary and the real number may use all 32 bits of the accumulator.



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

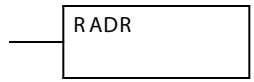
Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.
SP72	On anytime the value in the accumulator is an invalid floating point number.
SP73	On when a signed addition or subtraction 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.



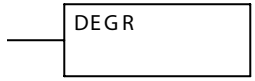
Radian Real Conversion (RADR)

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



Degree Real Conversion (DEGR)

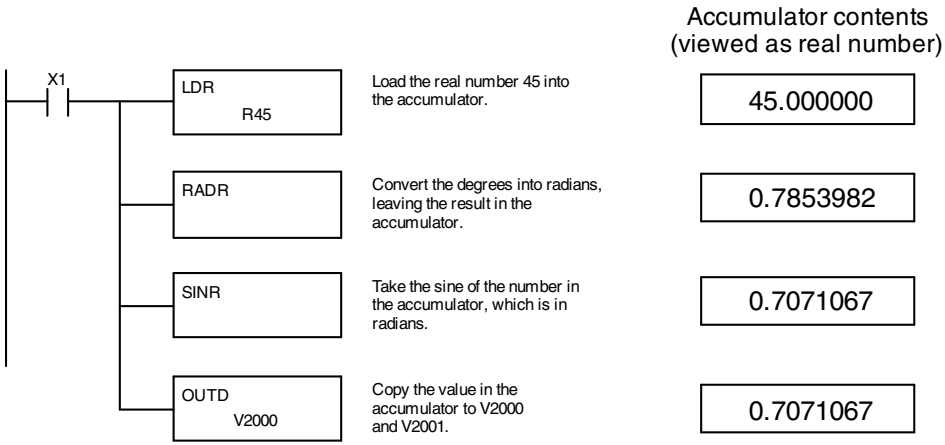
The Degree Real instruction converts the degree real radian value stored in the accumulator to the equivalent real number in degrees. The result resides in the accumulator.



The two instructions described above convert real numbers into the accumulator from degree format to radian format, and vice-versa. In degree format, a circle contains 360 degrees. In radian format, a circle contains 2π (about 6.28) radians. These convert between both positive and negative real numbers, and for angles greater than a full circle. These functions are very useful when combined with the transcendental trigonometric functions (see the section on math instructions).

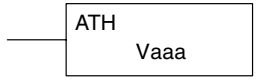
Discrete Bit Flags	Description
SP63	On when the result of the instruction causes the value in the accumulator to be zero.
SP70	On anytime the value in the accumulator is negative.
SP72	On anytime the value in the accumulator is an invalid floating point number.
SP73	On when a signed addition or subtraction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

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



ASCII to HEX (ATH)

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



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

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

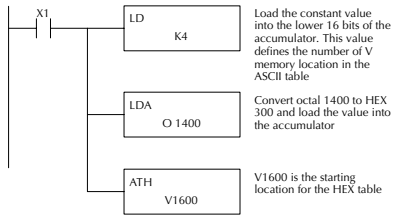
Step 3: Specify the starting V-memory location (Vaaa) for the HEX table in the ATH instruction.

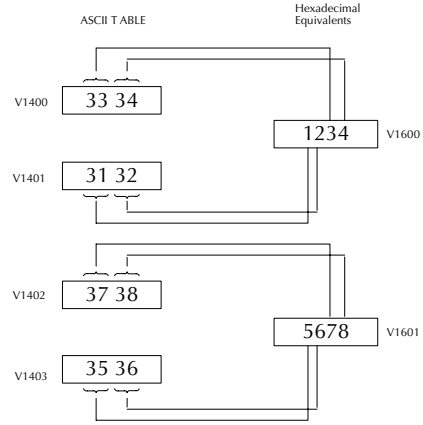
Helpful Hint: For parameters that require HEX values when referencing memory locations, the LDA instruction can be used to convert an octal address to the HEX equivalent and load the value into the accumulator.

Operand Data Type		D4-454 Range
		aaa
V-memory	V	See memory map

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

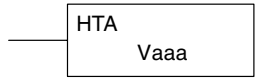
ASCII Values Valid for ATH Conversion			
ASCII Value	Hex Value	ASCII Value	Hex Value
30	0	38	8
31	1	39	9
32	2	41	A
33	3	42	B
34	4	43	C
35	5	44	D
36	6	45	E
37	7	46	F





HEX to ASCII (HTA)

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



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

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

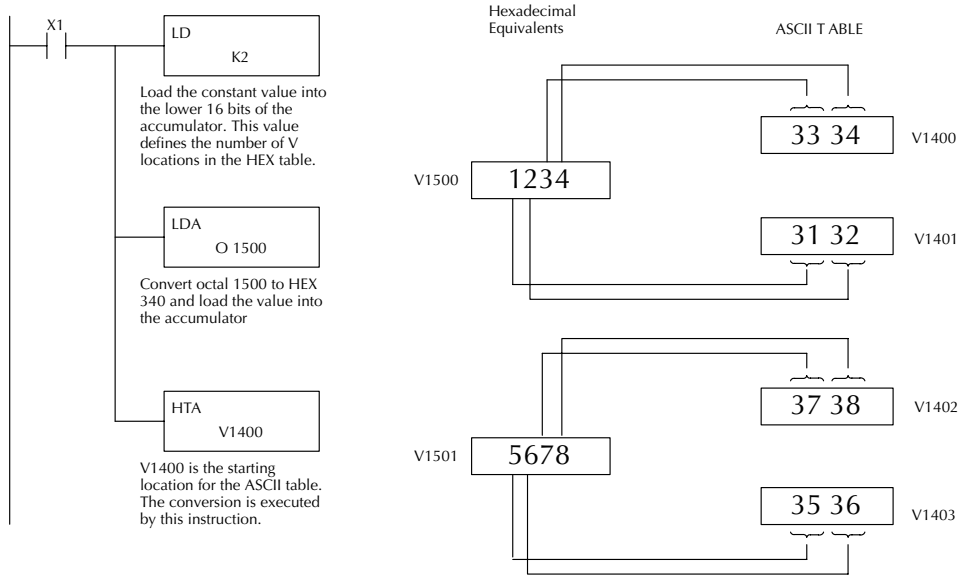
Step 2: Load the starting V-memory location for the HEX table into the accumulator. This parameter must be a HEX value.

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

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

Operand Data Type		D4-454 Range
		aaa
V-memory	V	See memory map

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



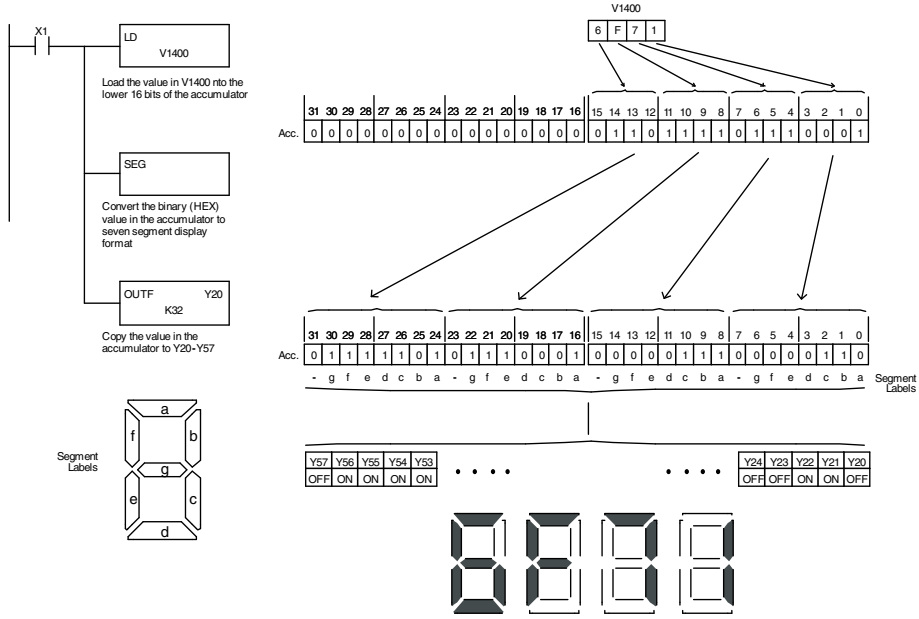
The table below lists valid ASCII values for HTA conversion.

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

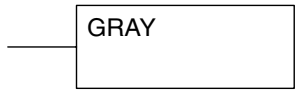
Segment (SEG)

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

In the following example, when X1 is on, the value in V1400 is loaded into the lower 16 bits of the accumulator using the Load instruction. The HEX value in the accumulator is converted to seven segment format using the Segment instruction. The bit pattern in the accumulator is copied to Y20 – Y57 using the Out Formatted instruction.



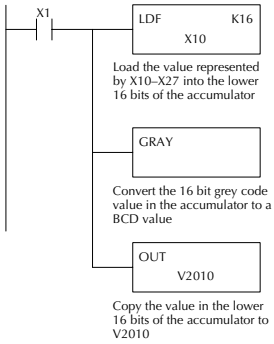
Gray Code (GRAY)

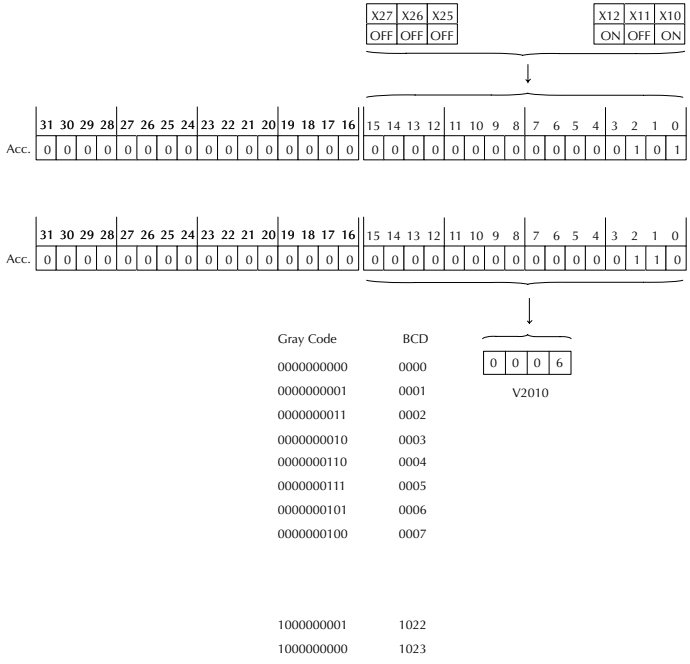


The Gray code instruction converts a 16-bit gray code value to a BCD value. The BCD conversion requires 10 bits of the accumulator. The upper 22 bits are set to "0". This instruction is designed for use with devices (typically encoders) that use the gray code numbering scheme.

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.





Shuffle Digits (SFLDGT)

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

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

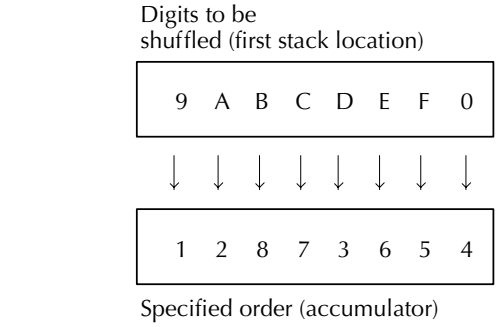
Step 2: Load the order that the digits will be shuffled to into the accumulator.

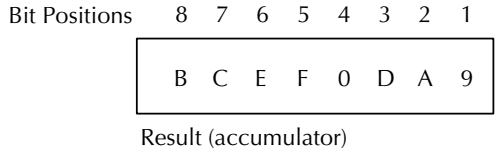
Step 3: Insert the SFLDGT instruction.

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

Shuffle Digits Block Diagram

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





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

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

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

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

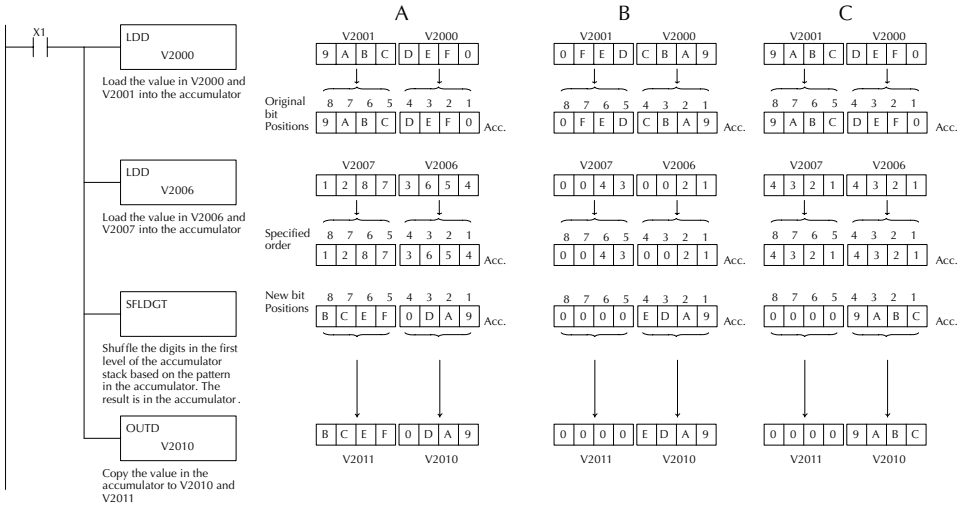
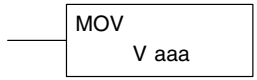


Table Instructions

Move (MOV)

The Move instruction moves the values from a V-memory table to another V-memory table the same length (a table being a consecutive group of V-memory locations). 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 MOV 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, 4096 decimal).

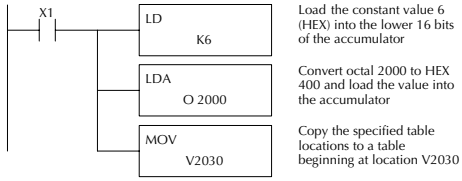
Step 2 Load the starting V-memory location for the locations to be moved into the accumulator. This parameter is a HEX value.

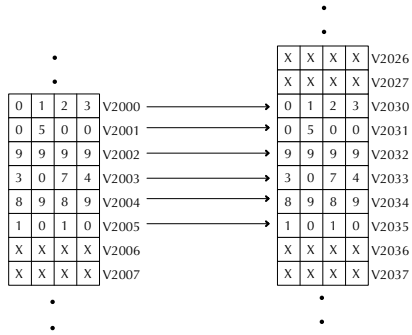
Step 3 Insert the MOV instruction which specifies starting V-memory location (Vaaa) for the destination table.

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		D4-454 Range
		aaa
V-memory	V	See memory map
Pointer	P	See memory map

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

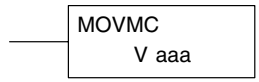




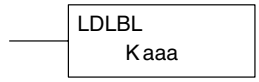
Move Memory Cartridge (MOVMC)

Load Label (LDLBL)

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



To copy data between V-memory and program ladder memory, the function parameters are loaded into the first two levels of the accumulator stack and the accumulator by two additional instructions. Listed below are the steps necessary to program the MOVMC and LDLBL 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 ladder memory and the beginning of the V-memory block into the first level of the 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 in V-memory (Vaaa). This is the copy destination.

Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map
Constant	K	1 – FFFF

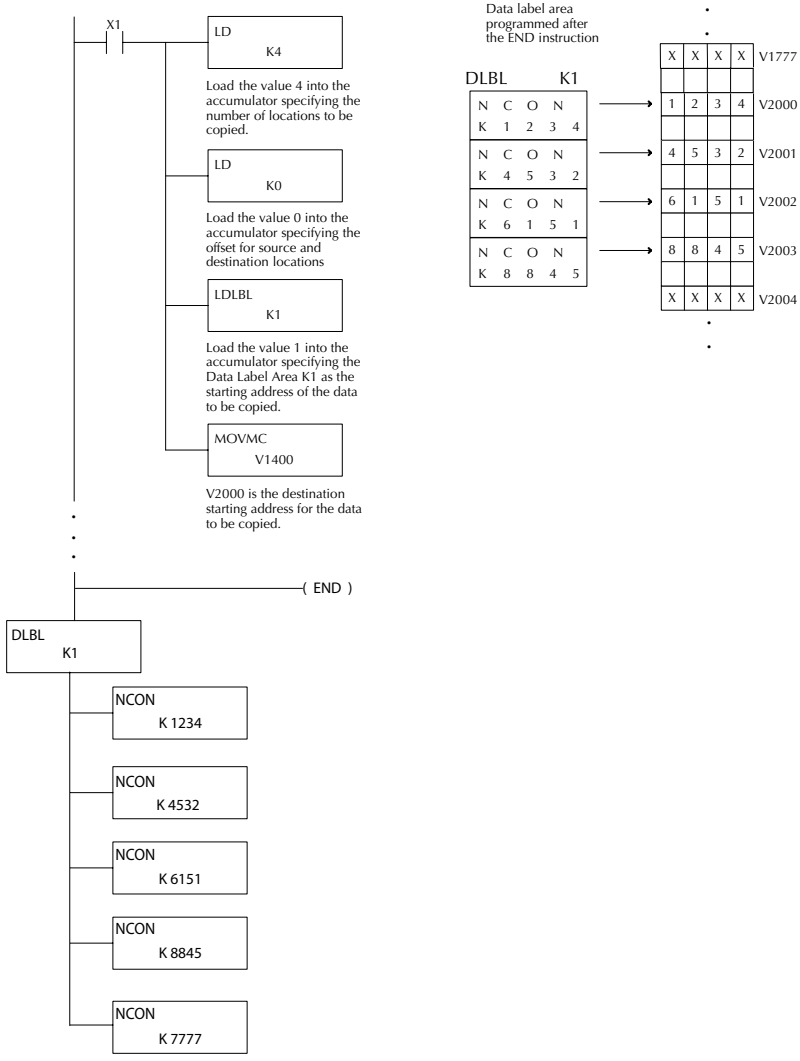
Discrete Bit Flags	Description
SP53	On if there is a table pointer error.

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

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

Copy Data From a Data Label Area to V-memory

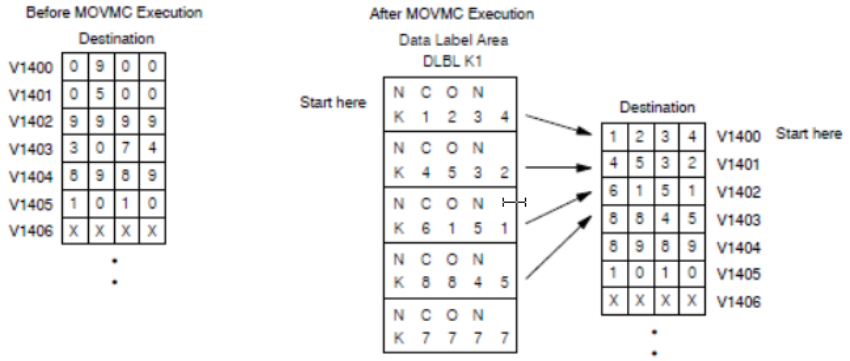
In the example below, data is copied from a Data Label Area to V-memory. When X1 is on, the constant value (K4) is loaded into the accumulator using the Load (LD) instruction. This value specifies the length of the table and is placed in the second stack location after the next Load and Load Label (LDLBL)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



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

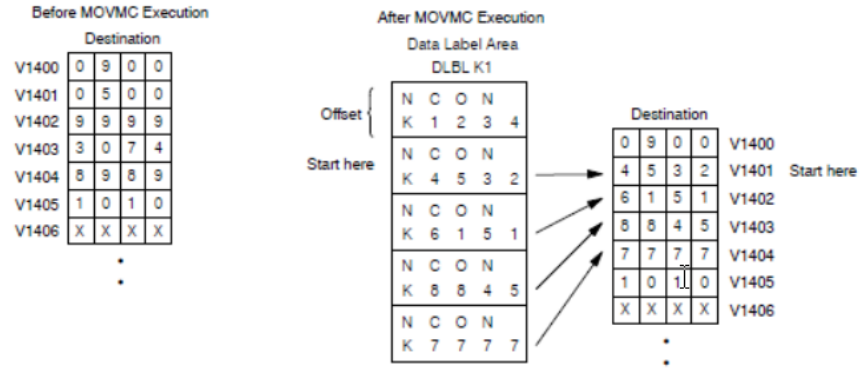
Example of Execution

Offset = 0, move 4 words

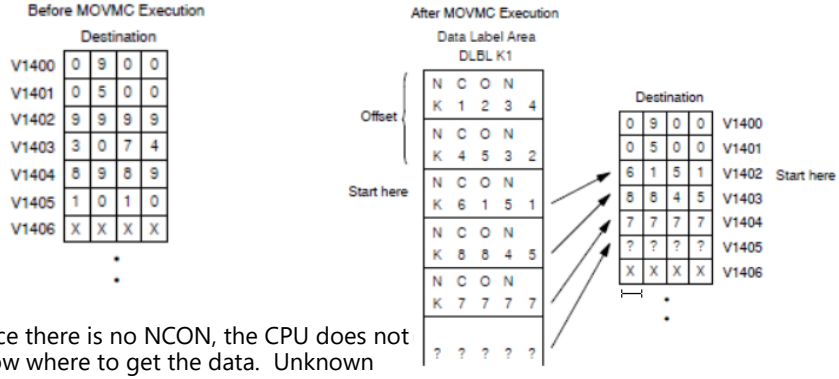


The example is fairly straightforward when an offset of zero is used. However, it also helpful for you to understand the results that would have been obtained if different offset values (1 and 2) were used. Notice how the offset is used for both the data label (source) and the destination table. Also,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

Offset = 1, move 4 words



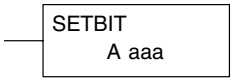
Offset = 2, move 4 words



Since there is no NCON, the CPU does not know where to get the data. Unknown values will be copied into V1405.

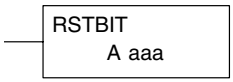
SETBIT

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



RSTBIT

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



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

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

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

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

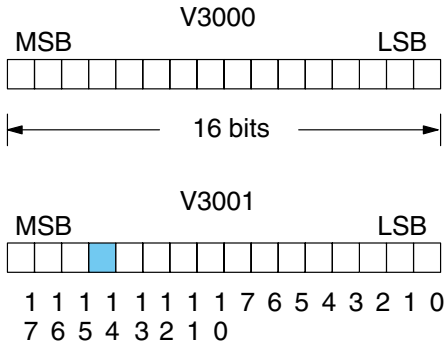
Helpful Hint: Remember that each V-memory location contains 16 bits. So, the bits of the first word of the table are numbered from 0 to 17 octal. For example, if the table length is six words, then 6 words = (6 x 16) bits, = 96 bits (decimal), or 140 octal. The permissible range of bit reference numbers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

Operand Data Type		D4-454 Range
		aaa
V-memory	V	See memory map

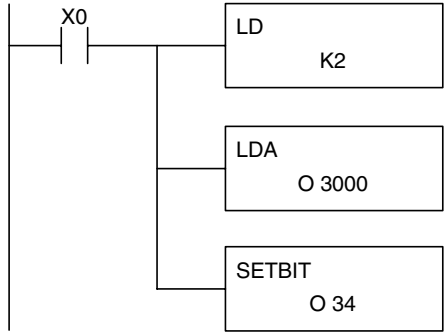
Discrete Bit Flags	Description
SP53	On when the specified bit is outside the range of the table.

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

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



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

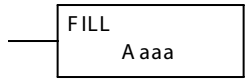


Load the constant value 2 (Hex.) into the lower 16 bits of the accumulator.

Convert octal 3000 to HEX and load the value into the accumulator. This is the table beginning.

Set bit 34 (octal) in the table to a ”1”.

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



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

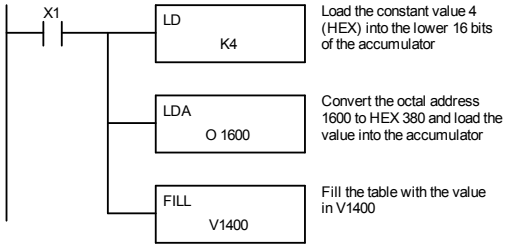
Step 2: Load the starting V-memory location for the table into the accumulator. This parameter must be a HEX value.

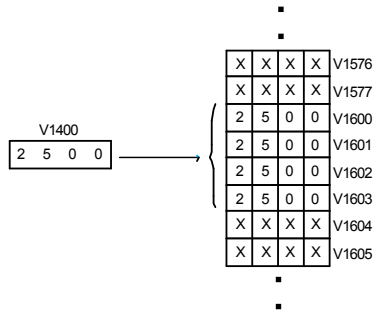
Step 3: Insert the Fill instruction which specifies the value to fill the table with. Helpful Hint: For parameters that require HEX values when referencing memory locations, the LDA instruction can be used to convert an octal address to the HEX equivalent and load the value into the accumulator.

Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Constant	K	0 – FF

Discrete Bit Flags	Description
SP53	On if the V-memory address is out of range.

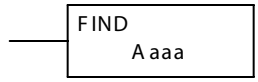
In the following example, when X1 is on, the constant value (K4) is loaded into the accumulator using the Load instruction. This value specifies the length of the table and is placed on the first level of the accumulator stack when the Load Address instruction is executed. The octal address 1600 (V1600) 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.





Find (FIND)

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



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

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

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

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

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

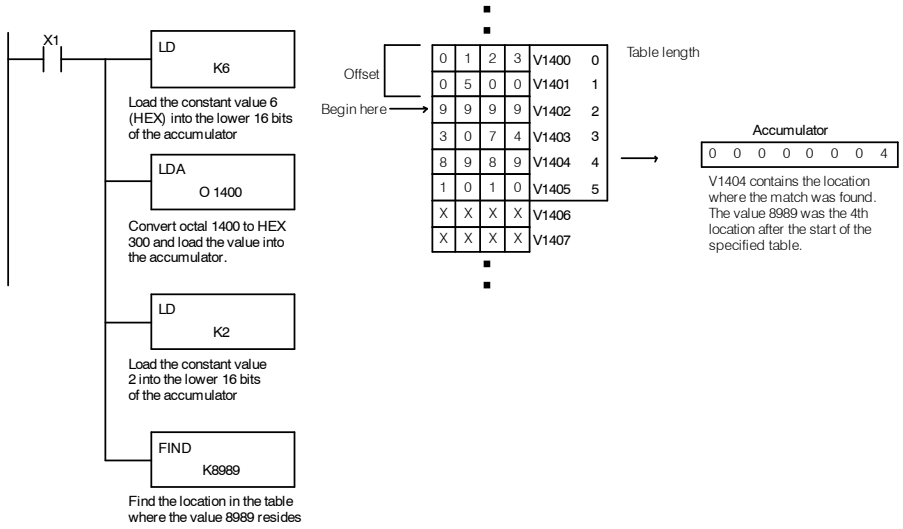
Helpful Hint: For parameters that require HEX values when referencing memory locations, the LDA instruction can be used to convert an octal address to the HEX equivalent and load the value into the accumulator.

Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map
Constant	K	0 – FF

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

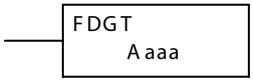
NOTE: *Status flags are only valid until another instruction that uses the same flags is executed. The pointer for this instruction starts at 0 and resides in the accumulator.*

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



Find Greater Than (FDGT)

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.



Step 1: Load the length of the table (up to 255 locations) into the first level of the accumulator stack. This parameter must be a HEX value, 0 – FF.

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

Step 3: Insert the FDGT 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 (in HEX) which is returned to the accumulator. 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.

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

Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map
Constant	K	0 – FF

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.

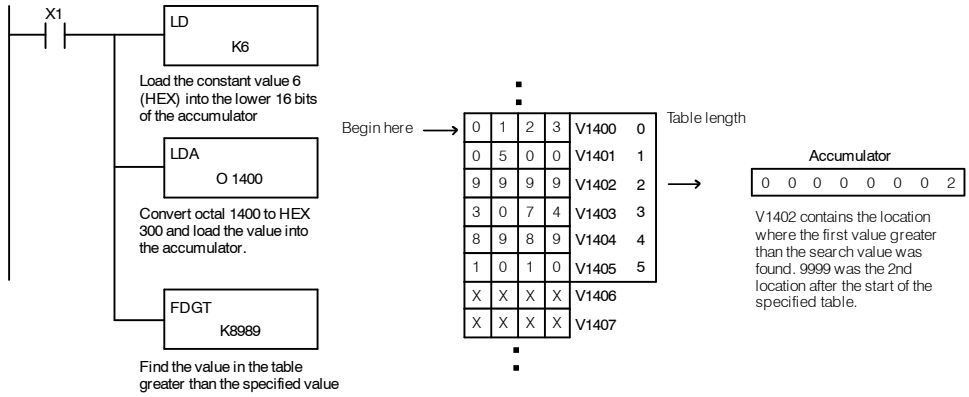
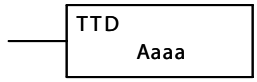


Table to Destination (TTD)

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



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

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

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

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

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

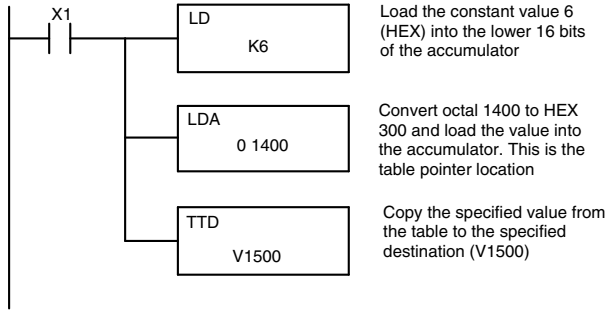
Helpful Hint: The pointer location should be set to the value where the table operation will begin. The special relay SP0 or a one-shot (PD) should be used so the value will only be set in one scan and will not affect the instruction operation.

Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map

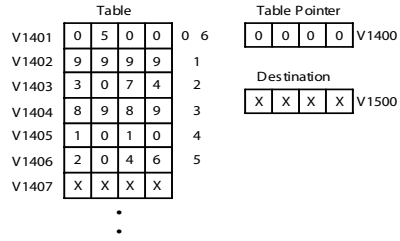
Discrete Bit Flags	Description
SP56	On when the table pointer equals the table length.

NOTE: *Status flags (SPs) are only valid until another instruction that uses the same flag is executed, or the end of the scan. The pointer for this instruction starts at 0 and resets when the table length is reached. At first glance it may appear that the pointer should reset to 0. However, it resets to 1, not 0.*

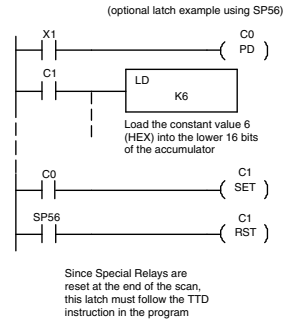
In the following example, when X1 is on, the constant value (K6) is loaded into the accumulator using the Load instruction. This value specifies the length of the table and is placed in the first stack location after the Load Address instruction is executed. The octal address 1400 (V1400) is the starting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



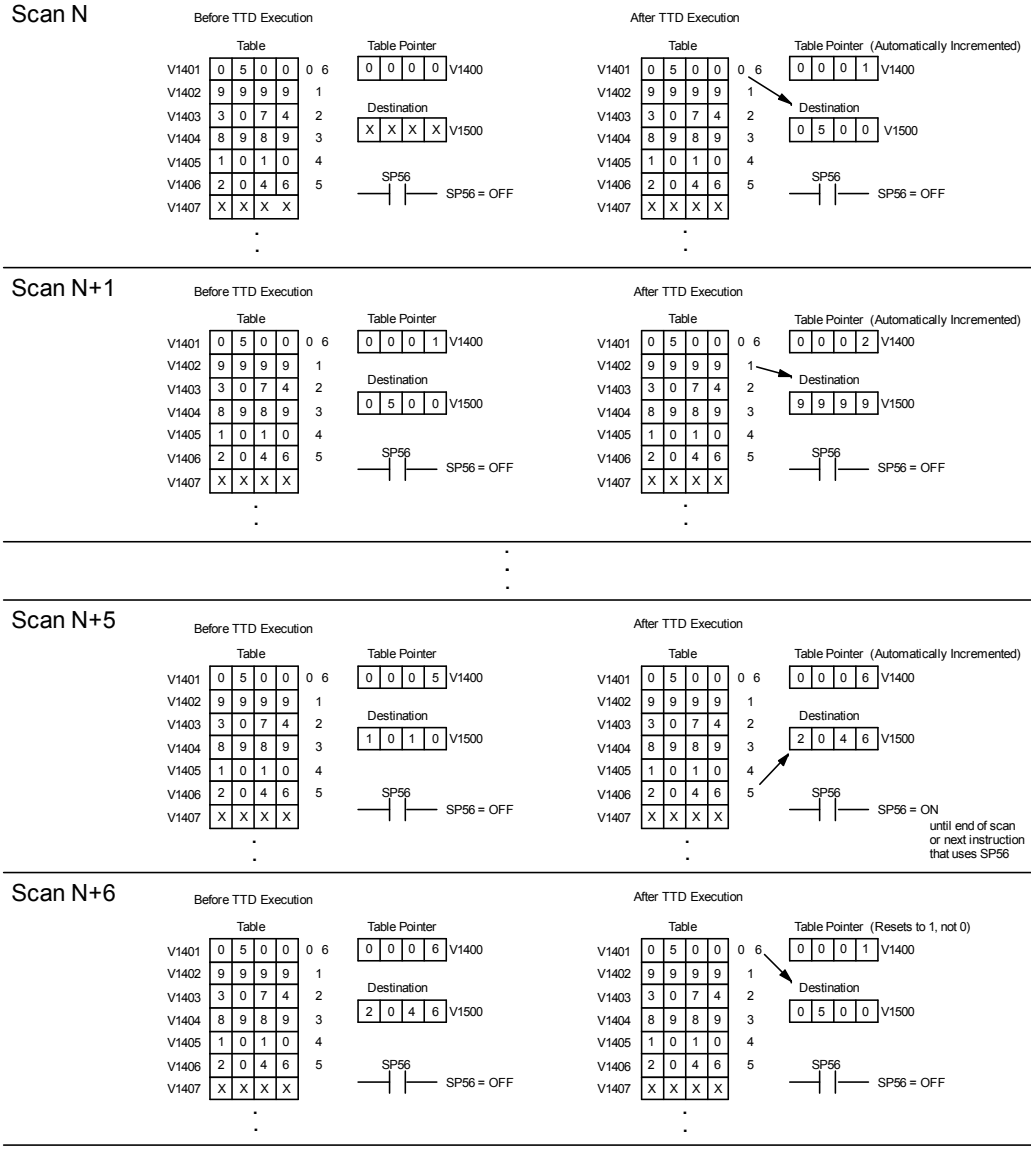
It is important to understand how the table locations are numbered. If you examine the example table, you'll notice that the first data location, V1401, will be used when the pointer is equal to zero, and again when the pointer is equal to six. Why? Because the pointer is only equal to zero before the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



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



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



Remove from Bottom (RFB)

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

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

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

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

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

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

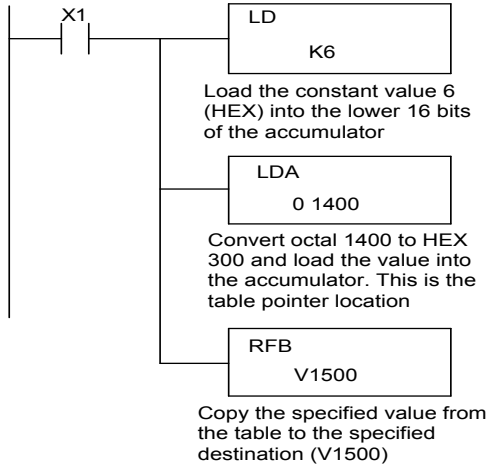
Helpful Hint: The pointer location should be set to the value where the table operation will begin. The special relay SP0 or a one-shot (PD) should be used so the value will only be set in one scan and will not affect the instruction operation.

Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map

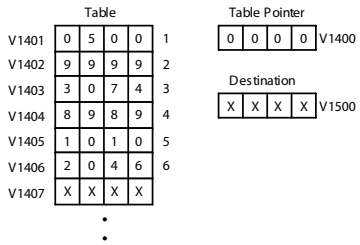
Discrete Bit Flags	Description
SP56	On when the table pointer equals zero.

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 must load a value into the pointer somewhere in your program.*

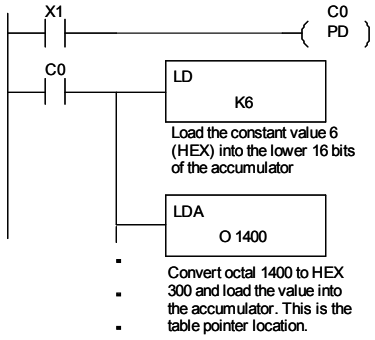
In the following example, when X1 is on, the constant value (K6) is loaded into the accumulator using the Load instruction. This value specifies the length of the table and is placed in the first stack location after the Load Address instruction is executed. The octal address 1400 (V1400) is the starting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



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



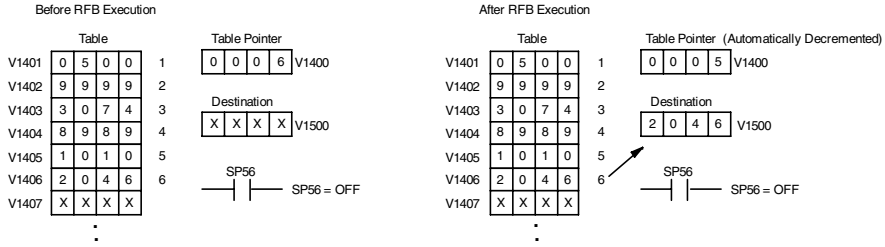
Also, our example uses a normal input contact (X1) to control the execution. Since the CPU scan is extremely fast, and the pointer decrements automatically, the table would cycle through the locations very quickly. If this is a problem for your application, you have an option of using a one-shot (PD)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



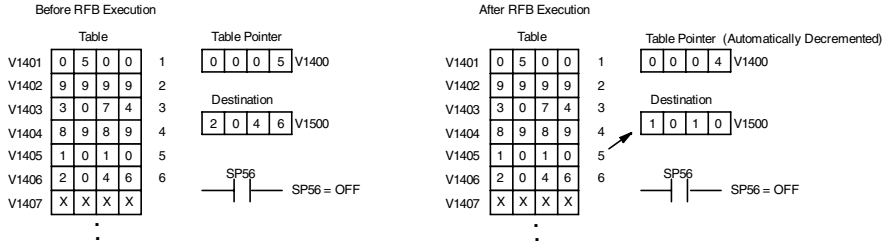
The following diagram shows the scan-by-scan results of the execution for our example program. Notice how the pointer automatically decrements from 6 to 0. Also, notice how SP56 is only on until the end of the scan.

Example of Execution

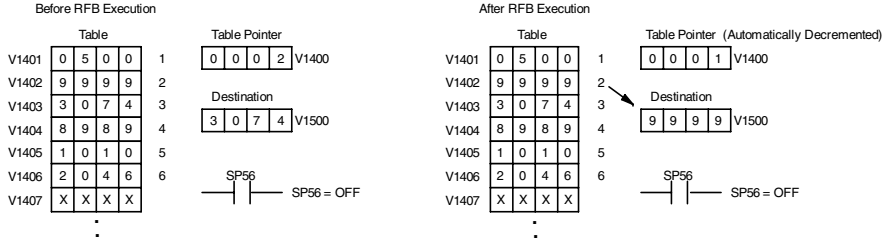
Scan N



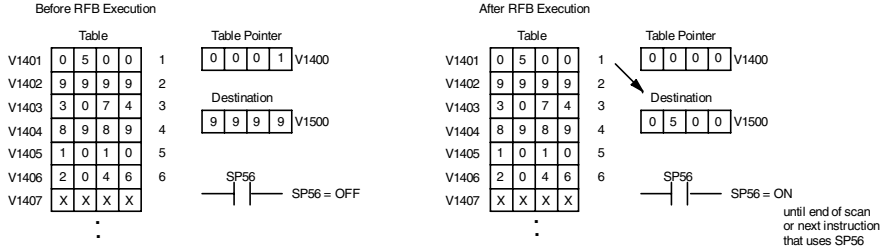
Scan N+1



Scan N+4

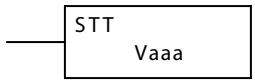


Scan N+5



Source to Table (STT)

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



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

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

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

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

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

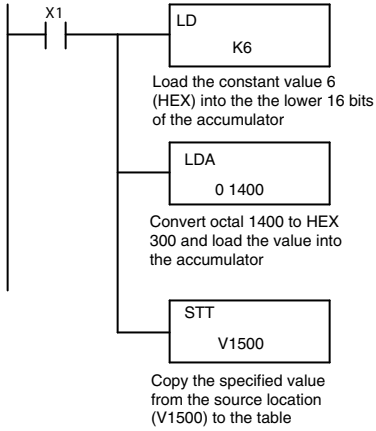
Helpful Hint: The table counter value should be set to indicate the starting point for the operation. Also, it must be set to a value that is within the length of the table. For example, if the table is 6 words long, then the allowable range of values that could be in the pointer should be between 0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

Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map

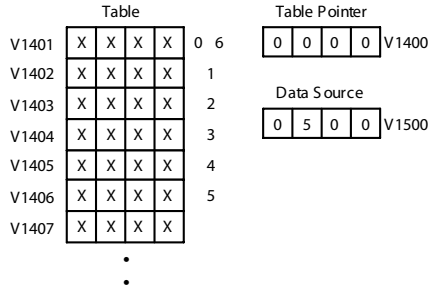
Discrete Bit Flags	Description
SP56	On when the table pointer equals the table length.

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.*

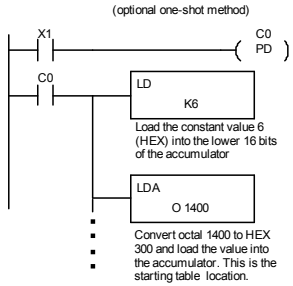
In the following example, when X1 is on, the constant value (K6) is loaded into the accumulator using the Load instruction. This value specifies the length of the table and is placed in the first stack location after the Load Address instruction is executed. The octal address 1400 (V1400), which is the 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.



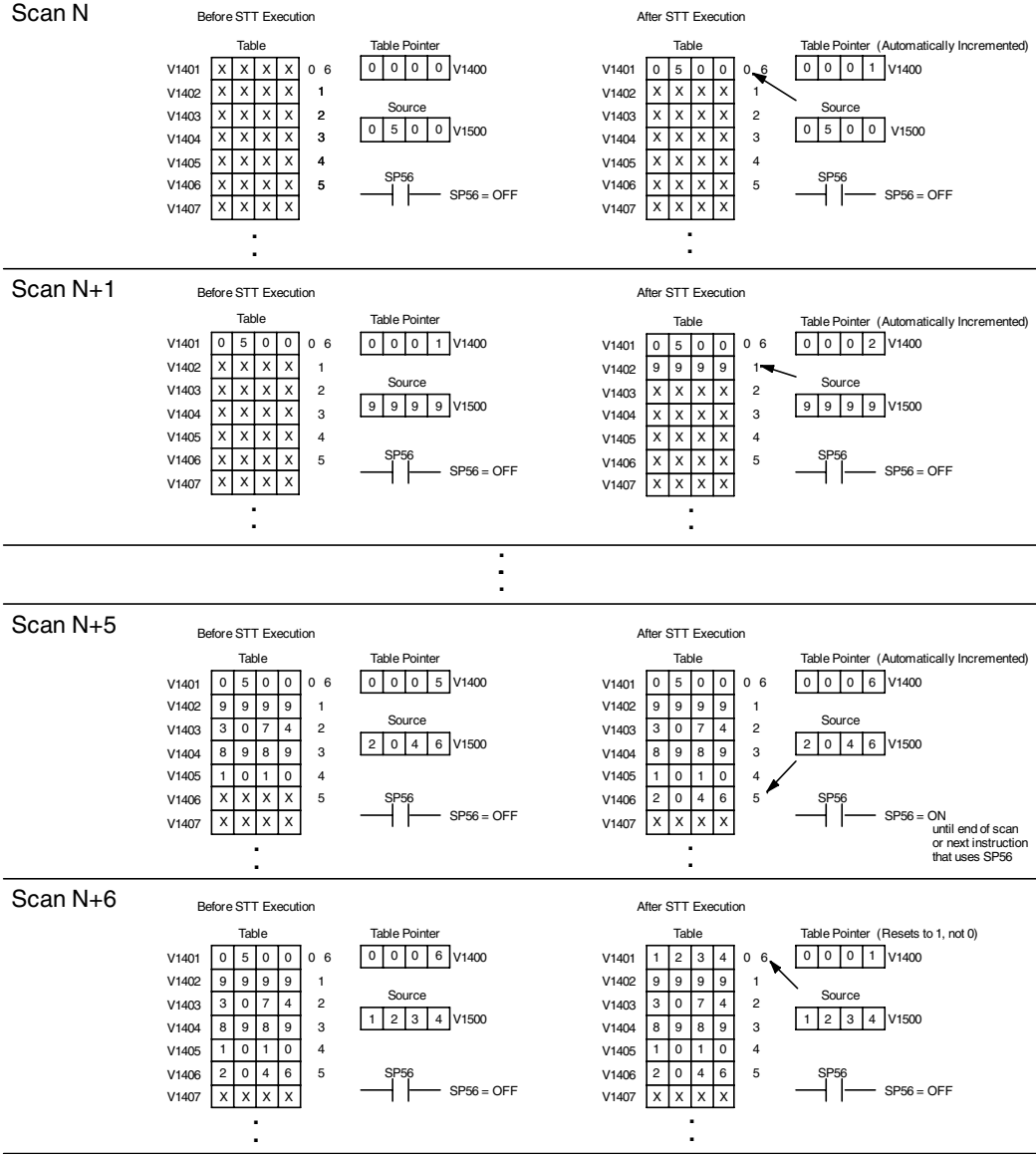
It is important to understand how the table locations are numbered. If you examine the example table, you'll notice that the first data storage location, V1401, will be used when the pointer is equal to zero, and again when the pointer is equal to six. Why? Because the pointer is only equal to zero before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



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

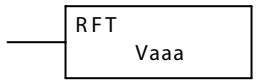


The following diagram shows the scan-by-scan results of the execution for our example program. Notice how the pointer automatically cycles from 0 to 6, and then starts over at 1 instead of 0. Also, notice how SP56 is affected by the execution. Although our example does not show it, we are assuming that 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.



Remove from Table (RFT)

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



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

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

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

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

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

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

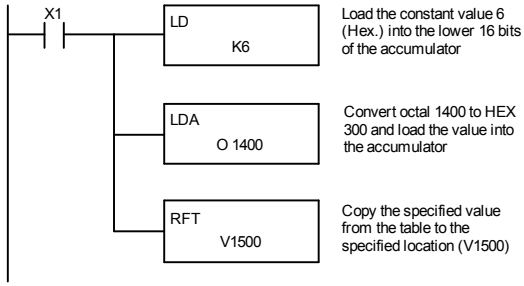
Helpful Hint: The table counter value should be set to indicate the starting point for the operation. Also, it must be set to a value that is within the length of the table. For example, if the table is 6 words long, then the allowable range of values that could be in the table counter should be between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

Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map

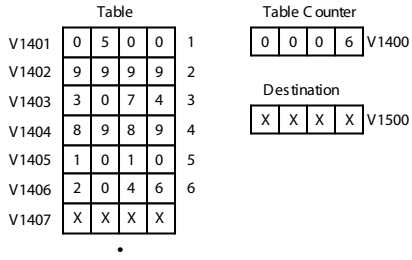
Discrete Bit Flags	Description
SP56	On when the table pointer equals zero.

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 must load a value into the pointer somewhere in your program.*

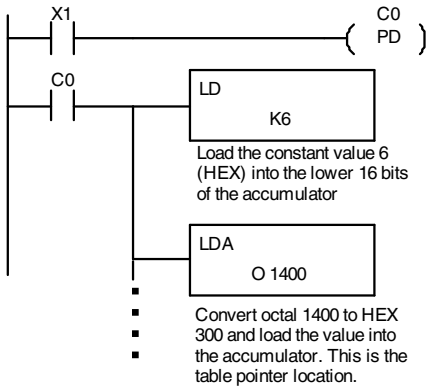
In the following example, when X1 is on, the constant value (K6) is loaded into the accumulator using the Load instruction. This value specifies the length of the table and is placed in the first stack location after the Load Address instruction is executed. The octal address 1400 (V1400) is the starting 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.



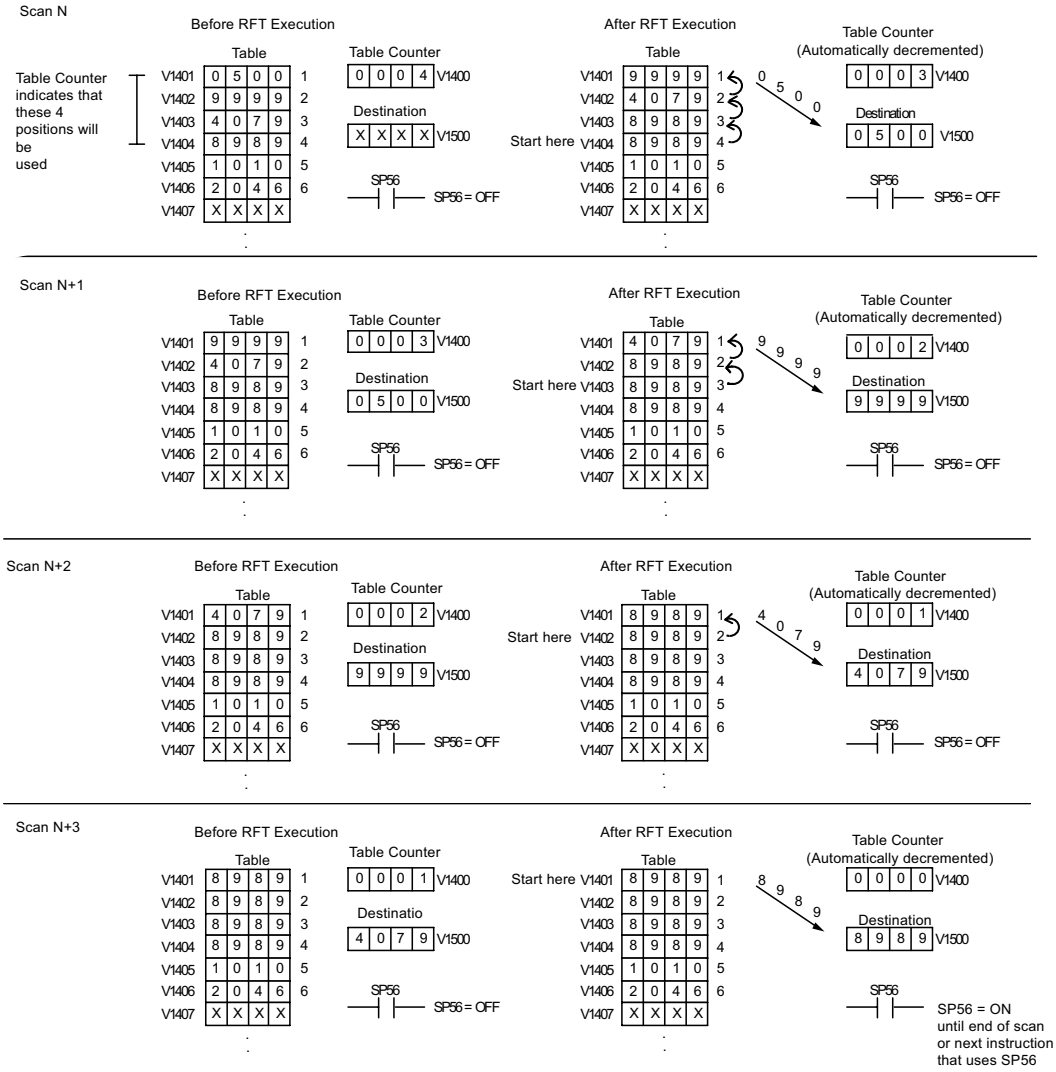
Since the table counter specifies the range of data that will be removed from the table, it is important to understand how the table locations are numbered. If you examine the example table, you'll notice that the data locations are numbered from the top of the table. For example, if the table counter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



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

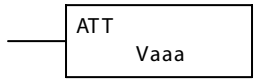


The following diagram shows the scan-by-scan results of the execution for our example program. In our example, we show the table counter set to 4, initially. (Remember, you can set the table counter to any value that is within the range of the table.) The table counter automatically decrements from 4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



Add to Top (ATT)

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



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

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

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

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

Helpful Hint: 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: 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.

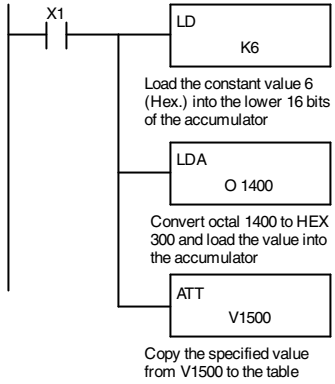
Helpful Hint: The table counter value should be set to indicate the starting point for the operation. Also, it must be set to a value that is within the length of the table. For example, if the table is 6 words long, then the allowable range of values that could be in the table counter should be between 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		D4-454 Range
	A	aaa
V-memory	V	See memory map

Discrete Bit Flags	Description
SP56	On when the table pointer equal to the table size.

NOTE: *Status flags (SPs) are only valid until another instruction that uses the same flag is executed or the end of the scan. The pointer for this instruction can be set to start anywhere in the table. It is not set automatically. You must 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.

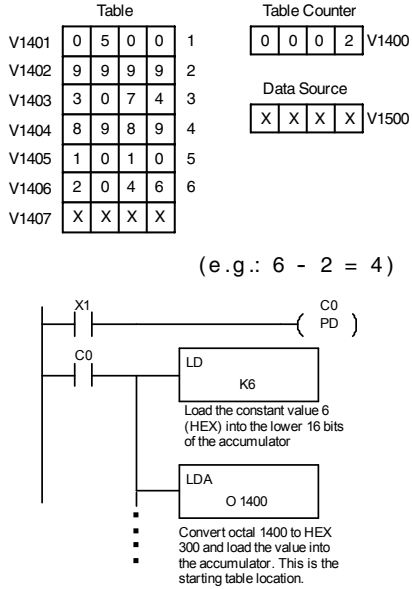


For the ATT instruction, the table counter determines the number of additions that can be made before the instruction will stop executing. So, it is helpful to understand how the system uses this counter to control the execution.

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 easily be calculated by:

Table length – table counter = number of executions

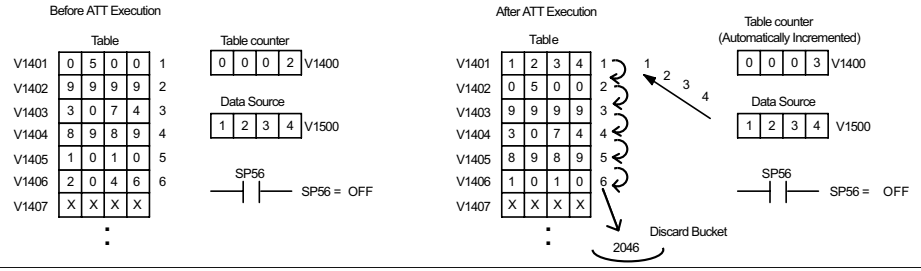
Also, our example uses a normal input contact (X1) to control the execution. Since the CPU scan is extremely fast, and the table counter increments automatically, the data would be moved into the table very quickly. If this is a problem for your application, you have an option of using a one-shot (PD)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



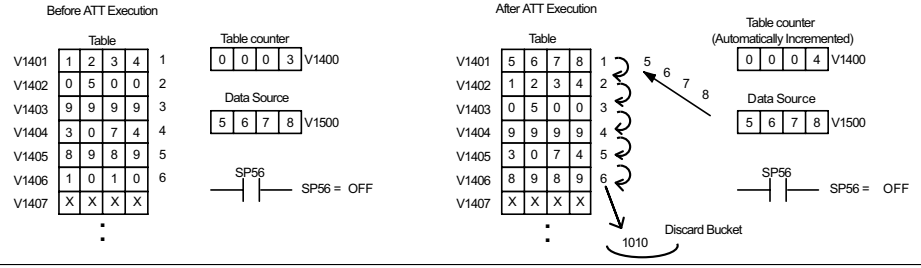
The following diagram shows the scan-by-scan results of the execution for our example program. The table counter is set to 2 initially, and it will automatically increment from 2 to 6 as the instruction is executed. Notice how SP56 comes on when the table counter is 6, which is equal to the table length. 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

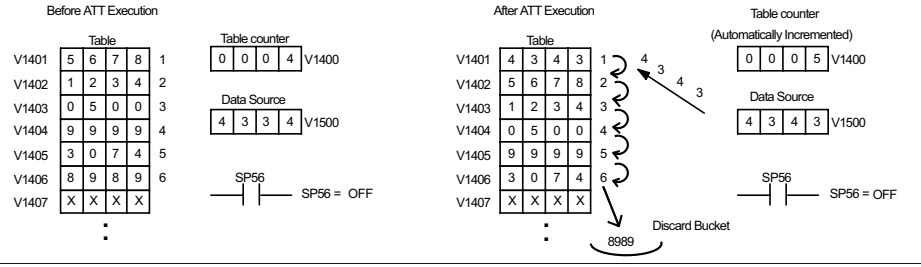
Scan N



Scan N+1



Scan N+2



Scan N+3

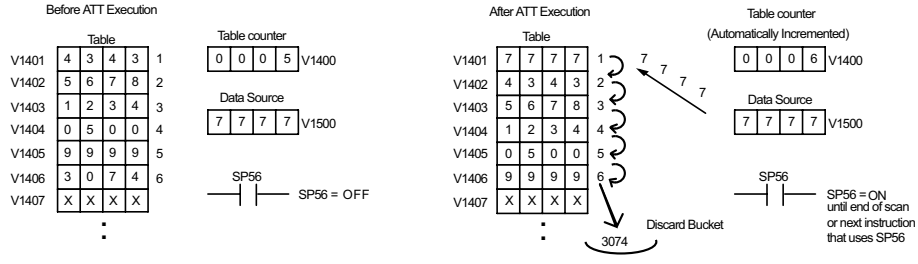


Table Shift Left (TSHFL)

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

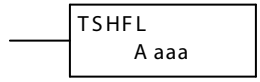
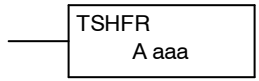
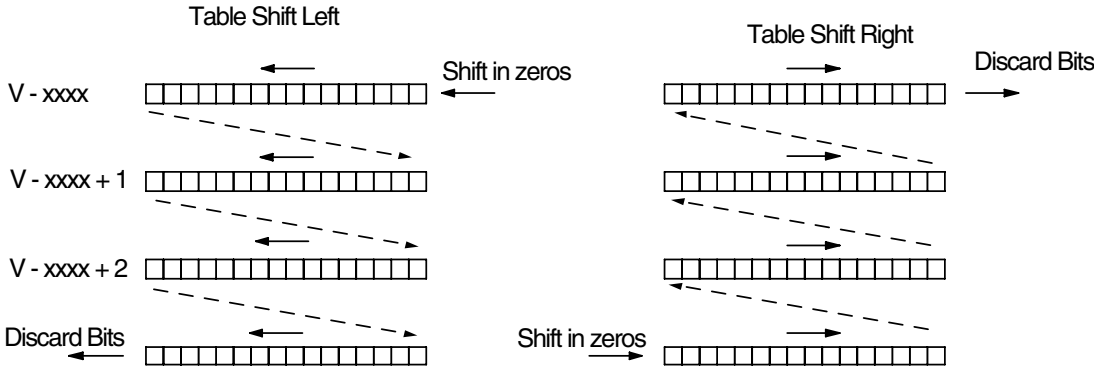


Table Shift Right (TSHFR)

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



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



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

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

Step 3: Insert the Table Shift Left or Table shift Right instruction. This specifies the number of bit positions you wish to shift the entire table. The number of bit positions must be in octal.

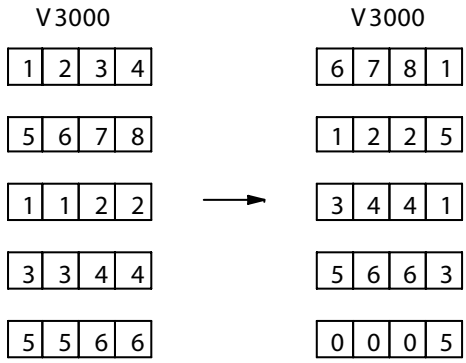
Helpful hint: Remember that each V-memory location contains 16 bits. So, the bits of the first word of the table are numbered from 0 to 17 octal. If you want to shift the entire table by 20 bits, that is 24 octal. SP 53 will be set if the number of bits to be shifted is larger than the total bits contained within the table. SP 67 will be set if the last bit shifted (just before it is discarded) is a "1".

Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map

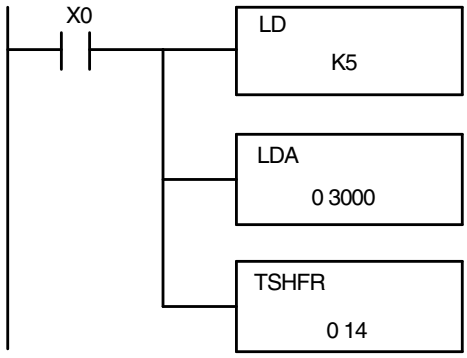
Discrete Bit Flags	Description
SP53	On when the number of bits to be shifted is larger than the total bits contained within the table
SP67	On when the last bit shifted (just before it is discarded) is a 1

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.



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.



Load the constant value 5 (Hex.) into the lower 16 bits of the accumulator.

Convert octal 3000 to HEX and load the value into the accumulator. This is the table beginning.

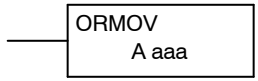
Do a table shift right by 12 bits, which is 14 octal.

AND Move (ANDMOV)

The AND Move instruction copies data from a table to the specified memory location, ANDing each word with the accumulator data as it is written.

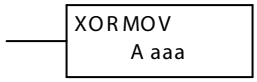
OR Move (ORMOV)

The Or Move instruction copies data from a table to the specified memory location, ORing each word with the accumulator contents as it is written.



Exclusive OR Move (XORMOV)

The Exclusive OR Move instruction copies data from a table to the specified memory location, XORing each word with the accumulator value as it is written.



The following description applies to the AND Move, OR Move, and Exclusive OR Move instructions. A table is just a range of V-memory locations. These instructions copy the data of a table to another specified location, preforming a logical operation on each word with the accumulator contents as the new table is written.

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

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

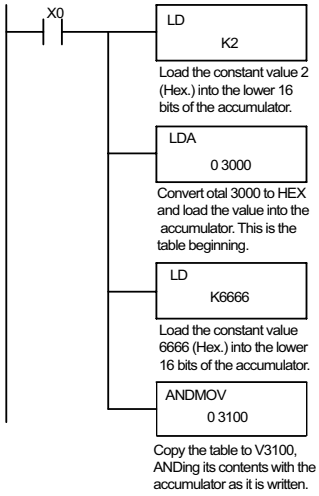
Step 3: Load the BCD/hex bit pattern into the accumulator which will be logically combined with the table contents as they are copied.

Step 4: Insert the AND Move, OR Move, or XOR Move instruction. This specifies the starting location of the copy of the original table. This new table will automatically be the same length as the original table.

Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map

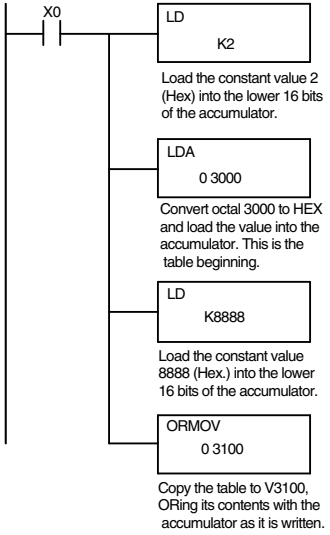
The example table below contains BCD data as shown (for demonstration purposes). Suppose we want to move a table of two words at V3000 and AND it with K6666. The copy of the table at V3100 shows the result of the AND operation for each word.

The program to the right 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.



The example below 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.

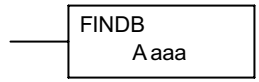


The example to the right shows a table of two words at V3000 and logically XORs it with K3333. The copy of the table at V3100 shows the result of the XOR operation for each word.

The ladder program example for the XORMOV is similar to the one above for the ORMOV. Just use the XORMOV instruction.

Find Block (FINDB)

The Find Block instruction searches for an occurrence of a specified block of values in a V-memory table. The function parameters are loaded into the first and second levels of the accumulator stack and the accumulator by three additional instructions. If the block is found, its starting address will be stored in the accumulator. If the block is not found, flag SP53 will be set.



Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map
V-memory	P	See memory map

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

The steps listed below are the steps necessary to program the Find Block function.

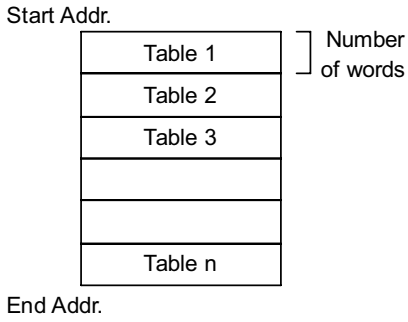
Step 1: Load the number of bytes in the block to be located. This parameter must be a HEX value, 0 to FF.

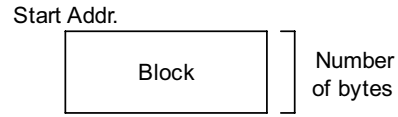
Step 2: Load the length of a table (number of words) to be searched. The Find Block will search multiple tables that are adjacent in V-memory. This parameter must be a HEX value, 0 to FF.

Step 3: Load the ending location for all the tables into the accumulator. This parameter must be a HEX value. You can use the LDA instruction to convert an octal address to hex.

Step 4: Load the table starting location for all the tables into the accumulator. This parameter must be a HEX value. You can use the LDA instruction to convert an octal address to hex.

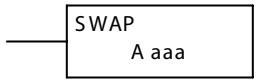
Step 5: Insert the Find Block instruction. This specifies the starting location of the block of data you are trying to locate.





Swap (SWAP)

The Swap instruction exchanges the data in two tables of equal length.



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

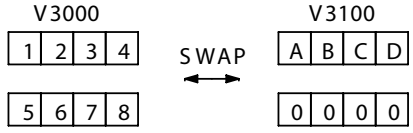
Step 2: Load the starting V-memory location for the first table into the accumulator. This parameter must be a HEX value. You can use the LDA instruction to convert an octal address to hex.

Step 3: Insert the Swap instruction. This specifies the starting address of the second table. This parameter must be a HEX value. You can use the LDA instruction to convert an octal address to hex.

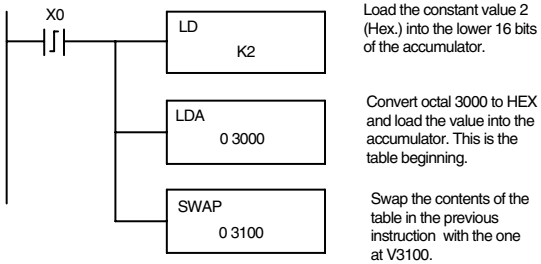
Helpful hint: The data swap occurs within a single scan. If the instruction executes on multiple consecutive scans, it will be difficult to know the actual contents of either table at any particular time. So, remember to swap just on a single scan.

Operand Data Type		D4-454 Range
		aaa
V-memory	V	See memory map

The example to the right shows a table of two words at V3000. We will swap its contents with another table of two words at 3100 by using the Swap instruction. The required ladder program is given below.



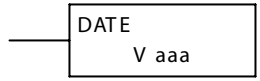
The example program below uses a PD contact (triggers for one scan for off-to-on transition). First, we load the length of the tables (two words) into the accumulator. Then we load the address of the first table (V3000) into the accumulator using the LDA instruction, converting the octal address to hex. Note that it does not matter which table we declare "first", because the swap results will be the same.



Clock/Calendar Instructions

Date (DATE)

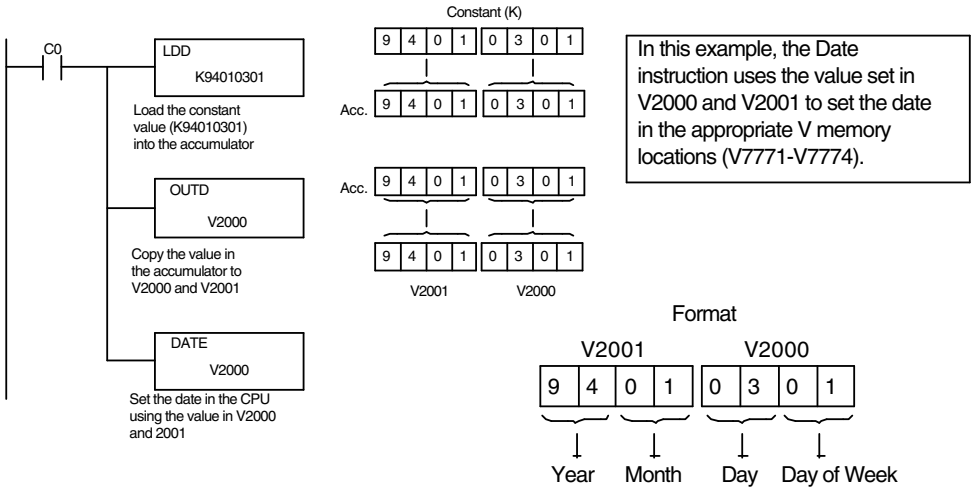
The Date instruction can be used to set the date in the CPU. The instruction requires two consecutive V-memory locations (Vaaa) to set the date. If the values in the specified locations are not valid, the date will not be set. The current date can be read from 4 consecutive V-memory locations (V7771 – V7774).



Date	Range	V-memory Location (BCD) (READ Only)
Year	0-99	V7774
Month	1-12	V7773
Day	1-31	V7772
Day of Week	0-06	V7771
The values entered for the day of week are: 0=Sunday, 1=Monday, 2=Tuesday, 3=Wednesday, 4=Thursday, 5=Friday, 6=Saturday		

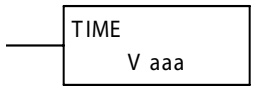
Operand Data Type		D4-454 Range
		aaa
V-memory	V	See memory map

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.



Time (TIME)

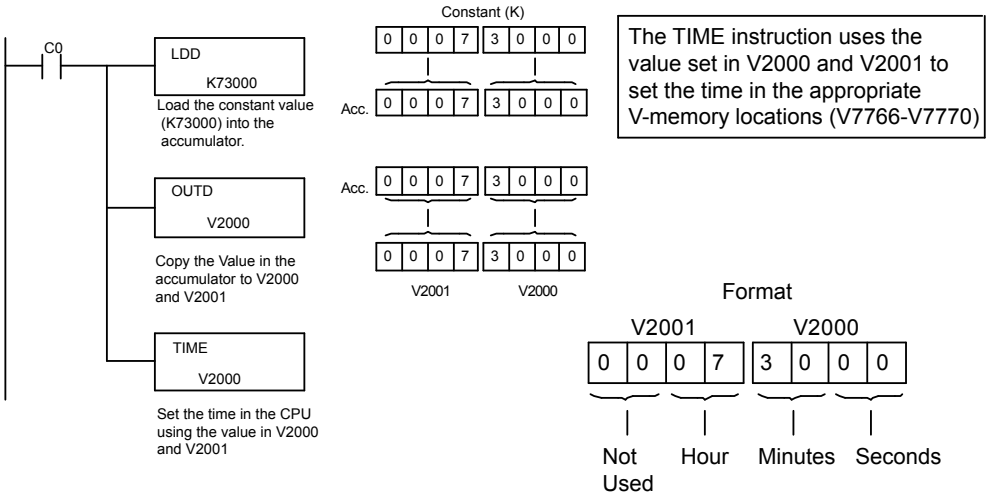
The Time instruction can be used to set the time (24 hour clock) in the CPU. The instruction requires two consecutive V-memory locations (Vaaa) which are used to set the time. If the values in the specified locations are not valid, the time will not be set. The current time can be read from memory locations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



Date	Range	V-Memory Location (BCD) (READ Only)
1/100 seconds (10ms)	0-99	V7747
Seconds	0-59	V7766
Minutes	0-59	V7767
Hour	0-23	V7770

Operand Data Type		D4-454 Range
		aaa
V-memory	V	See memory map

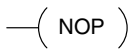
In the following example, when C0 is on, the constant value (K73000) is loaded into the accumulator using the Load Double instruction (C0 should be a contact from a one-shot (PD) instruction). The value in the accumulator is output to V2000 using the Out Double instruction. The Time instruction uses the value in V2000 to set the time in the CPU.



CPU Control Instructions

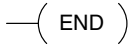
No Operation (NOP)

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



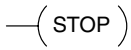
End (END)

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.



Stop (STOP)

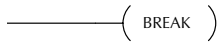
The Stop instruction changes the operational mode of the CPU from Run to Program (Stop) mode. This instruction is typically used to stop PLC operation in an error condition.

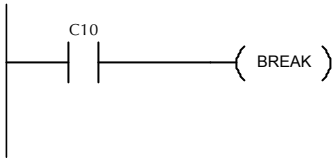


In the following example, when C0 turns on, the CPU will stop operation and switch to the program mode.

Break (BREAK)

The Break instruction changes the operational mode of the CPU from Run to Test Program mode. This instruction is typically used to aid in debugging an application program. The Break instruction allows V-memory and image register data to be retained where it would be normally cleared with the Stop instruction or a normal Run to Program transition.





Reset Watch Dog Timer (RSTWT)

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

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

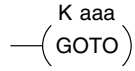
If the scan time is consistently longer than the watch dog timer's setting, the timeout value may be increased from the default value of 200ms with the DirectSOFT programming software. This eliminates the need for the RSTWT instruction.

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.

Program Control Instructions

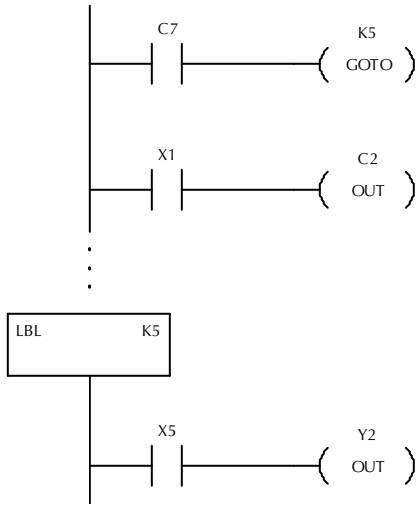
Goto Label (GOTO) (LBL)

The Goto / Label skips all instructions between the Goto and the corresponding LBL instruction. The operand value for the Goto and the corresponding LBL instruction are the same. The logic between Goto and LBL instruction is not executed when the Goto instruction is enabled. Up to 256 Goto instructions and 256 LBL instructions can be used in the program.



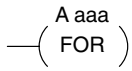
Operand Data Type		D4-454 Range
		aaa
Constant	K	1-256

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.



For / Next (FOR) (NEXT)

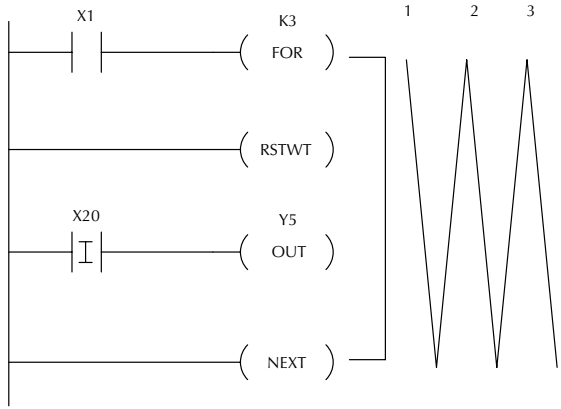
The For and Next instructions are used to execute a section of ladder logic between the For and Next instruction a specified numbers of times. When the For instruction is enabled, the program will loop the specified number of times. If the For instruction is not energized, the section of ladder logic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



For / Next instructions cannot be nested. The normal I/O update and CPU housekeeping are suspended while executing the For / Next loop. The program scan can increase significantly, depending on the amount of times the logic between the For and Next instruction is executed. With the exception of immediate I/O instructions, I/O will not be updated until the program execution is completed for that scan. Depending on the length of time required to complete the program execution, it may be necessary to reset the watch dog timer inside of the For / Next loop using the RSTWT instruction.

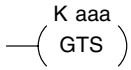
Operand Data Type		D4-454 Range
		aaa
V-memory	V	See memory map
Constant	K	1-9999

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



Goto Subroutine (GTS) (SBR)

The Goto Subroutine instruction allows a section of ladder logic to be placed outside the main body of the program, to execute only when needed. You can have unlimited number of GTS instructions. Upon completion of executing the subroutine, program execution returns to the main program immediately after the GTS instruction. 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. There can be a maximum of 256 SBR instructions used in the program. When the subroutine is called from the main program, the CPU will execute the subroutine (SBR) with the same constant number (K) as the GTS instruction which called the subroutine.

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

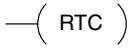
Operand Data Type		D4-454 Range
		aaa
Constant	K	1-FFFF

Subroutine Return (RT)

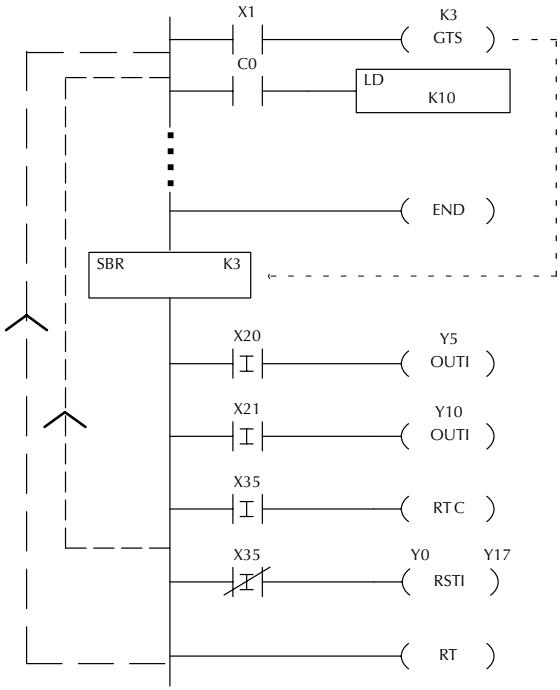
When a Subroutine Return is executed in the subroutine the CPU will return to the point in the main body of the program from which it was called. The Subroutine Return is used as termination of the subroutine. It must be the last instruction in the subroutine and is a stand alone instruction (no input contact on the rung).

Subroutine Return Conditional (RTC)

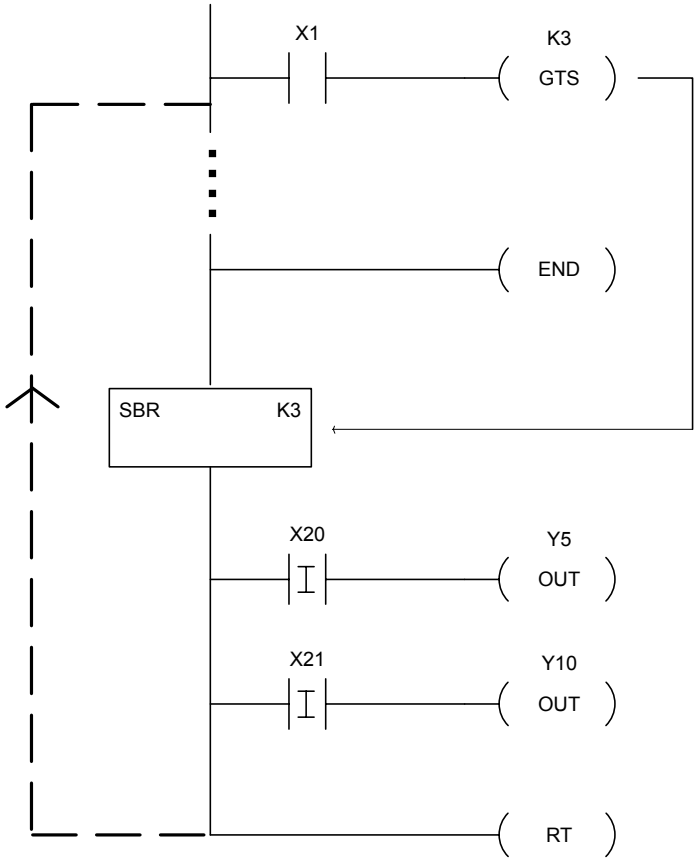
The Subroutine Return Conditional instruction is an optional instruction used with an input contact to implement a conditional return from the subroutine. The Subroutine Return (RT) is still required for termination of the Subroutine.



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 subroutine will be executed. If X35 is on, the CPU will return to the main program at the RTC instruction. If X35 is not on, Y0 – Y17 will be reset to off and the CPU will return to the main body of the program.

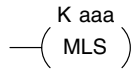


In the following example, when X1 is on, Subroutine K3 will be called. The CPU will jump to the Subroutine Label K3 and the ladder logic in the subroutine will be executed. The CPU will return to the main body of the program after the RT instruction is executed.



Master Line Set (MLS)

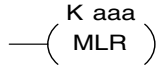
The Master Line Set instruction allows the program to control sections of ladder logic by forming a new power rail controlled by the main left power rail. The main left rail is always master line 0. When an MLS K1 instruction is used, a new power rail is created at level 1. Master Line Sets and Master Line Resets can be used to nest power rails up to seven levels deep.



Operand Data Type		D4-454 Range
		aaa
Constant	K	1-7

Master Line Reset (MLR)

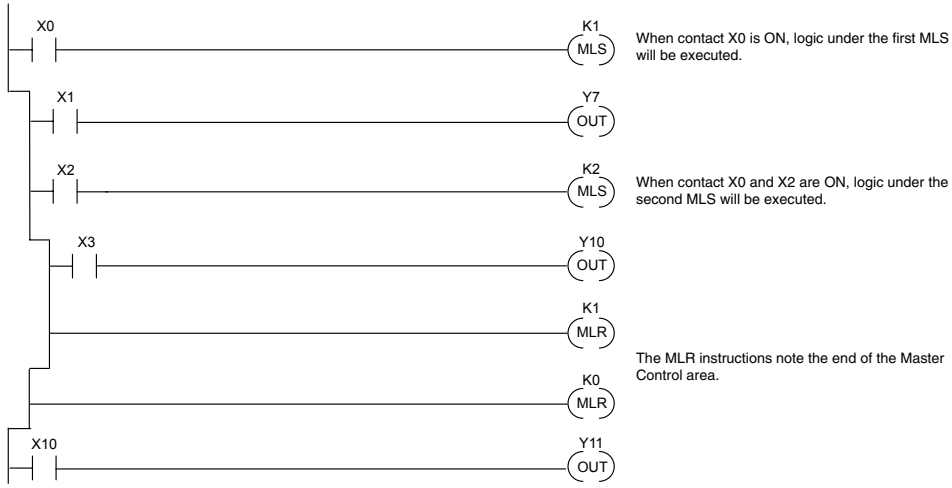
The Master Line Reset instruction marks the end of control for the corresponding MLS instruction. The MLR reference is one less than the corresponding MLS.



Operand Data Type		D4-454 Range
		aaa
Constant	K	0-6

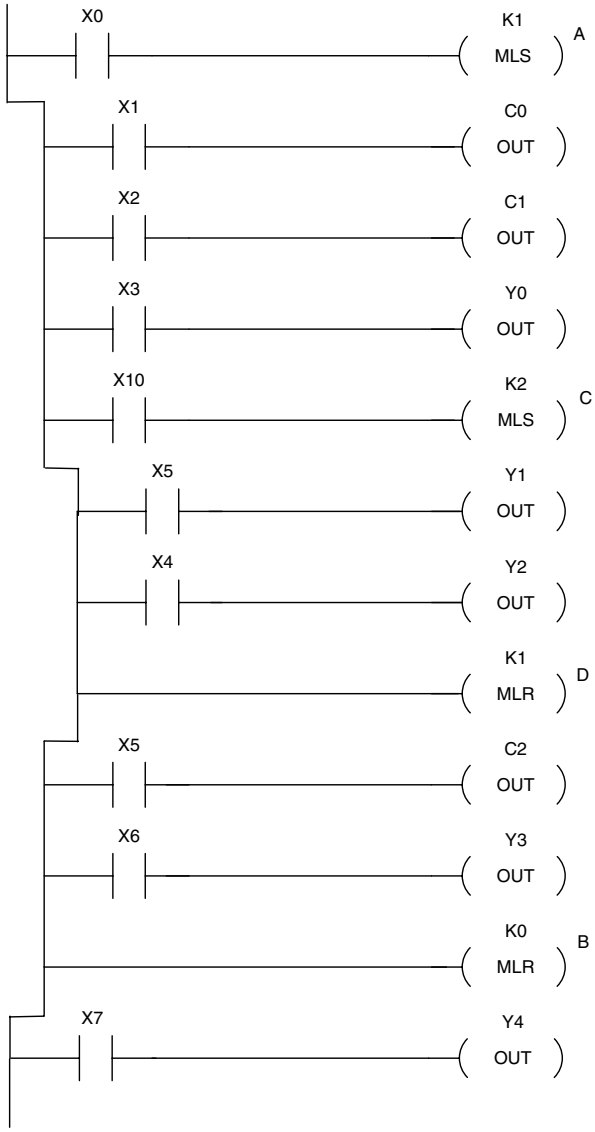
Understanding Master Control Relays

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



MLS/MLR Example

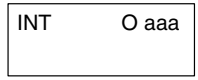
In the following MLS/MLR example logic between the first MLS K1 (A) and MLR K0 (B) will function only if input X0 is on. The logic between the MLS K2 (C) and MLR K1 (D) will function only if input X10 and X0 is on. The last rung is not controlled by either of the MLS coils.



Interrupt Instructions

Interrupt (INT)

The Interrupt instruction allows a section of ladder logic to be placed outside the main body of the program and executed when needed. Interrupts can be called from the program. 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.



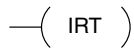
The interrupt label and all associated logic must be placed after the End statement in the program. When the interrupt routine is called, the CPU will complete execution of the instruction it is currently processing in ladder logic, then execute the interrupt routine.

There are two software interrupts and INT 16 and INT 17. This means that the hardware interrupts INT 16 and INT 17 and the software interrupts cannot be used at the same time.Once the interrupt is serviced, the program execution will continue from where it was before the interrupt occured. The software interrupts are setup by programming the interrupt times in V736 and V737. The valid range is 3 – 999ms. The value must be a BCD value. The interrupt will not execute if the value is out of range.

Operand Data Type		D4-454 Range
		aaa
Constant	O	0-17

Interrupt Return (IRT)

An Interrupt Return is normally executed as the last instruction in the interrupt routine. It returns the CPU to the point in the main program from which it was called. The Interrupt Return is a stand-alone instruction (no input contact on the rung).

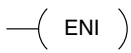


Interrupt Return Conditional (IRTC)

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

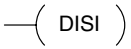
Enable Interrupts (ENI)

The Enable Interrupt instruction is placed in the main ladder program (before the End instruction), enabling the interrupt. The interrupt remains enabled until the program executes a Disable Interrupt instruction.



Disable Interrupts (DISI)

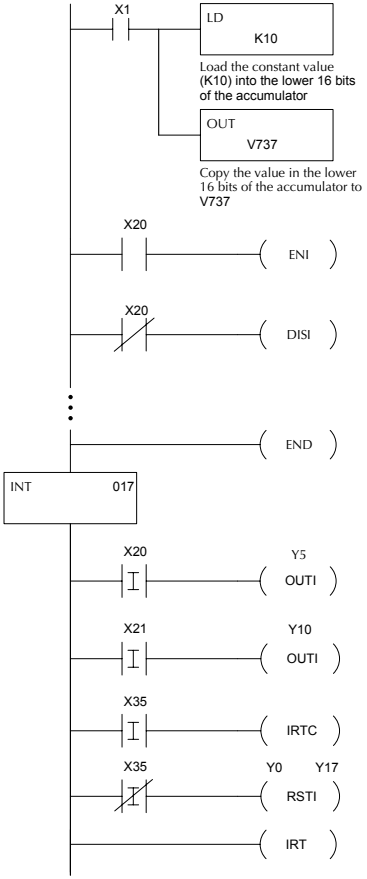
A Disable Interrupt instruction in the main body of the application program (before the End instruction) will disable the interrupt (either external or timed). The interrupt remains disabled until the program executes an Enable Interrupt instruction.



Interrupt Example for Software Interrupt

In the following example, when X1 is on, the value 10 is copied to V737. This value sets the software interrupt to 10 ms. When X20 turns on, the interrupt will be enabled. When X20 turns off, the interrupts will be disabled. Every 10 ms the CPU will jump to the interrupt label INT 017. The application ladder logic in the interrupt routine will be performed. If X35 is on the CPU will return to the main program with the IRTC instruction. If X35 is not on Y0 – Y17 will be reset to off and then the CPU will return to the program when the IRT instruction is executed. The software interrupt is limited to the range 3 – 999 milliseconds. Entering a 0, 1, or 2 will yield an interrupt time of 3 milliseconds.

NOTE: *To take advantage of the speed of an Interrupt routine, use Immediate instructions for inputs and outputs. For example, Store immediate (STRI), Out immediate (OUTI) and Set immediate (SETI).*



Message Instructions

System Errors and Fault Messages

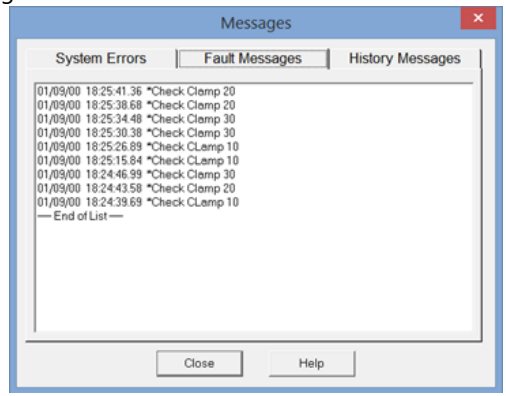
The D4-454 CPUs provide error logging capabilities. There are certain predefined system error messages and codes, but you can also use the Fault instruction to create your own specific messages. The CPU logs the error, the date, and the time the error occurred. There are two separate tables that store this information.

System Error Table -- The D4-454 can show up to 32 errors in an error table. When an error occurs, the error is loaded into the first available location. Therefore, the most recent error may not appear in the top row of the table. If the table is full when an error occurs, the oldest error is pushed (erased) from the table and the new error is inserted in the row.

Fault Message Table -- the D4-454 also allow you to build your own error codes and messages. These are called Fault Messages. You can build error codes, or up to 16 messages that can contain up to 23-character alphanumeric characters. In either case, you can have up to 16 messages or codes shown in the table. When a message is triggered, it is put in the first available table location. Therefore, the most recent error may not appear in the top row of the table. If the table is full when an error occurs, the oldest error is pushed (erased) from the table and the new error is inserted in the row.

The following diagram shows an example of a Fault Message table as shown in the DirectSOFT programming software.

There are several instructions that can be used in combination to create these error codes and fault messages.

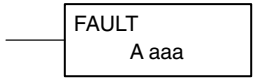


- FAULT -- Fault
- DLBL -- Data Label
- ACON -- ASCII Constant
- NCON -- Numeric Constant

The next few pages provide details on these instructions. Also, at the end of this section, there are two examples that show how the instructions are used together.

Fault (FAULT)

The Fault instruction is used to display a message or numeric error code in DirectSOFT, or on an Operator Interface Panel. The message has a maximum of 23 characters and can be either V-memory data, numerical constant data or ASCII text.



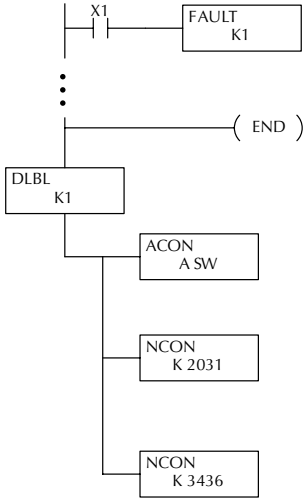
To display the value in a V-memory location, specify the V-memory location in the instruction. To display ASCII or numeric data, you must use the DLBL (Data Label) instruction, and ACON (ASCII constant) or NCON (Numeric constant) instructions, in conjunction with the Fault instruction. In this case, you should specify the constant (K) value in the Fault instruction for the corresponding data label area that contains the ACON and/or NCON instructions.

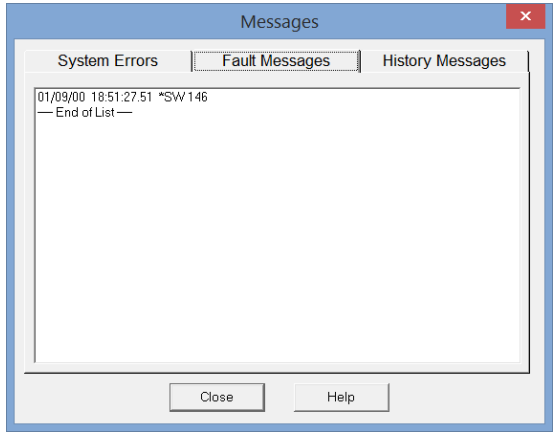
Operand Data Type		D4-454 Range
		aaa
V-memory	V	See memory map
Constant	K	1-FFFF

Discrete Bit Flags	Description
SP50	On when the FAULT instruction is executed

Fault Example

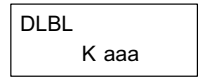
In the following example when X1 is on, the message SW 146 will be logged in the fault message table. The NCON use the HEX ASCII equivalent of the text to be displayed. (HEX ASCII for a blank is 20, a 1 is 31, 4 is 34 and 6 is 36).





Data Label (DLBL)

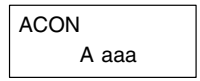
The Data Label instruction marks the beginning of an ASCII/ numeric data area. DLBLs are programmed after the End statement. A maximum of 64 DLBL instructions can be used in a program. Multiple NCONs and ACONs can be used in a DLBL area.



Operand Data Type		D4-454 Range
		aaa
Constant	K	1-64

ASCII Constant (ACON)

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

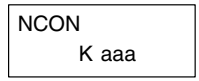


Operand Data Type		D4-454 Range
		aaa
ASCII	A	0-9 A-Z

When using DirectSOFT programming software, it is possible to store up to 40 characters in an ACON. See the ASCII Table Appendix for a complete listing of ASCII characters available. When the 40 characters are downloaded to the CPU, they are actually broken down into multiple ACONs that contain 2 characters each.

Numerical Constant (NCON)

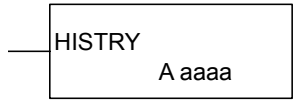
The Numerical Constant instruction is used with the DLBL instruction to store the HEX ASCII equivalent of numerical data for use with other instructions. Two digits can be stored in an NCON instruction.



Operand Data Type		D4-454 Range
		aaa
Constant	K	0-FFFF

History (HISTRY)

The History instruction stores event history information in memory of the PLC.

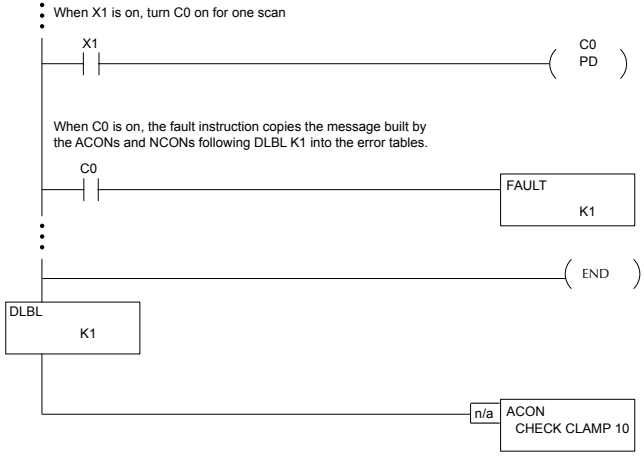


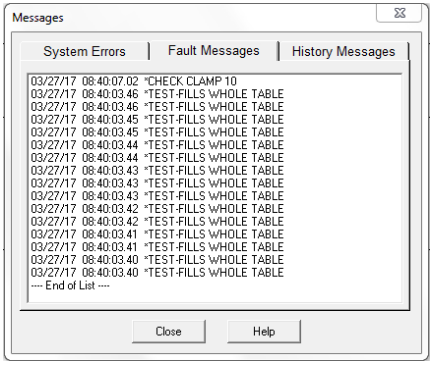
Important Information about FAULT Execution

It is important to understand how the Error Message and Error Code tables work when the Fault instruction is executed. Each time the instruction is executed, the code or message is inserted into the table. Since the CPU scan is extremely fast, then it is easily possible to completely fill up a table with a single message repeated each scan. In many cases this is not desirable, since it's nice to maintain a history of the errors. (That's why there are 32 positions in the Error Code table and 16 positions in the Error Message table.)

For example, let's say you are examining a limit switch (X1). When the switch is closed, you want an error message (CHECK CLAMP 10) to be logged into the Error message table. In the real world, the limit switch may be closed for several seconds (or minutes, or hours...). During this time, the CPU will execute the Fault instruction a number of times, so the entire Error Message table will be full of a single message.

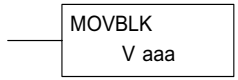
How do you solve this problem? Simple. Instead of using the limit switch to trigger the Fault instruction, use the limit switch to trigger a control relay used as a one-shot (PD coil instruction). Then, use the control relay as the input to the Fault instruction. Since the Fault is now triggered by the PD, which is only on for one scan, the message will only be copied into the table one time. This keeps the history of older messages intact.





Move Block (MOVBLK)

The Move Block instruction copies a specified number of words from a Data Label Area of program memory (ACON, NCON) to the specified V-memory location. Below are the steps for using the Move Block function:



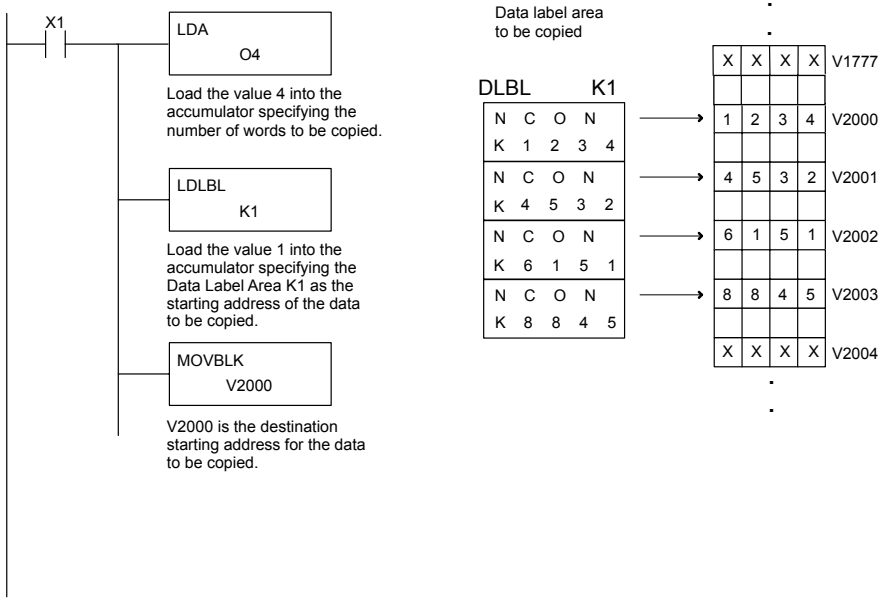
Step 1: Load the number of words (octal) to be copied into the 1st level of the accumulator stack.

Step 2: Load the source data label (LDLBL Kaaa) into the accumulator. This is where the data will be copied from.

Step 3: Insert the MOVBLK instruction that specifies the V-memory destination. This is where the data will be copied to.

Copy Data From a Data Label Area to V-memory

In the example below, data is copied from a Data Label Area to V-memory. When X1 is on, the octal value (O4) is copied to the first level of the accumulator stack using the Load Address (LDA) instruction. This value specifies the number of words to be copied. Load Label (LDLBL) instruction will load the source data address (K1) into the accumulator. This is where the data will be copied from. The MOVBLK instruction specifies the destination starting location and executes the copying of data from the Data Label Area to V-memory.



Print Message (PRINT)

The Print Message instruction prints the embedded text or text/data variable message (maximum 128 characters) to the specified communications port (Port 1, 2 and/or 3 on the D4-454 CPU), which must have the communications port configured.

```
PRINT     A aaa
"Hello, this is a PLC message"
```

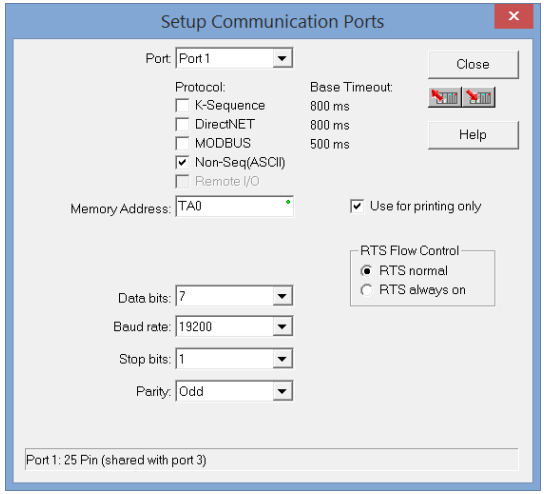
Operand Data Type		D4-454 Range
		aaa
Constant	K	1, 2 or 3

You may recall, from the CPU specifications in Chapter 3, that the DL454's ports are capable of several protocols. Port 1, 2 and 3 can be configured for the non-sequence protocol. To configure a port in DirectSOFT, choose the PLC menu, then Setup, then Setup Secondary Comm Port.

For example,

Port: From the port number list box at the top, choose Port 2.

Protocol: Click the check box to the left of Non-sequence, and then you'll see the dialog box shown below.



Baud Rate: Choose the baud rate that matches your printer.

Stop Bits, Parity: Choose number of stop bits and parity setting to match your printer.

Memory Address: Choose a V-memory address for DirectSOFT to use to store the port setup information. You will need to reserve 66 contiguous words in V-memory for this purpose. IE V2000 to V2077. This area cannot be used by any other function in the ladder program.

Before ending the setup, click the button indicated to send the port configuration to the CPU, and click Close. See Chapter 3 for port wiring information, in order to connect your printer to the D4-454.

Text element – This part of the instruction is used to create the character string to be sent out of the port. The character strings are defined as the character (more than 0) ranged by the double quotation marks. Two hex numbers preceded by the dollar sign means an 8-bit ASCII character code. Also, two characters preceded by the dollar sign is interpreted according to the following table:

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

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

Example:

"" Length 0 without character

"A" Length 1 with character A

" " Length 1 with blank

" $" " Length 1 with double quotation mark

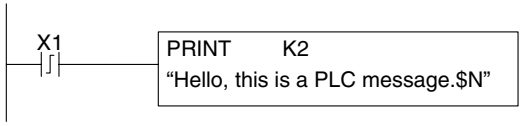
" $ R $ L " Length 2 with one CR and one LF

" $ 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 E499 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 output device. This prepares the printer to print the next line, starting from the left margin.



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

V-memory element - this is used for printing V-memory contents in the integer format or real format. Use V-memory number or V-memory number with ":" and data type. The data types are shown in the table below. The Character code must be 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 E499.*

Example:

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

V2000 Prints binary data in V2000 for decimal number

V2000:B Prints BCD data in V2000

V2000:D Prints binary number in V2000 and V2001 for decimal number

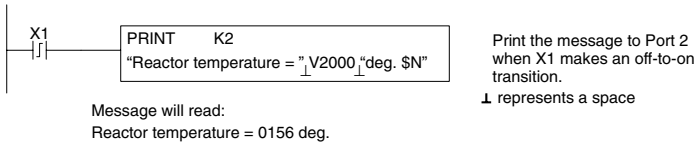
V2000:DB Prints BCD data in V2000 and V2001

V2000:R Prints floating point number in V2000/V2001 as real number

V2000:E Prints 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 - This is used for printing text stored in V-memory. Use the % followed by the number of characters after V-memory number for representing the text. If you assign "0" as the number of characters, the print function will read the character



count from the first location. Then it will start at the next V-memory location and read that number of ASCII codes for the text from memory.

Example:

V2000%16 16 characters in V2000 to V2007 are printed.

V2000%0 The characters in V2001 to Vxxxx (determined by the number in V2000) will be printed.

Bit element

This is used for printing the state of the designated bit in V-memory or a relay bit. The bit element can be assigned by the designating point (.) and bit number preceded by the V-memory number or relay number. The output type is described as shown in the table below.

#	Data Format	Description
1	None	Print 1 for an ON state, and 0 for an OFF state
2	:BOOL	Print "TRUE" for an ON state, and "FALSE" for an OFF state
3	:ONOFF	Print "ON" for an ON state, and "OFF" for an OFF state

Example:

V2000.15 Prints the status of bit 15 in V2000, in 1/0 format

C100 Prints the status of C100 in 1/0 format

C100:BOOL Prints the status of C100 in TRUE/FALSE format

C100:ONOFF 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:

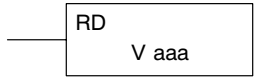
Special relay flags for port 1, 2, and 3 indicate the status of the D4-454 CPU ports (busy, or communications error). See the Special Relays appendix for a description.

Element Type	Maximum Characters
Text, 1 character	1
16 bit binary	6
32 bit binary	11
4 digit BCD	4
8 digit BCD	8
Floating point (real number)	12
Floating point (real with exponent)	12
V-memory/text	2
Bit (1/0 format)	1
Bit (TRUE/FALSE format)	5
Bit (ON/OFF format)	3

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.*

Intelligent I/O Instructions

Read from Intelligent Module (RD)



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

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

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

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

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

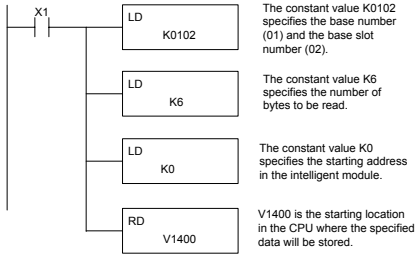
Helpful Hint: Use the LDA instruction to convert an octal address to its HEX equivalent and load it into the accumulator when the HEX format is required.

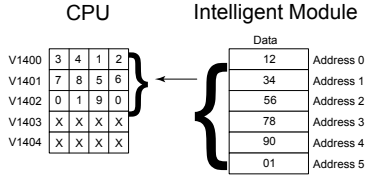
Operand Data Type		D4-454 Range
		aaa
V-memory	V	See memory map

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

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

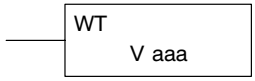
In the following example, when X1 is ON, the RD instruction will read six bytes of data from a intelligent module in base 1, slot 2, starting at address 0 in the intelligent module, and copy the information into V-memory locations V1400-V1402.





Write to Intelligent Module (WT)

The Write to Intelligent Module instruction writes a block of data (1-128 bytes maximum) to an intelligent I/O module from a block of V-memory in the CPU. The function parameters are loaded into the first and second level of the accumulator stack and the accumulator by three additional instructions.



Listed below are the steps to program the Write to Intelligent module function.

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

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

Step 3: Load the intelligent module address which will receive the data into the accumulator. This parameter must be a HEX value.

Step 4: Insert the WT instruction which specifies the starting V-memory location (Vaaa) where the data will be written from in the CPU.

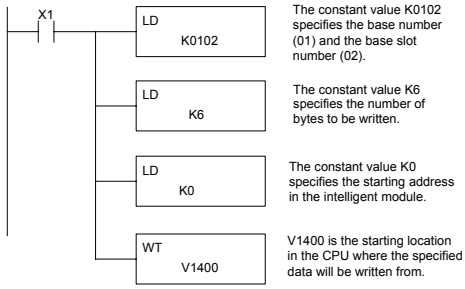
Helpful Hint: Use the LDA instruction to convert an octal address to its HEX equivalent and load it into the accumulator when the HEX format is required.

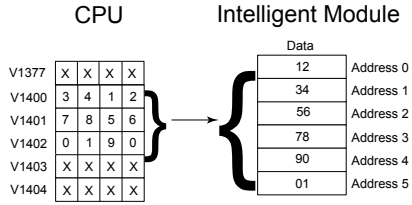
Operand Data Type		D4-454 Range
		aaa
V-memory	V	See memory map

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 data from V-memory locations V1400-V1402.

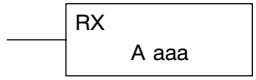




Network Instructions

Read from Network (RX)

The Read from Network instruction is used by the master device on a network to read a block of data from a slave device on the same network. The function parameters are loaded into the first and second level of the accumulator stack and the accumulator by three additional instructions.



Listed below are the steps necessary to program the Read from Network function.

Step 1: Load the slave address (0–90 BCD) into the first byte and the PLC internal port (Kf2) or slot number of the master DCM or ECOM (0–7) into the second byte of the second level of the accumulator stack.

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

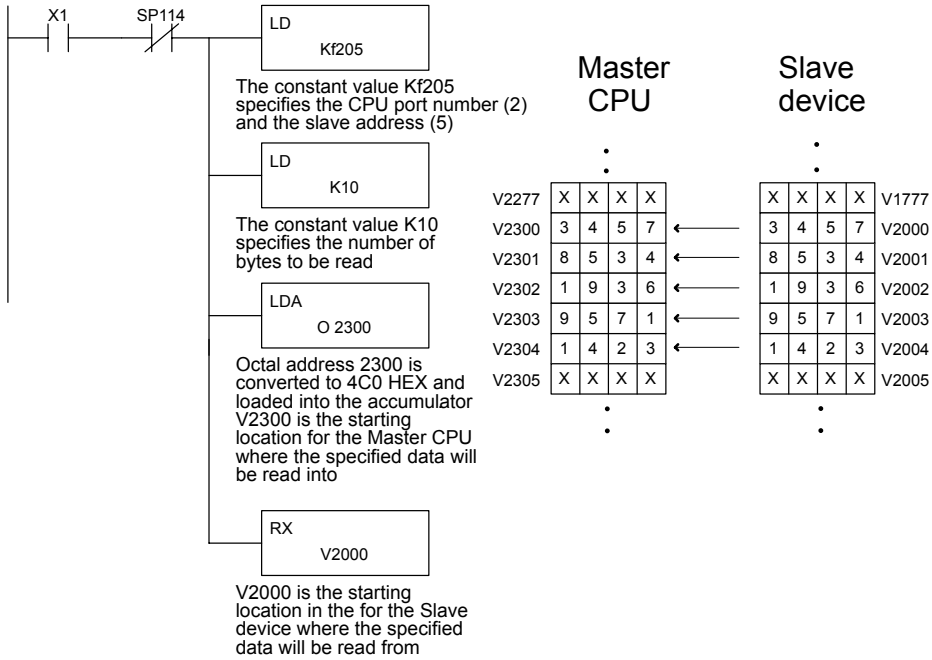
Step 3: Load the address of the data to be read into the accumulator. This parameter requires a HEX value.

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

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

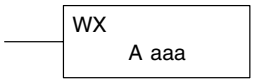
Operand Data Type		D4-454 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Inputs	X	0 – 1777
Outputs	Y	0 – 1777
Control Relays	C	0 – 3777
Stage	S	0 – 1777
Timer	T	0 – 377
Counter	CT	0 – 377
Special Relay	SP	0 – 777
Global I/O GX		0 – 377
Program Memory	$	0 – 7680 (2K program mem.)

In the following example, when X1 is on and the port busy relay SP114 (see special relays) is not on, the RX instruction will access port 2 operating as a master. Ten consecutive bytes of data (V2000 – V2004) will be read from a slave device at station address 5 and copied into V-memory locations V2300 – V2304 in the CPU.



Write to Network (WX)

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



Listed below are the program steps necessary to execute the Write to Network function.

Step 1: Load the slave address (0–90 BCD) into the low byte and "Kf2" into the high byte of the accumulator (the next two instructions push this word down to the second layer of the stack).

Step 2: Load the number of bytes to be transferred into the accumulator (the next instruction pushes this word onto the top of the stack).

Step 3: Load the starting Master CPU address into the accumulator. This is the memory location where the data will be written from. 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 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		D4-454 Range
	A	aaa
V-memory	V	See memory map
Pointer	P	See memory map
Inputs	X	0–1777
Outputs	Y	0–1777
Control Relays	C	0–3777
Stage	S	0–1777
Timer	T	0–377
Counter	CT	0–377
Special Relay	SP	0–777
Global I/O GX		0–377
Program Memory	$	0–7680 (2K program mem.)

In the following example, when X1 is on and the module busy relay SP114(see special relays) is not on, the WX instruction will access port 2 operating as a master. Ten consecutive bytes of data are read from the Master CPU and copied to V-memory locations V2000–V2004 in the slave device at station address 5.