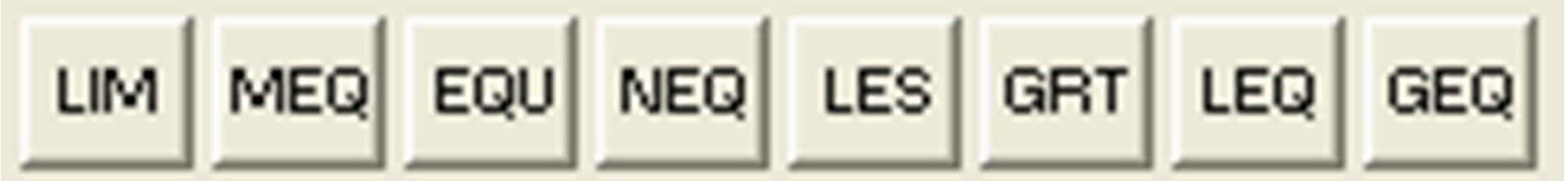
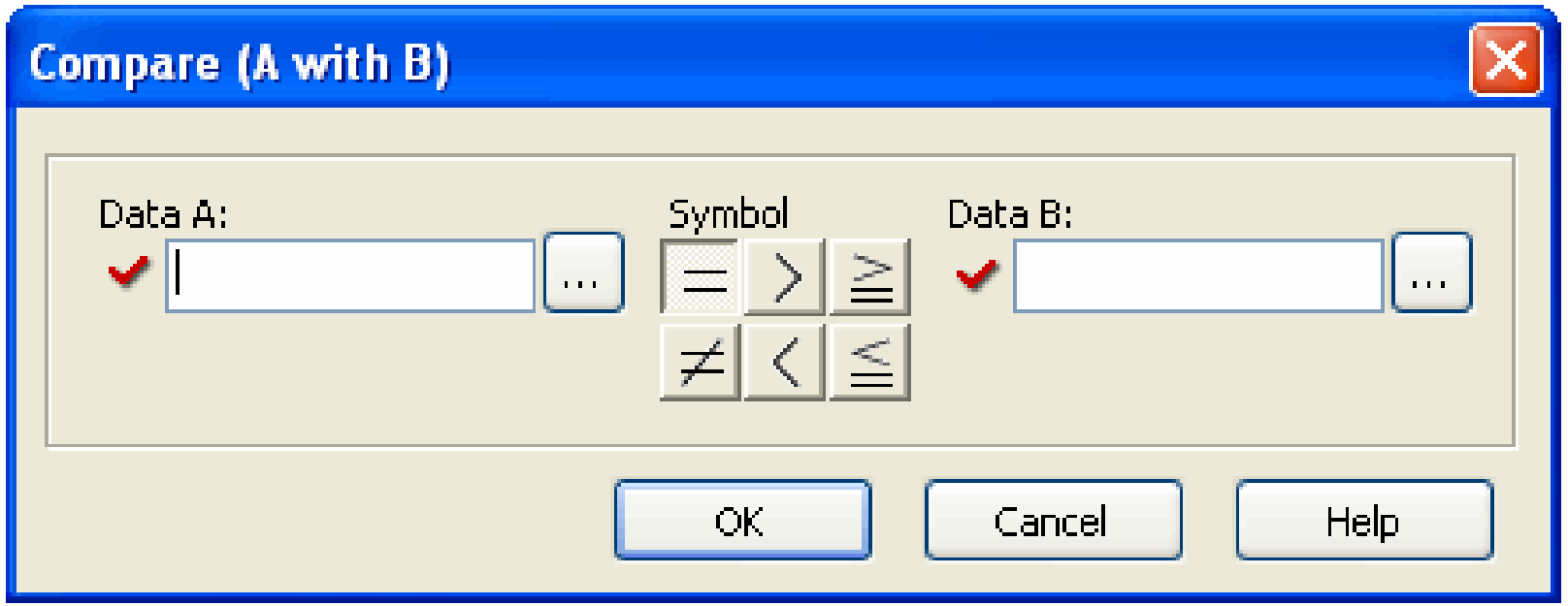
Arithmetic, Logical, Data Handling and Control Instructions

Chapter 6

Table of Content

Comparison Instructions Review





Arithmetic and Logical Instructions Overview

- The majority of the instructions take two input values, perform the specified arithmetic function, and output the result to an assigned memory location.
  - Source is the address(es) of the value(s) on which the mathematical, logical, or move operation is to be performed.
  - This can be word addresses or program constants. An instruction that has two source operands does not accept program constants in both operands.
  - Destination is the address of the result of the operation.
  - Signed integers are stored in two's complement form and apply to both source and destination parameters.

- The arithmetic status bits are found in Word 0, bits 0 to 3 in the controller status file. After an instruction is executed, the arithmetic status bits in the status file are updated:

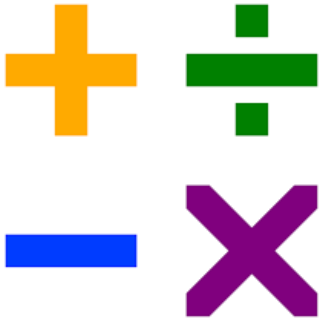
- When using either an SLC 5/03 (OS301 and higher), SLC 5/04, or SLC 5/05 processor; floating point and string values (specified at the word level) are supported.

NOT(!)

AND(&&)

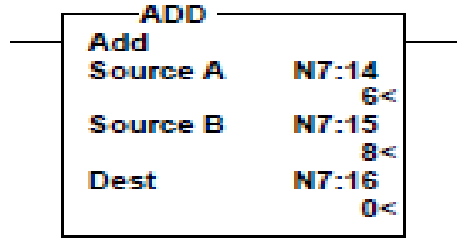
OR(||)

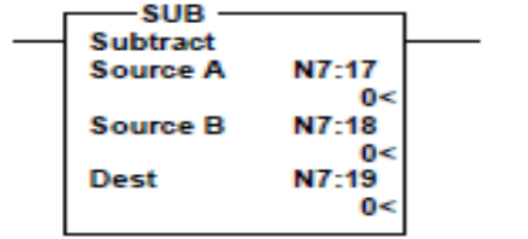
With this Bit:		The Controller:
S:0/0	Carry (C)	sets if carry is generated; otherwise cleared.
S:0/1	Overflow (V)	indicates that the actual result of a math instruction does not fit in the designated destination.
S:0/2	Zero (Z)	indicates a 0 value after a math, move, or logic instruction.
S:0/3	Sign (S)	indicates a negative (less than 0) value after a math, move, or logic instruction.

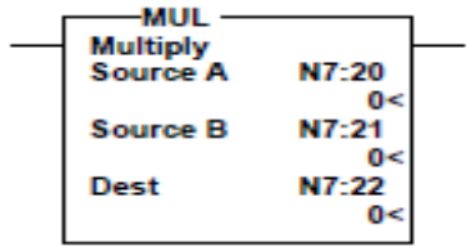


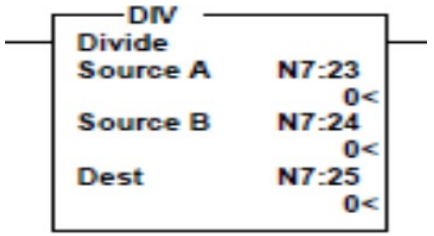
Arithmetic Instructions

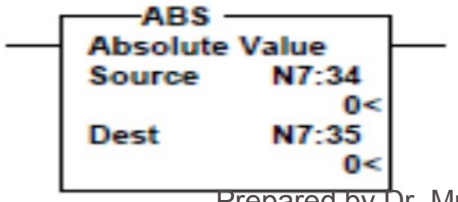
- Use the ADD instruction to add one value (source A) to another value (source B) and place the result in the destination.
- Use the SUB instruction to subtract one value (source B) from another (source A) and place the result in the destination.
- Use the MUL instruction to multiply one value (source A) by another (source B) and place the result in the destination.
- Use the DIV instruction to divide one value (source A) by another (source B). The rounded result is then placed in the destination
- When this instruction is evaluated as true, the square root of the absolute value of the source is calculated and the rounded result is placed in the destination.
- Use the ABS instruction to calculate the absolute value of the Source and place the result in the Destination.
- Use the XPY instruction to raise a value (source A) to a power (source B) and store the result in the destination.
- Use the CLR instruction to set the destination value of a word to zero.

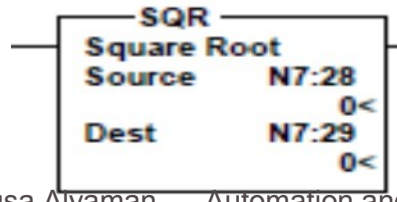


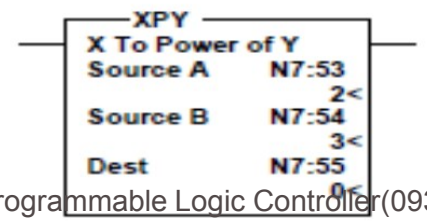


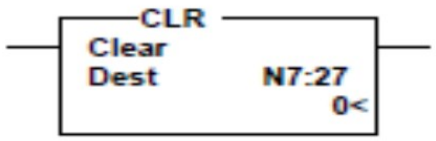












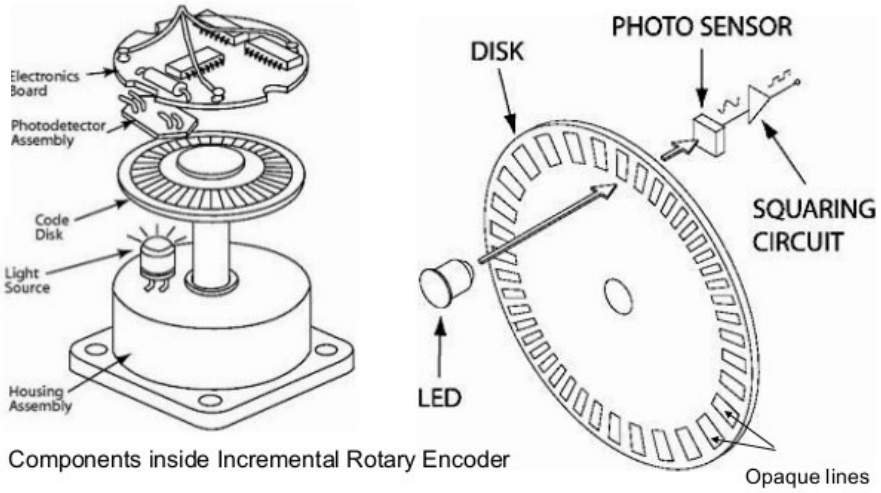
Example 1

Find the length of the long leg of a right triangle and store it in N7:0 register, knowing the two other leg lengths are placed in the following two registers N7:1, N7:2

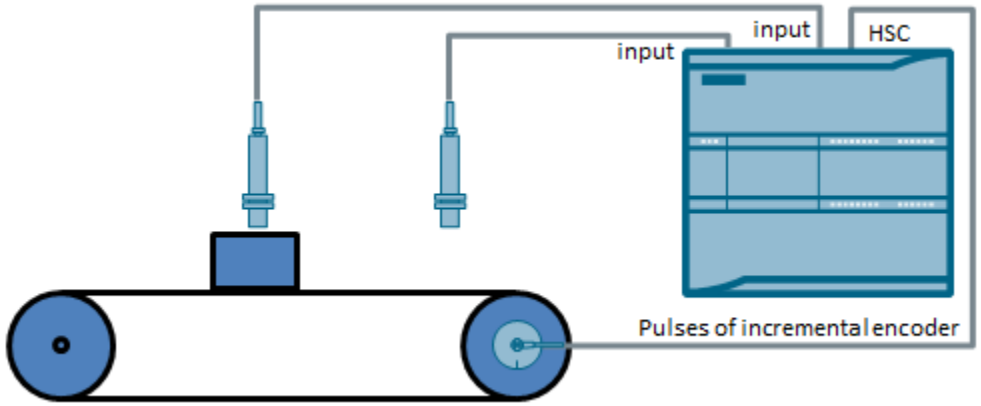
Example 2

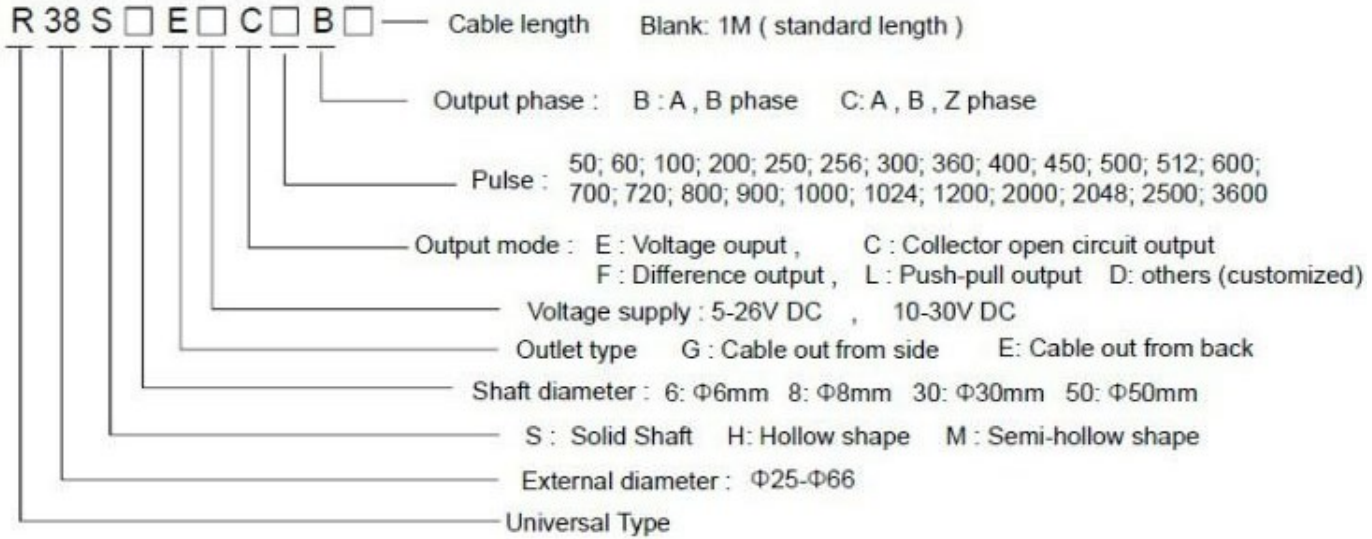
An incremental encoder with 720 pulse per revaluation is required on a conveyor to provide a digital readout display with the distance between two proximity switches in cm.

Note: The conveyor diameter is 14 cm, if the HSC counts is stored in N7:3, it starts by the first proximity switch and stops by the second one, then reset.



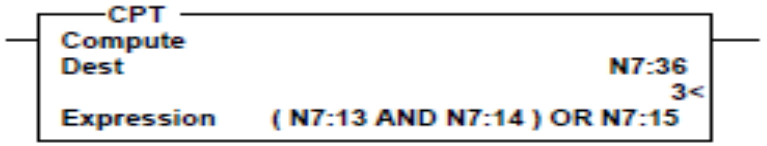
Components inside Incremental Rotary Encoder





Compute (CPT)

- The CPT instruction performs copy, arithmetic, logical, and conversion operations. You define the operation in the Expression and the result is written in the Destination.
- The CPT uses functions to operate on one or more values in the Expression to perform operations such as:
  - converting from one number format to another
  - manipulating numbers
  - performing trigonometric functions



- Instructions that can be used in the Expression include:

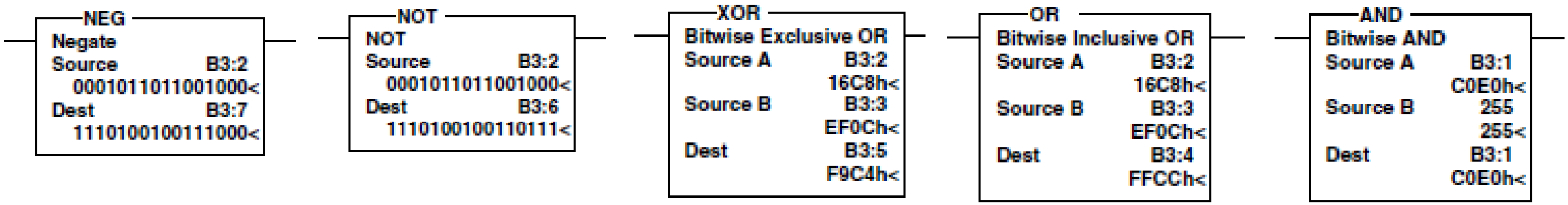
+, -, *, | (DIV), SQR, - (NEG), NOT, XOR, OR, AND, TOD, FRD, LN, TAN, ABS, DEG, RAD, SIN, COS, ATN, ASN, ACS, LOG, and ** (XPY).

- The execution time of a CPT instruction is longer than a single arithmetic operation and uses more instruction words
- Enter the following parameters when programming this instruction:
  - Destination can be a word address or the address of a floating-point data element.
  - Expression is zero or more lines, with up to 28 characters per line, up to 255 characters

Example 1

Logical Instructions

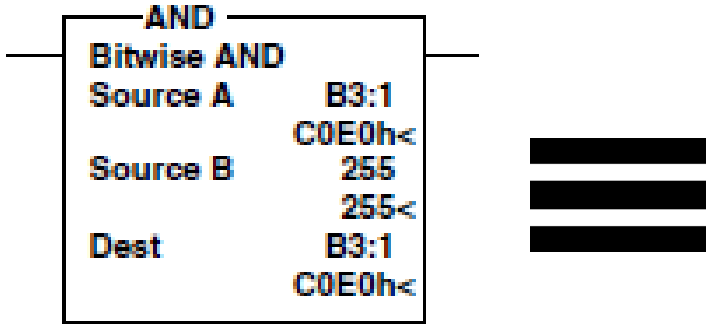
- This instruction performs a bit-by-bit logical AND. The operation is performed using the value at source A and the value at source B. The result is stored in the destination
- This instruction performs a bit-by-bit logical OR. The operation is performed using the value at source A and the value at source B. The result is stored in the destination.
- This instruction performs a bit-by-bit logical XOR. The operation is performed using the value at source A and the value at source B. The result is stored in the destination.
- This instruction performs a bit-by-bit logical NOT. The operation is performed using the value at source A. The result (one's complement of A) is stored in the destination.
- Use the NEG instruction to change the sign of the source and then place it in the destination. The destination contains the two's complement of the source. For example, if the source is 5, the destination would be -5.



Example 3

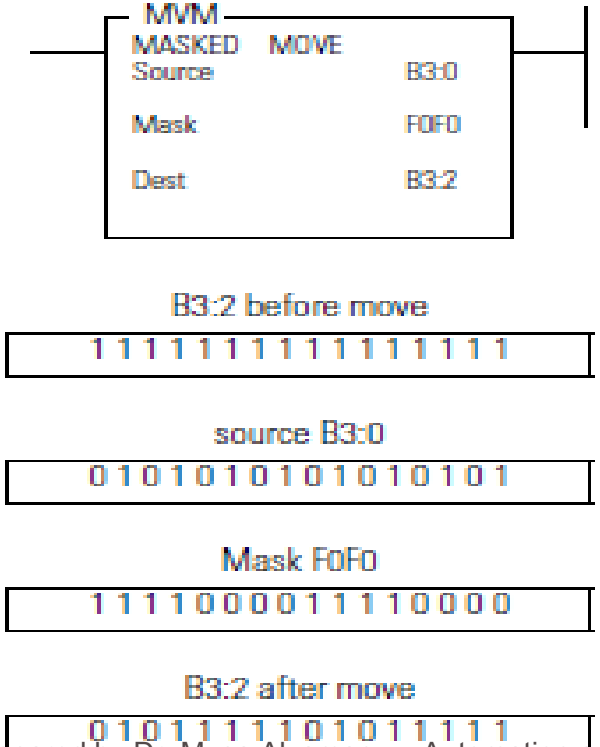
Write XOR instruction using AND, OR and NOT instructions

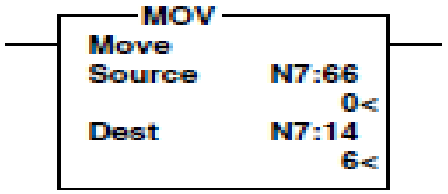
Write AND instruction using OR and NOT instructions

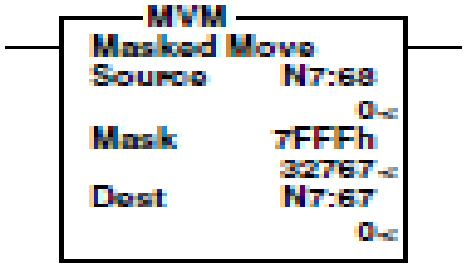


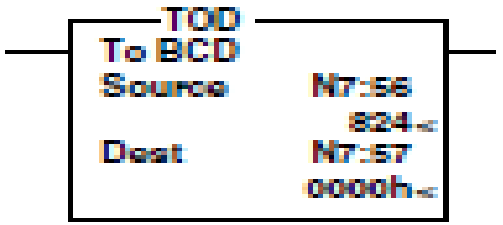
Data Handling instructions

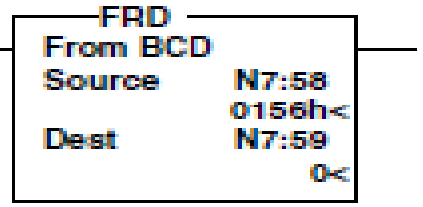
- Convert to BCD (TOD) Use this instruction to convert 16-bit integers into BCD values.
- Convert from BCD (FRD) Use this instruction to convert BCD values to integer values
- Move (MOV) This output instruction moves the source value to the destination location. As long as the rung remains true, the instruction moves the data each scan.
- Masked Move (MVM) The MVM instruction is a word instruction that moves data from a source location to a destination, and allows portions of the destination data to be masked by a separate word. As long as the rung remains true, the instruction moves the data each scan.





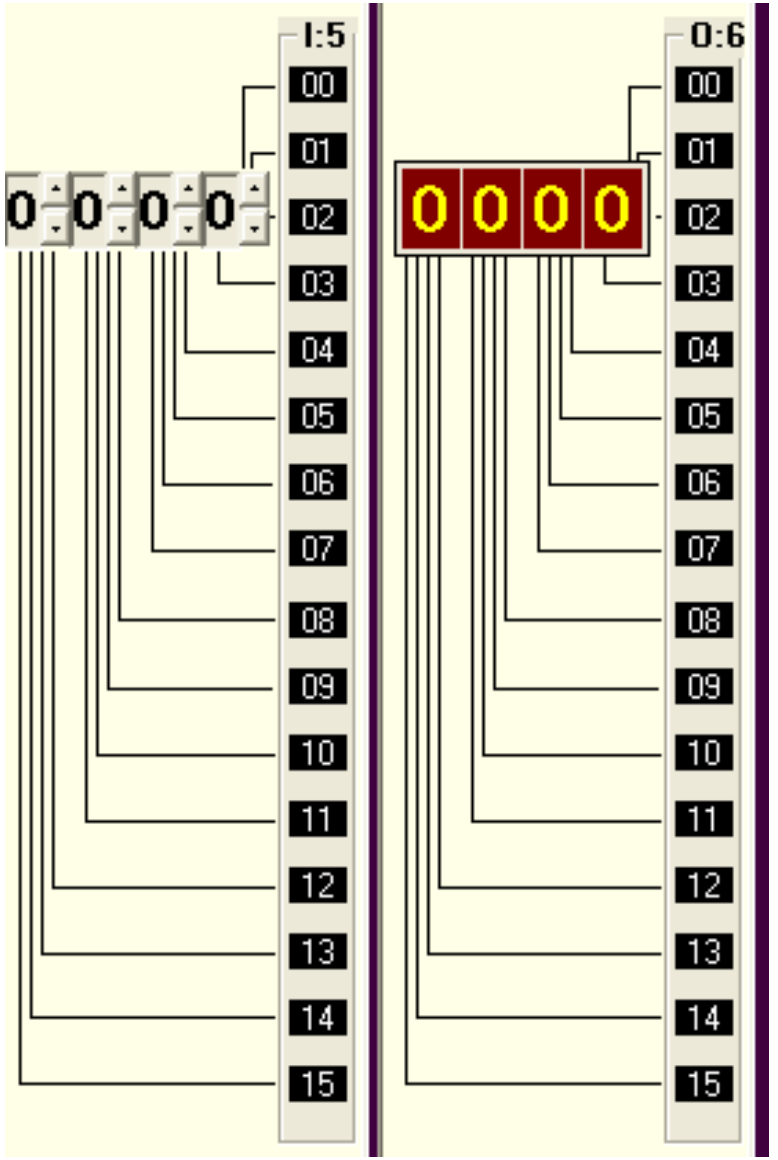






Example 4

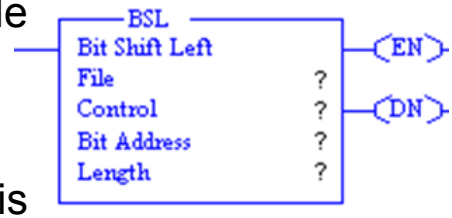
Read the number A (consists from first two digits in I:5), number B (consists from last two digits in I:5), and display the result of multiplication of number A and number B in O:6.



Shift Instructions (*Bit Shift Left (BSL), Bit Shift Right (BSR)*)

The following parameters are associated with shift instructions:

- **File** is the address of the bit array you want to manipulate. You must use the file indicator (#) in the bit array address.
- **Control** is the control element that stores the status byte of the instruction and the size of the array (in number of bits). Note that the control address should not be used for any other instruction
- **Bit Address** is the address of the source bit that the instruction inserts in the first (lowest) bit position (BSL) or the last (highest) bit position (BSR).
- **Length** (size of bit array) is the number of bits in the bit array, up to 2048 bits. A length value of 0 causes the input bit to be transferred to the UL bit. A length value that points past the end of the programmed file causes a runtime major error to occur.



Status bits of the control element may be addressed by mnemonic. They include:

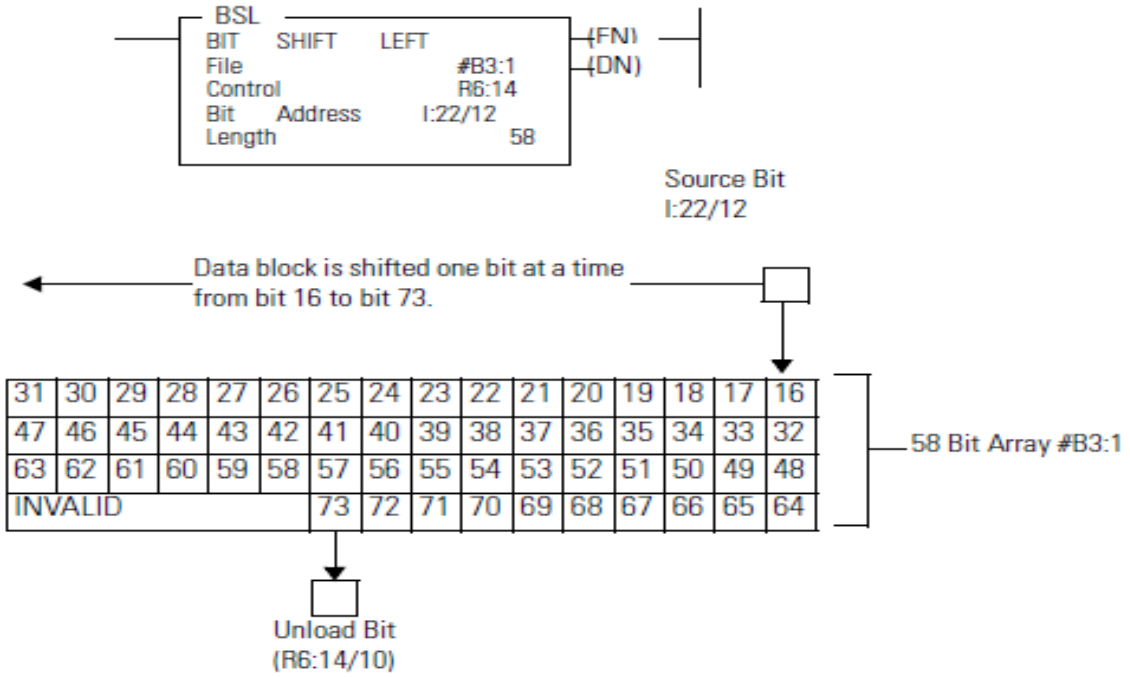
- **Unload Bit UL (bit 10)** stores the status of the bit exited from the array each time the instruction is enabled.
- **Error Bit ER (bit 11),** when set, indicates the instruction detected an error such as entering a negative number for the length or position. Avoid using the output bit when this bit is set.
- **Done Bit DN (bit 13),** when set, indicates the bit array has shifted one position.
- **Enable Bit EN (bit 15)** is set on a false-to-true transition of the rung and indicates the instruction is enabled. When the register shifts and input conditions go false, the enable, done, and error bits are reset.

Table 7.2 Control File Structure

	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Word 0	EN		DN		ER	UL			Not Used							
Word 1	Size of bit array (number of bits)															
Word 2	Reserved															

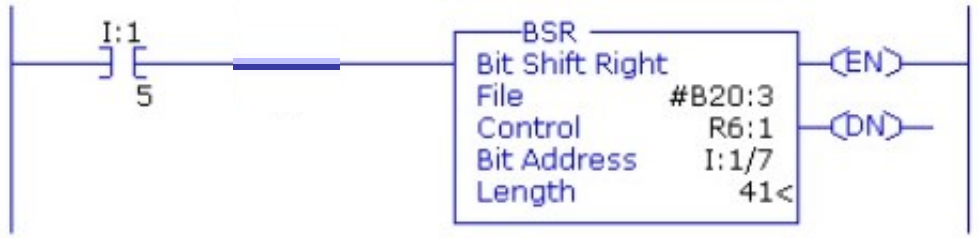
Bit Shift Left (BSL)

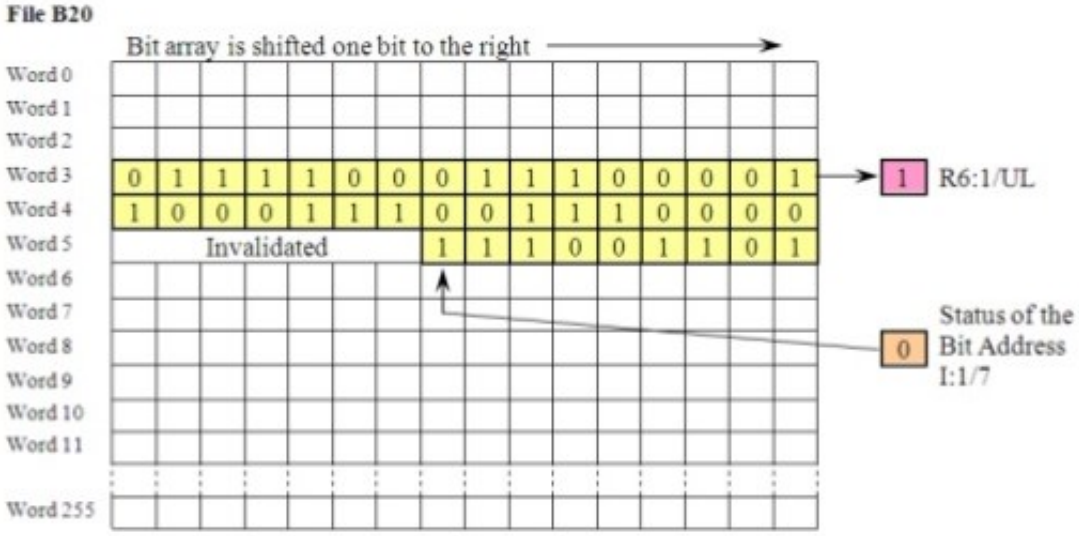
- BSL is output instructions that load data into a bit array one bit at a time. The data is shifted through the array, then unloaded one bit at a time.
- When the rung goes from false-to-true, the processor sets the enable bit (EN bit 15) and the data block is shifted to the left (to a higher bit number) one bit position.
- The specified bit at the bit address is shifted into the first bit position.
- The last bit is shifted out of the array and stored in the unload bit (UL bit 10).
- The shift is completed immediately.
- For wraparound operation, set the position of the bit address to the last bit of the array or to the UL bit, whichever applies.
- If you wish to shift more than one bit per scan, you must create a loop in your application using the JMP, LBL, and CTU instructions.
- The instruction invalidates all bits beyond the last bit in the array (as defined by the length) up to the next word boundary



Bit Shift Right (BSR)

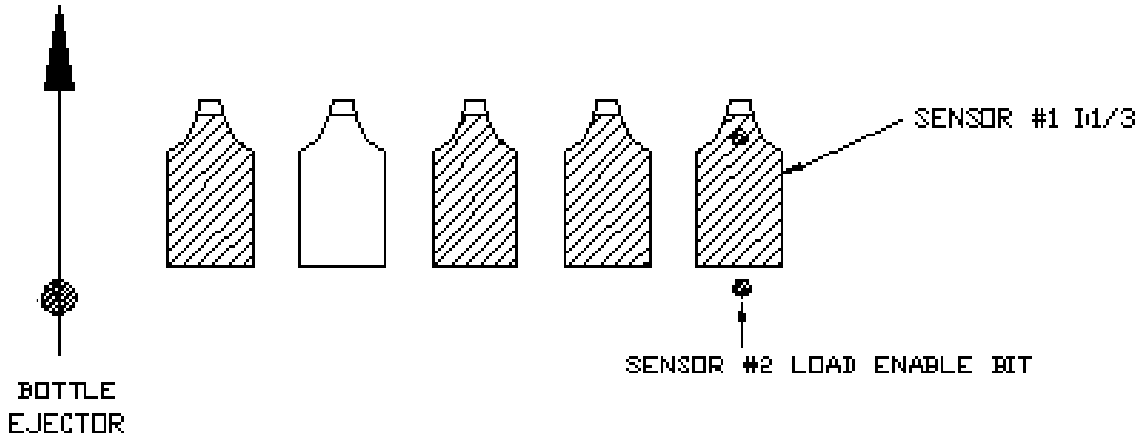
- When the rung goes from false-to-true, the enable bit (EN bit 15) is set and the data block is shifted to the right (to a lower bit number) one bit position.
- The specified bit at the bit address is shifted into the last bit position.
- The first bit is shifted out of the array and stored in the unload bit (UL bit 10) in the status byte of the control element.
- The shift is completed immediately.
- For wraparound operation, set the position of the bit address to the first bit of the array or to the UL bit, whichever applies.
- If you wish to shift more than one bit per scan, you must create a loop in your application using the JMP, LBL, and CTU instructions.
- The instruction invalidates all bits beyond the last bit in the array (as defined by the length) up to the next word boundary





Example 5

After filling process, bottles are moved on the conveyor belt for packing process. . Sensor #1 detects if any empty bottle is left on the conveyor or not. Sensor #2 used to detect the arrival of a bottle regardless it is empty or not. A bottle ejector is connected after space for five bottles. Implement automation of this in PLC using shift instruction in Ladder Diagram.



Program Flow instructions overview

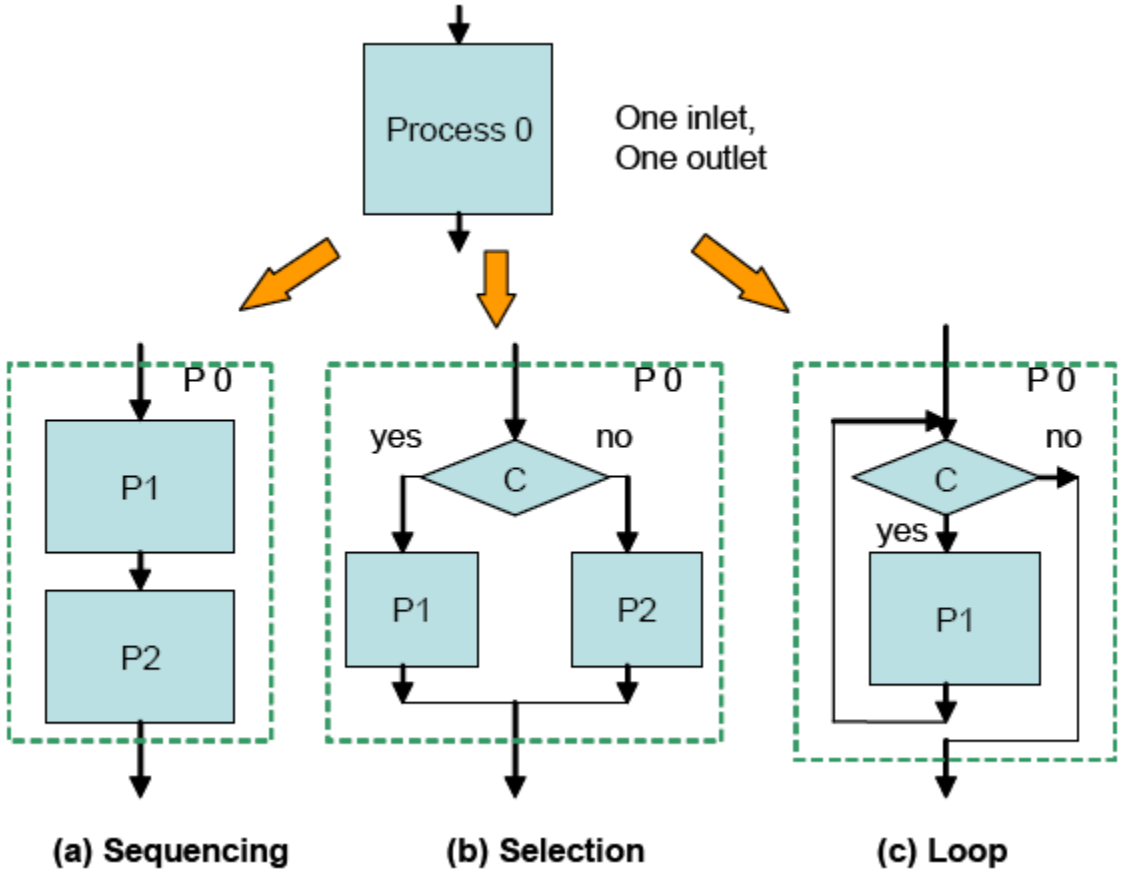
Use these instructions to control the sequence in which your program is executed.

Control instructions allow you to change the order in which the processor scans a ladder program.

Typically, these instructions are used to minimize scan time, create a more efficient program, and troubleshoot a ladder program.

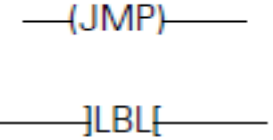
The following is a list of some program flow counter instructions in SLC 500:

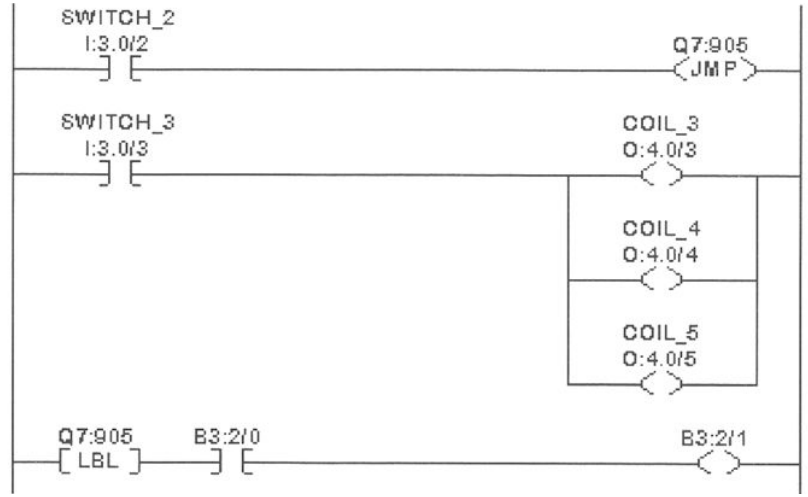
- **JMP and LBL**
- **JSR, SBR, and RET**



Jump to Label (JMP) and Label (LBL)

- Use these instructions in pairs to skip portions of the ladder program.
- Jumping forward to a label saves program scan time by omitting a program segment until needed.
- Jumping backward lets the controller execute program segments repeatedly.
- Use a counter, timer, or the program scan register (system status register, word S:3, bits 0 to 7) to limit the amount of time you spend looping inside of JMP/LBL instructions.
- Enter a decimal label number from 0 to 255 in each subroutine file.
- The JMP instruction causes the controller to skip rungs. You can jump to the same label from one or more JMP instructions
- This input instruction is the target of JMP instructions having the same label number.
- You must program this instruction as the first instruction of a rung. This instruction has no control bits.
- You can program multiple jumps to the same label by assigning the same label number to multiple JMP instructions. However, label numbers must be unique.



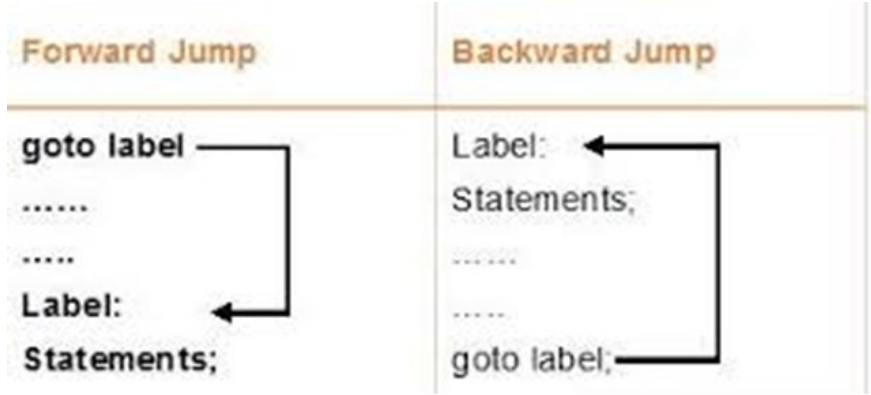


If the Rung Containing the Jump Instruction is	Then the Program
True	Skips from the rung containing the JMP instruction to the rung containing the designated LBL instruction and continues executing. You can jump forward or backward.
False	Does not execute the JMP instruction.

Example 6

- Implement the following C++ code using Ladder Logic assume X is stored in N7:1 and Y in N7:2:

```
if (X>5)
Y= 4X+1;
else
Y= X³-2X+1
```

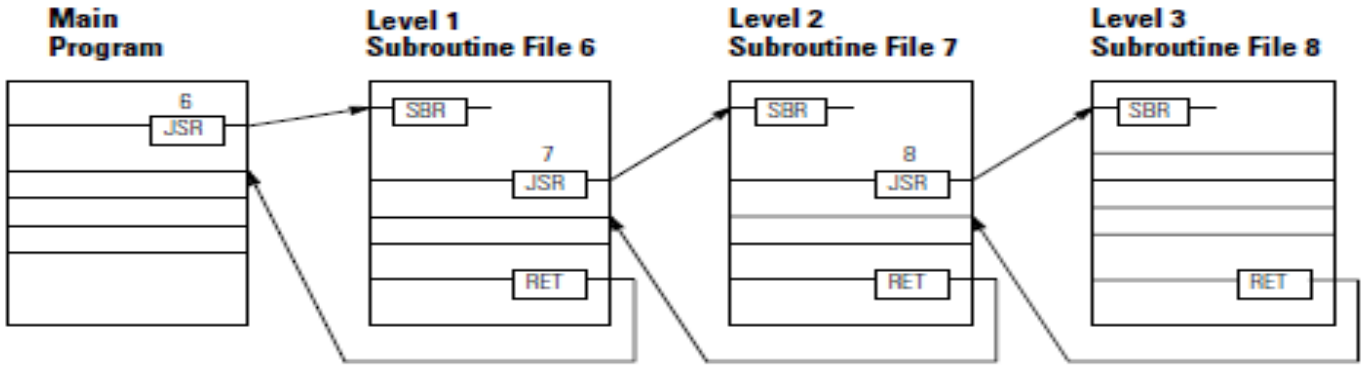


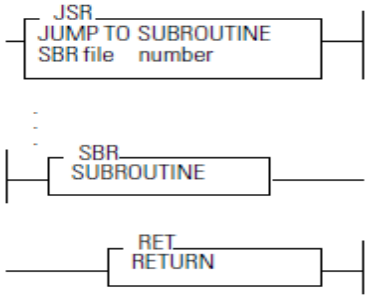
- Implement the following C++ code using Ladder Logic assume A is stored in N7:1 and i in N7:2:

```
for(i=0, i<10, i++)
A= A+5
```

Jump to Subroutine (JSR), Subroutine (SBR), and Return (RET)

- The JSR, SBR, and RET instructions are used to direct the controller to execute a separate subroutine file within the ladder program and return to the instruction following the JSR instruction.
- If you use the SBR instruction, the SBR instruction must be the first instruction on the first rung in the program file that contains the subroutine.
- Use a subroutine to store recurring sections of program logic that must be executed from several points within your application program.
- A subroutine saves memory because you program it only once.
- The controller does not update I/O until it reaches the end of the main program (after executing all subroutines).
- Outputs controlled within a subroutine remain in their last state until the subroutine is executed again.
- Nesting subroutines allows you to direct program flow from the main program to a subroutine and then on to another subroutine. The following rules apply when nesting subroutines.
  - With Fixed and SLC 5/01 processors, you can nest subroutines up to four levels
  - With SLC 5/02 and higher processors, you can nest subroutines up to eight levels.
- An error occurs if more than the allowable levels of subroutines are called (subroutine stack overflow) or if more returns are executed than there are call levels (subroutine stack underflow).





Jump to Subroutine (JSR), Subroutine (SBR), and Return (RET)

Using JSR

- When the JSR instruction is executed, the controller jumps to the subroutine instruction (SBR) at the beginning of the target subroutine file and resumes execution at that point.
- You cannot jump into any part of a subroutine except the first instruction in that file.
- You must program each subroutine in its own program file by assigning a unique file number (3 to 255)

Using SBR

- The target subroutine is identified by the file number that you entered in the JSR instruction. This instruction serves as a label or identifier for a program file as a regular subroutine file.
- This instruction has no control bits. It is always evaluated as true. The instruction must be programmed as the first instruction of the first rung of a subroutine. Use of this instruction is optional; however, we recommend using it for clarity.

Using RET

- This output instruction marks the end of subroutine execution or the end of the subroutine file.
- It causes the controller to resume execution at the instruction following the JSR instruction.
- If a sequence of nested subroutines is involved, the instruction causes the processor to return program execution to the previous subroutine.
- Without an RET instruction, the END instruction (always present in the subroutine) automatically returns program execution to the instruction following the JSR instruction in your calling ladder file.