9

Programming Basics

In This Chapter. . . .

Introduction

This chapter describes some basic programming concepts used with the DL305 CPUs. It doesn't provide detailed information on each instruction, but instead shows how you can use the most basic elements of the instruction set. If you have quite a bit of PLC programming experience, you may already know some of the information. However, we suggest you at least read the portion that discusses the accumulator operation. The accumulator is used in many different operations.

This chapter provides an overview of the following programming concepts.

1. Boolean Instructions
2. Timer Instructions
3. Counter Instructions
4. Shift Register Instruction
5. Accumulator Instructions

Detailed examples of all categories of instructions are included in Chapters 11 & 12.

The DL305 CPUs can be programmed with the *Direct*SOFT PC-based programming package, or by using the DL305 handheld programmer. There is a separate manual available for each of these products. If your are not familiar with the chosen programming device we recommend you use the appropriate programming device manual along with this manual to program your DL305 system.

The following examples will help you understand how DL305 instructions are put together to create a program solution.
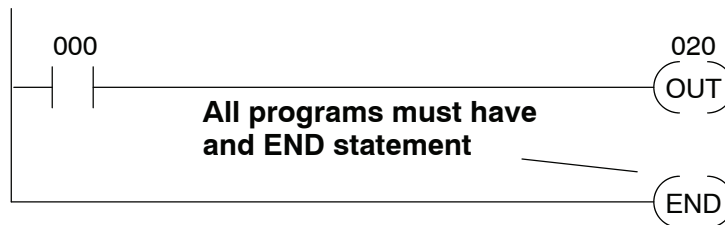
Using Boolean Instructions

Do you ever wonder why so many PLC manufacturers always quote the scan time for a 1K boolean program? Simple. Most all programs utilize many boolean instructions. These are typically very simple instructions designed to join input and output contacts in various series and parallel combinations. Since the ***Direct*SOFT** package allows you to use graphic symbols to build the program, you don't absolutely *have* to know the boolean equivalents of the instructions. However, it may be helpful at some point, especially if you ever have to troubleshoot the program with a Handheld Programmer.

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

END Statement   *All* DL305 programs require an END statement as the last instruction. This tells the CPU this is the end of the program. Any instructions placed after the END statement will not be executed. (This can be useful in some cases. See Chapter 13 for an example.)

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.
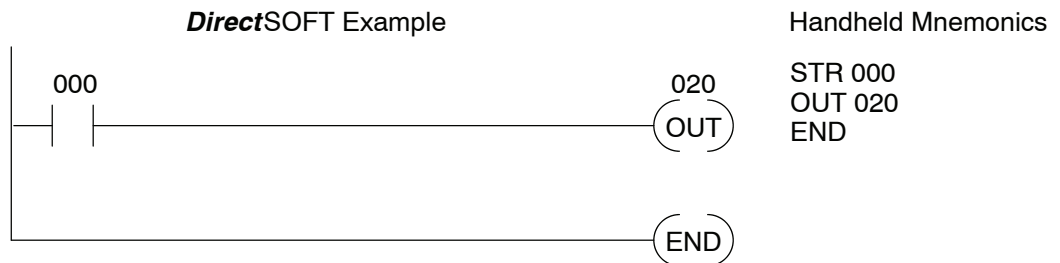
Normally Closed Contact   Normally closed contacts are also very common. This is accomplished with the Store Not or, STRN instruction. The following example shows a simple rung with a normally closed contact.
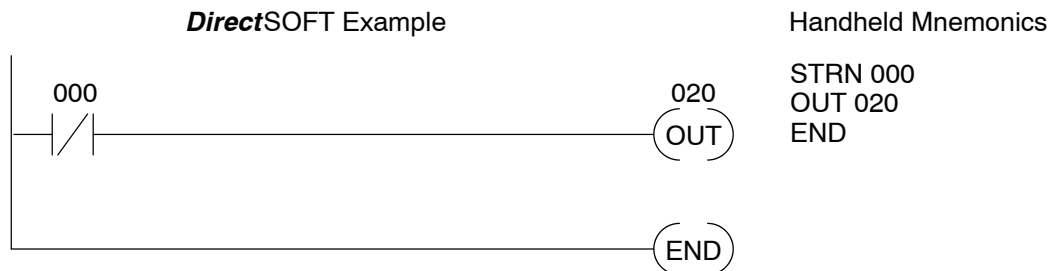
Contacts in Series   Use the AND instruction to join two or more contacts in series. The following example shows two contacts in series and a single output coil.

*Direct*SOFT Example   Handheld Mnemonics

```
  000   001                        020      STR 000
  ─┤ ├──┤ ├────────────────────────(OUT)    AND 001
                                            OUT 020
                                            END

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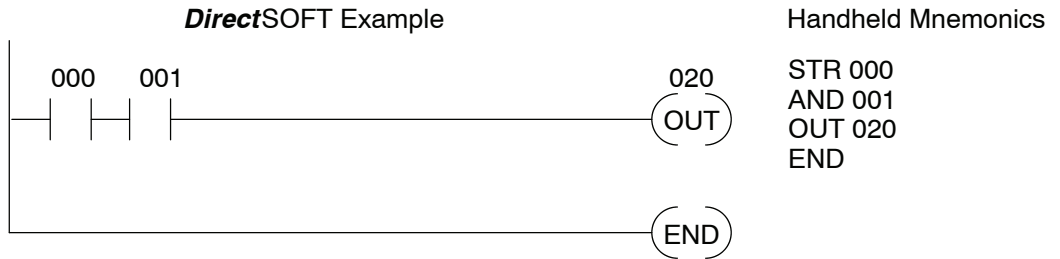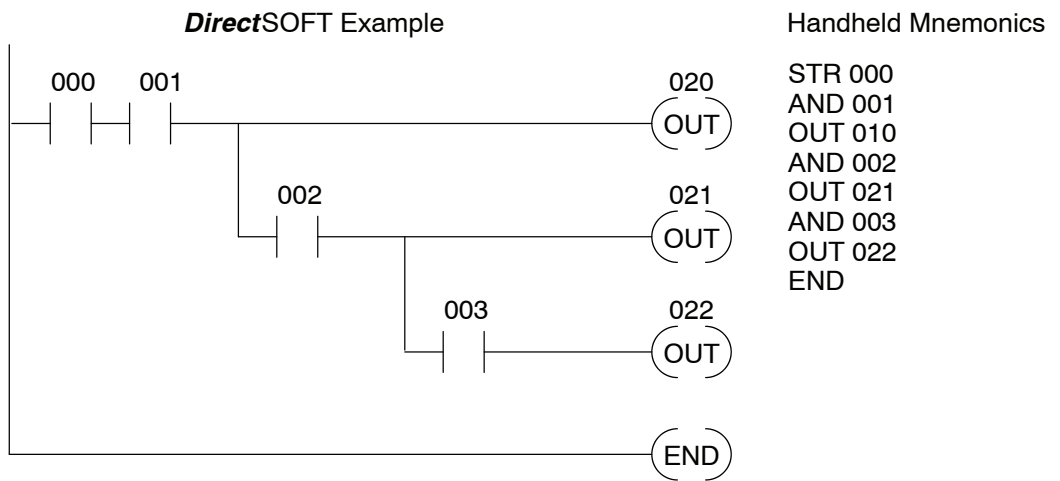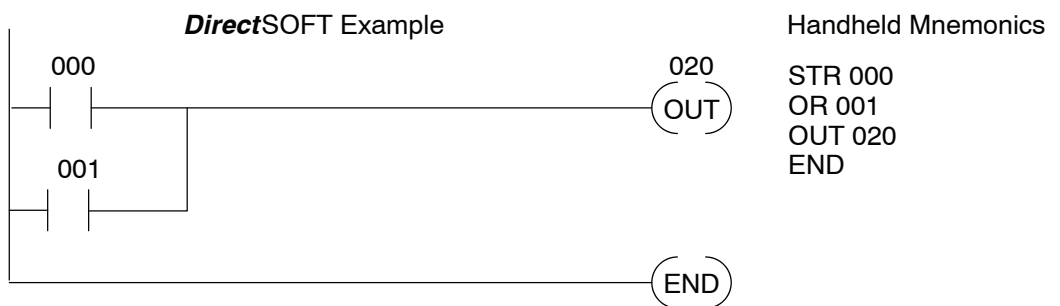
Midline Outputs   Sometimes it is necessary to use midline outputs to get additional outputs that are conditional on other contacts. The following example shows how you can use the AND instruction to continue a rung with more conditional outputs.

*Direct*SOFT Example   Handheld Mnemonics

```
  000   001                        020      STR 000
  ─┤ ├──┤ ├──────┬──────────────────(OUT)   AND 001
                 │                          OUT 010
             002 │               021        AND 002
             ─┤ ├┴──────┬─────────(OUT)     OUT 021
                        │                   AND 003
                    003 │       022         OUT 022
                    ─┤ ├┴────────(OUT)       END

  ────────────────────────────────(END)
```

Parallel Elements   You may also 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.

*Direct*SOFT Example   Handheld Mnemonics

```
  000                              020      STR 000
  ─┤ ├────────┬────────────────────(OUT)    OR 001
              │                             OUT 020
  001         │                             END
  ─┤ ├────────┘

  ────────────────────────────────(END)
```

Joining Series Branches in Parallel

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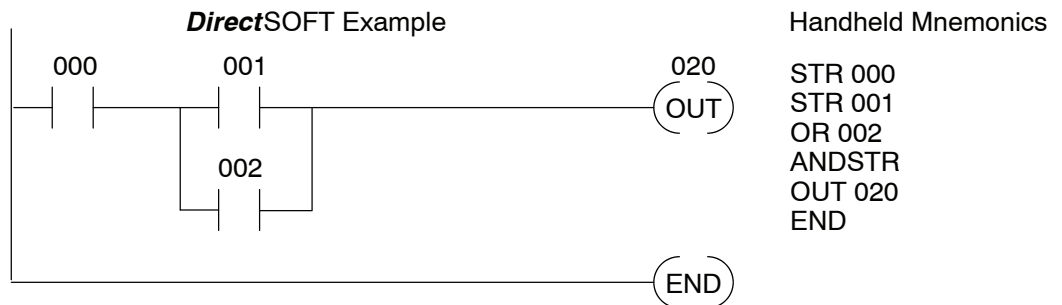
*Direct*SOFT Example	Handheld Mnemonics

```
   000   001                        020      STR 000
    | |   | |                      (OUT)     AND 001
                                             STR 002
   002   003                                 AND 003
    | |   | |                                ORSTR
                                             OUT 020
                                    (END)    END
```
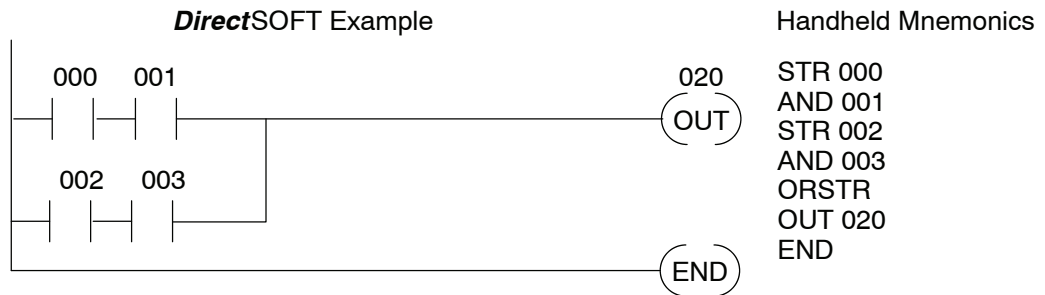
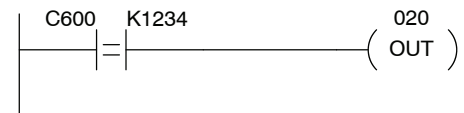**Joining Parallel Branches in Series**

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

*Direct*SOFT Example	Handheld Mnemonics

```
   000        001                   020      STR 000
    | |        | |                 (OUT)     STR 001
                                             OR 002
               002                           ANDSTR
                | |                          OUT 020
                                             END
                                    (END)
```

Comparative Boolean

Many applications require comparisons of data values. This is especially true in applications that use counters. Some PLC manufacturers make it really difficult to do a simple comparison of a counter value and a constant or register. The DL330 and DL340 CPUs provide Comparative Boolean instructions that allow you to quickly and easily solve this problem. Comparative Boolean evaluates two 4-digit values using boolean contacts. The valid evaluations are equal and not equal.

In the following example when the value in counter C600 is equal to the constant value 1234, output 020 will energize.

```
   C600   K1234              020
    |  =  |                 ( OUT )
```

The DL330P also provides Comparative Boolean instructions, but they are greater than and less than instructions instead of equal and not equal.

Combination Networks

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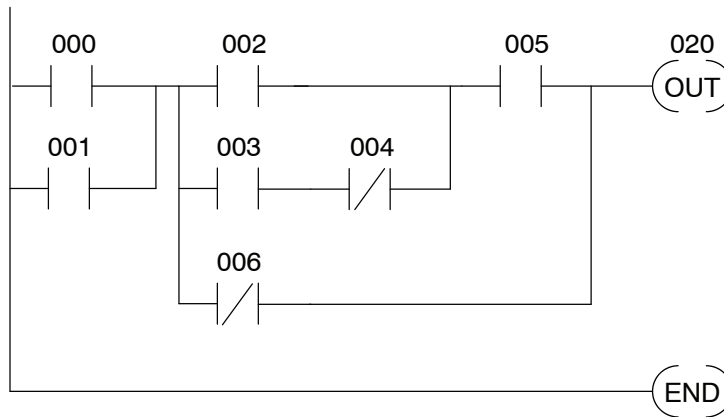

Boolean Stack

There are limits to how many elements you can include in a rung. This is because the DL305 CPUs use an 8-level boolean stack to evaluate the various logic elements. The boolean stack is a temporary storage area that solves the logic for the rung. Each time you enter a STR instruction, the instruction is placed on the top of the boolean stack. Any other instructions on the boolean stack are pushed down a level. The AND, OR, ANDSTR, and ORSTR instructions combine levels of the boolean stack when they are encountered. Since the boolean stack is only eight levels, an error will occur if the CPU encounters a rung that uses more than the eight levels of the boolean stack.

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

The following example shows how the boolean stack is used to solve boolean logic.
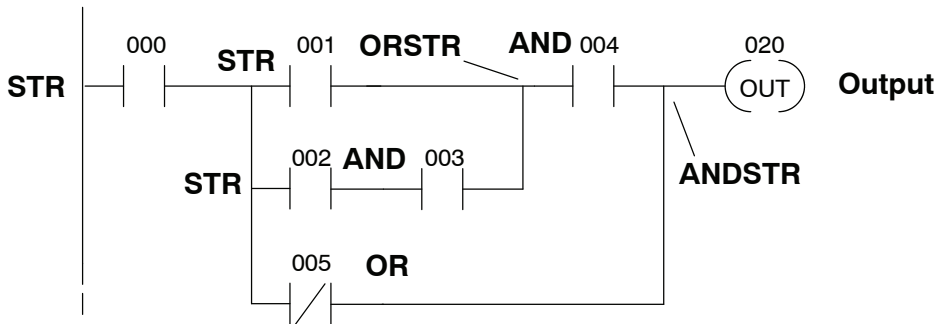
STR 000

1	STR 000
2	
3	
4	
5	
6	
7	
8	

STR 001

1	STR 001
2	STR 000
3	
4	
5	
6	
7	
8	

STR 002

1	STR 002
2	STR 001
3	STR 000
4	
5	
6	
7	
8	

AND 003

1	002 AND 003
2	STR 001
3	STR 000
4	
5	
6	
7	
8	

ORSTR

1	001 OR (002 AND 003)
2	STR 000
3	
8	

AND 004

1	004 AND [001 OR (002 AND 003)]
2	STR 000
3	
8	

OR 005

1	NOT 005 OR 004 AND [001 OR (002 AND 003)]
2	STR 000
3	
8	

ANDSTR

1	000 AND (NOT 005 OR 004) AND [001 OR (002 AND 003)]
2	
3	
8	

Using Timers

Timers are used to time an event for a desired length of time. The single input timer will time as long as the input is on. When the input changes from on to off the timer current value is reset to 0. Timers normally time in tenth of a second intervals, but you can turn on Special Relay 770 to change the timers to hundredth of a second intervals. There is discrete bit associated with each timer to indicate the current value is equal to or greater than the preset value. The timing diagram below shows the relationship between the timer input, associated discrete bit, current value, and timer preset.
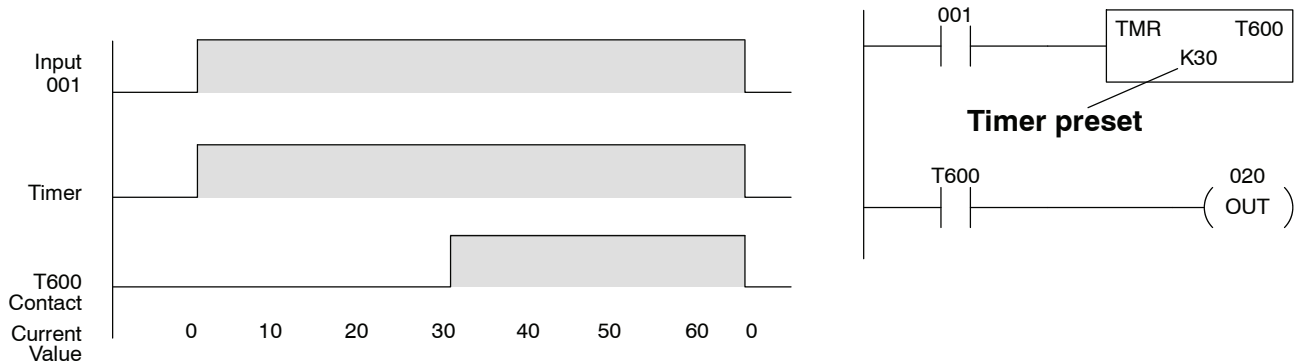
Using Counters

Counters are used to count events. There are two types of counters.

- Regular Up counters
- Stage counters (used with the RLL*PLUS* instructions)

The up counter has two inputs, a count input and a reset input. The maximum count value is 9999. The timing diagram below shows the relationship between the counter input, counter reset, associated discrete bit, current value, and counter preset.
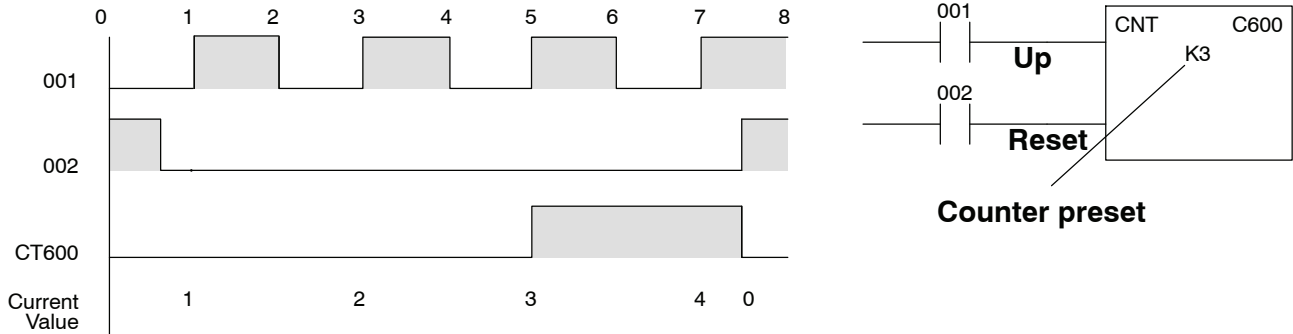
The stage counter has a count input and is reset by the RST instruction. This instruction is used with the RLL*PLUS* instructions. The maximum count value is 9999. The timing diagram below shows the relationship between the counter input, associated discrete bit, current value, counter preset and reset instruction.
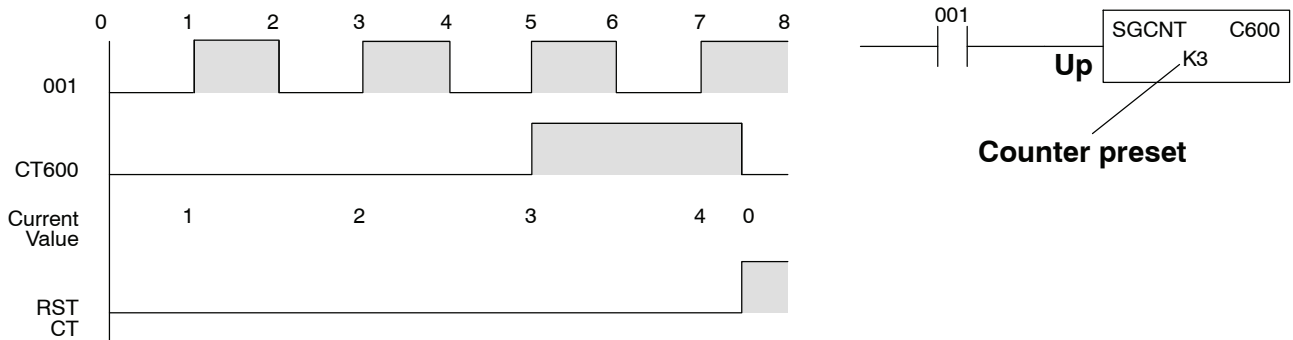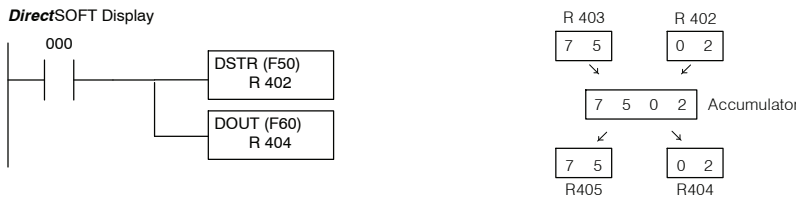
Using the Accumulator

The accumulator in the DL305 series CPUs is a 16 bit register which is used as a temporary storage location for data being copied or manipulated in some manor. For example, you have to use the accumulator to perform math operations such as add, subtract, multiply, etc. Since there are 16 bits, you can use up to a 4-digit BCD number. The accumulator is reset to 0 at the end of every CPU scan.

Copying Data to and from the Accumulator

The Data Store (DSTR) and Data Out (DOUT) instructions and their variations are used to copy data from a register location to the accumulator, or to copy data from the accumulator to a register location.

In the following example, when input 000 is on the value (7502) in R402 and R403 is loaded into the accumulator using the Data Store (F50) instruction. The value in the accumulator is output to data registers R404 and R405 using the Data Out (F60) instruction.

You probably noticed it took two registers to hold a 4-digit BCD number. This is because each BCD digit requires four binary bit positions.

Since the accumulator is 16 bits and register locations are 8 bits, there are variations of the DSTR and DOUT instructions that allow you to copy a single register, or even half of a register (4 bits) either to or from the accumulator. The following example shows how you could use the DSTR3 and DOUT2 instructions to copy the lower 4 bits from register 5 to the upper 4 bits of register 16. (These registers correspond to I/O points and Control Relays respectively.)
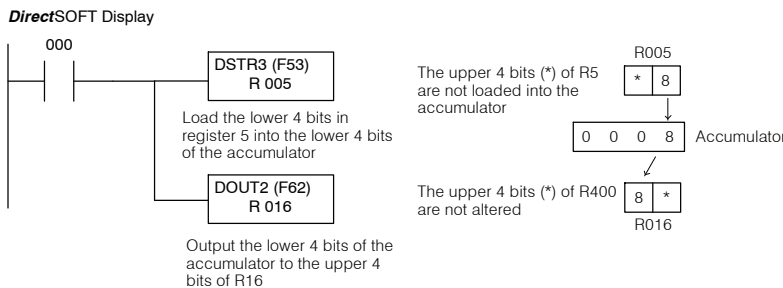
Changing the Accumulator Data

Instructions that change or manipulate data in some way also use the accumulator. The result of the change resides in the accumulator. The original data that was being changed is cleared from the accumulator. In the following example, when input 000 is on the value in R000 and R010 is loaded into the accumulator using the Data Store 5 (F55) instruction. The bit pattern in the accumulator is shifted to the left 4 bit positions using the Shift Left (F80) instruction. Notice how the result resides in the accumulator. The value in the accumulator is copied to data registers R404 and R405 using the Data Out (F60) instruction.

*Direct*SOFT Display
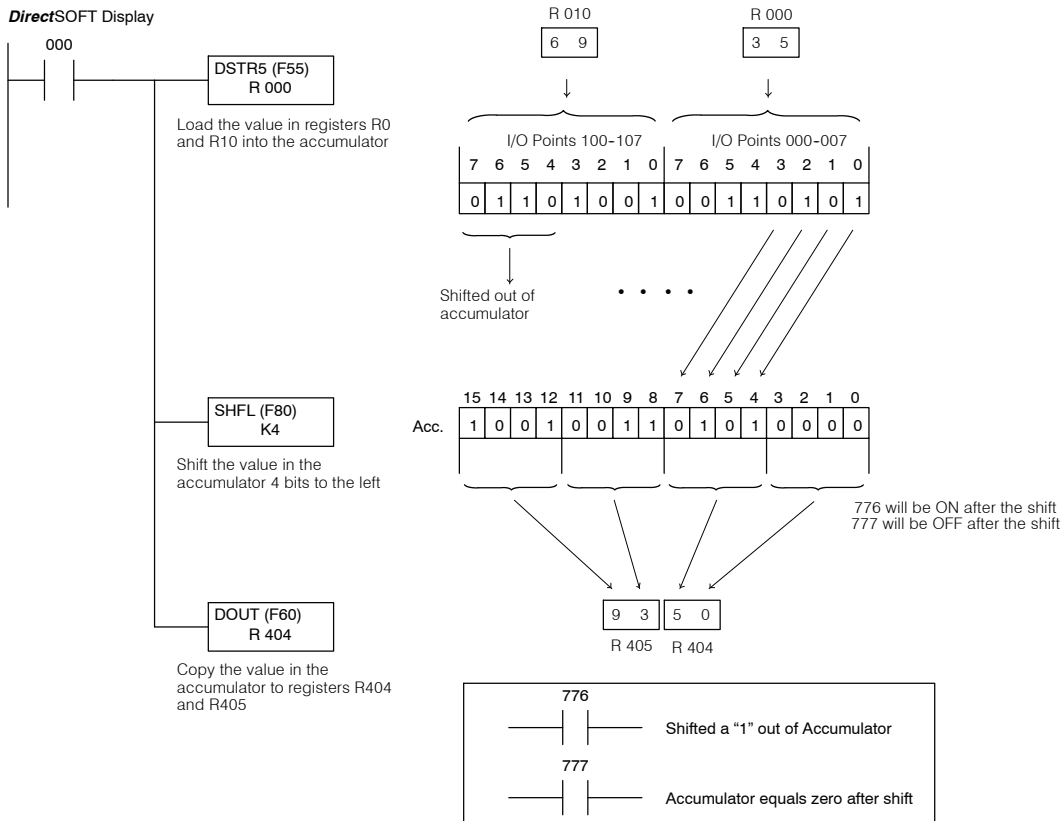
000

DSTR5 (F55)
R 000

Load the value in registers R0 and R10 into the accumulator

R 010
6   9

R 000
3   5

I/O Points 100-107

7	6	5	4	3	2	1	0
0	1	1	0	1	0	0	1

I/O Points 000-007

7	6	5	4	3	2	1	0
0	0	1	1	0	1	0	1

Shifted out of accumulator

• • • • •

SHFL (F80)
K4

Shift the value in the accumulator 4 bits to the left

Acc.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	0	0	1	1	0	1	0	1	0	0	0	0

776 will be ON after the shift
777 will be OFF after the shift

DOUT (F60)
R 404

Copy the value in the accumulator to registers R404 and R405

9   3   5   0

R 405   R 404

776
Shifted a "1" out of Accumulator

777
Accumulator equals zero after shift

Accumulator Operations

The following table lists several instructions that utilize the accumulator. Not all instructions allow you to use all the different memory types. Chapters 11 & 12 provide details on these instructions.

Category	Mnemonic	Description	Memory Areas					
			I/O	CRs	Data Register	Current Values	4-digit BCD Const.	Shift Register Coils
Data Load	DSTR (F50)	Load a 4-digit constant or a 2-bytes of register data into the accumulator	❍	❍	❍	❍	❍	❍
	DSTR 1 (F51)	Load 1-byte of register data into the accumulator	❍	❍	❍	✕	✕	❍
	DSTR 2 (F52)	Load the upper 4 bits of a register into the lower 4 bits of the accumulator	❍	❍	❍	✕	✕	❍
	DSTR 3 (F53)	Load the lower 4 bits of a register into the upper 4 bits of the accumulator	❍	❍	❍	✕	✕	❍
	DSTR 5 (F55)	Load the digital values of 16 I/O points (2 bytes) into the accumulator	❍	✕	✕	✕	✕	✕
Data Out	DOUT (F60)	Write the accumulator to 2 sequential registers	❍	❍	❍	❍	❍	❍
	DOUT 1 (F61)	Write the lower byte of the accumulator to a register	❍	❍	❍	✕	✕	❍
	DOUT 2 (F62)	Write the lower 4 bits of the accumulator to the upper 4 bits of a register	❍	❍	❍	✕	✕	❍
	DOUT 3 (F63)	Write the lower 4 bits of the accumulator to the lower 4 bits of a register	❍	❍	❍	✕	✕	❍
	DOUT 5 (F65)	Write the contents of the accumulator to a 16-point output module (2 bytes)	❍	❍	❍	✕	✕	❍
Math	CMP (F70)	Compare a 2-byte BCD reference or a 4-digit BCD constant to the accumulator	❍	❍	❍	❍	❍	❍
	ADD (F71)	Add a 2-byte BCD reference or a 4-digit BCD constant to the accumulator	❍	❍	❍	❍	❍	❍
	SUBTRACT (F72)	Subtract a 2-byte BCD reference or a 4-digit BCD constant from the accumulator	❍	❍	❍	❍	❍	❍
	MULTIPLY (F73)	Multiply a 2-byte BCD reference or a 4-digit BCD constant by the value in the accumulator	❍	❍	❍	❍	❍	❍
	DIVIDE (F74)	Divide the accumulator by a 2-byte BCD reference or a 4-digit BCD constant	❍	❍	❍	❍	❍	❍

❍ — Memory Type available for use with the instruction

X — Not available

Programming Basics

Category	Mnemonic	Description	Memory Areas					
			I/O	CRs	Data Register	Current Values	4-digit BCD Const.	Shift Register Coils
Bit Manipulation	DAND (F75)	Performs a bit "AND" on a 2-byte reference or a 4-digit BCD constant and the bits in the accumulator	❍	❍	❍	❍	❍	❍
	DOR (F76)	Performs a bit "OR" on a 2-byte reference or a 4-digit BCD constant and the bits in the accumulator	❍	❍	❍	❍	❍	❍
	SHIFT RIGHT (F80)	Shifts the contents of the accumulator to the right a specified number of times. 1 – 15 bits can be shifted.	✕	✕	✕	✕	✕	✕
	SHIFT LEFT (F81)	Shifts the contents of the accumulator to the left a specified number of times. 1 – 15 bits can be shifted.	✕	✕	✕	✕	✕	✕
Data Conversion	DECODE (F82)	Decodes the first 4 bits of the accumulator into a decimal number.	✕	✕	✕	✕	✕	✕
	ENCODE (F83)	Encodes an accumulator bit into a 4-bit code that represents the decimal number (0–15).	✕	✕	✕	✕	✕	✕
	INV (F84)	Logically inverts the contents of the accumulator (1 to 0, 0 to 1).	✕	✕	✕	✕	✕	✕
	BCD–BIN (F85)	Converts the accumulator value from BCD to Binary	✕	✕	✕	✕	✕	✕
	BIN–BCD (F86)	Converts the accumulator value from Binary to BCD	✕	✕	✕	✕	✕	✕
Fault Detection	FAULT (F20)	Sends a 4-digit BCD number, from a 2-byte reference or a constant, to the programmer display	✕	✕	✕	✕	✕	✕

❍ — Memory Type available for use with the instruction

X — Not available