COMPARISON AND FLOW CONTROL INSTRUCTIONS

5

This final module covers two more categories of programming instructions: comparison instructions and flow control instructions. Comparison instructions are used in a MicroLogix 1000 to compare the values stored in two memory locations. Flow control instructions are used to change the order of execution of instructions in a ladder program. This module contains four sections:

1. Comparison instructions
2. Flow control instructions—alter sequence
3. Flow control instructions—halt execution
4. Flow control instructions—alter I/O scan

Key Points

After finishing this module, you will:

- understand the different comparison instructions available in the MicroLogix 1000—including equal, masked comparison for equal, and the limit test instructions
- know the flow control instructions that alter the sequence of program execution—including the jump, jump to subroutine, and master control reset instructions
- understand temporary end and suspend flow control instructions, which halt the execution of the control program
- grasp the two flow control instructions that alter the I/O scan—the immediate input with mask instruction and the immediate output with mask instruction

5-1 Comparison Instructions

Comparison instructions compare the values stored in two memory locations. These two values can be the data stored in two different word locations, or one can be the data stored in a word and the other can be a constant value.

At the end of this section, you will know:

- the equal instruction
- other basic comparison instructions that work like the equal instruction
- the masked comparison for equal instruction
- the limit test instruction

Equal Instruction

The **equal instruction** is a block instruction that looks like the one shown in Figure 5-1. This instruction compares the data values specified by the source A and source B parameters. If these data values are equal, then the equal block's output will energize, providing continuity to the rest of the rung.

The value specified by source A must be a word location in memory (see Figure 5-2). This word location may specify the accumulated value for a timer or counter, the contents of an integer file word, or any other data stored in memory. The value specified by source B may be either a word location or a constant. If source B contains a word location, then it specifies the location of particular data in memory, just as the source A parameter does. If source B is a constant, then this parameter contains a fixed decimal value to which the instruction compares the source A data.

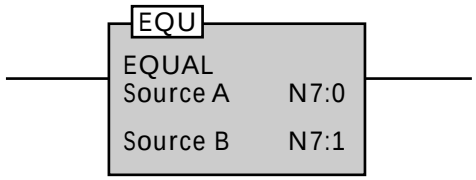

Figure 5-1. An equal instruction.

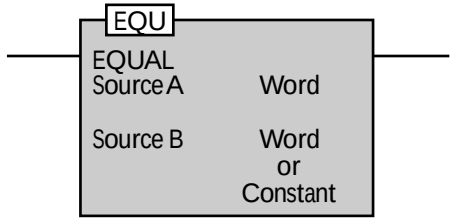

Figure 5-2. In an equal instruction, the value specified by source A must be a word location. The value specified by source B may be either a word location or a constant.

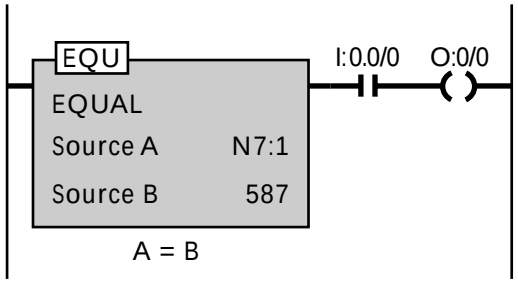

Figure 5-3. An equal instruction programmed in a ladder rung.

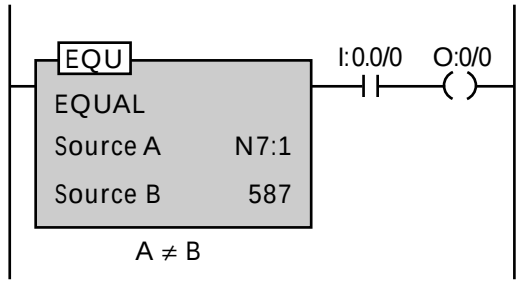

Figure 5-4. If the source A and B values are not equal, then the equal block will not be energized.

An equal instruction is always the first instruction programmed in a rung (see Figure 5-3). This instruction looks at the data stored in the source A word, which may hold a value such as the input value from a set of thumbwheel switches. The equal instruction then compares this source A data to the value indicated by source B. If these two data values are equal, the equal block energizes, providing continuity to the rest of the rung. If the other input conditions in the rung are satisfied, the output will turn on. If the source A and B values are not equal, then the equal block will not be energized (see Figure 5-4). As a result, the output will not be energized, even if the other input conditions are satisfied.

Other Basic Comparison Instructions

The five other comparison instructions used in a MicroLogix 1000 work much like the equal instruction. All of these other comparison instructions are block instructions that specify source A and source B values. As in an equal block, source A must be a word location, while source B can be either a word location or a constant. The other comparison instructions are as follows:

- **not equal instruction**—energizes its output if the source A and B values are not equal to each other
- **less than instruction**—energizes its output if the value in source A is less than the value stored in source B
- **less than or equal instruction**—energizes its output if the source A value is either less than or equal to the source B value
- **greater than instruction**—energizes its output if the value stored in source A is greater than the value stored in source B

- **greater than or equal instruction**—energizes its output if the source A value is greater than or equal to the source B value

Masked Comparison For Equal Instruction

A **masked comparison for equal instruction**, which is shown in Figure 5-5, is abbreviated as MEQ in a MicroLogix's ladder program. This instruction compares part of the word specified by the source location with the value specified by the compare location. It uses a mask value to filter out those parts of the source value that will not be compared. An MEQ instruction's source, compare, and mask parameters work as follows:

- The source parameter specifies the word location of the data to be compared. This is equivalent to the source A parameter in an equal instruction.
- The mask parameter specifies either the word location of the mask value or the mask value itself. If the mask value is entered directly into the MEQ block, then it will be expressed in hexadecimal form. If the mask parameter specifies a word location, then the decimal value in the word will be displayed in the block in hexadecimal form.
- The compare parameter specifies either a word location or a constant value. The compare parameter is much like an equal instruction's source B parameter.

When an MEQ instruction is enabled, it takes the value stored in the source word and then masks out the bits denoted by zeroes in the mask (see Figure 5-6). This leaves only the bits that are specified by ones in the mask. The block then compares this masked value to the compare value. If the two bit patterns defined by the mask are equal, the block's output turns on.

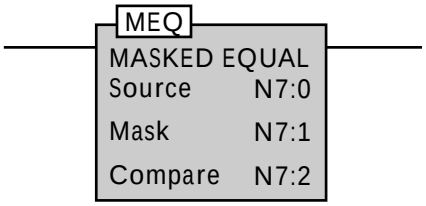

Figure 5-5. A masked comparison for equal instruction.

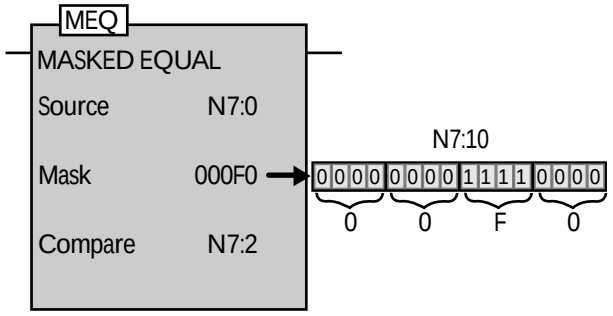

Figure 5-6. When an MEQ instruction is enabled, it masks out the source word bits denoted by zeroes in the mask.

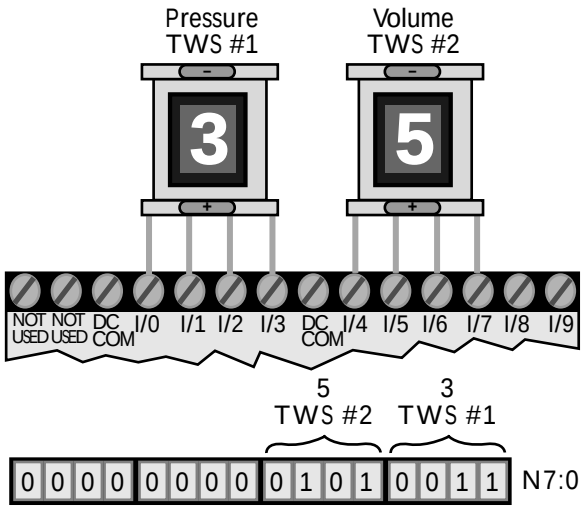

Figure 5-7. An MEQ instruction used to decode data from multiple thumbwheel switches.

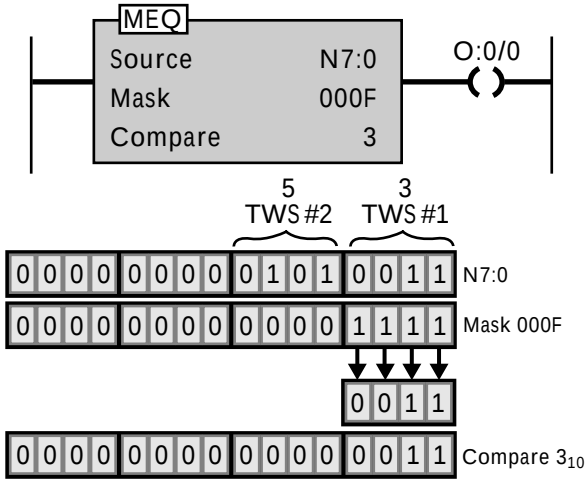

Figure 5-8. A ladder program in which output O:0/0 turns on when the pressure level is 3.

An MEQ instruction is useful for applications such as decoding data from multiple thumbwheel switches. Figure 5-7 shows an example of this type of application in which two thumbwheel switches are connected to terminals 0 through 7 of a MicroLogix 1000. The first thumbwheel switch sends pressure data to the PLC, while the second thumbwheel switch sends volume data. The MicroLogix stores this data in the first eight bits of file N7:0.

Figure 5-8 shows a ladder program in which output O:0/0 turns on when the pressure level is 3. The program works like this:

- The source value for the masked comparison for equal instruction is integer file 0 (N7:0).
- The mask value is 000F, which will mask out all but the first thumbwheel switch's bits.
- The compare value is the decimal value 3, which is equivalent to the binary value 11.
- When the masked comparison block is energized, it compares the masked pressure value (0011) with the compare value (0011). Because they are equal, the instruction block will energize the rung.

Note that the masked comparison for equal instruction only compares the bits that are specified by ones in the mask. If the source word had contained any number other than 3, the comparison would have been false.

If you wanted to compare the volume data instead, you would have to specify a hex mask with the value 00F0 (Figure 5-9). This mask value will only pass and compare the data from the second thumbwheel switch. You would also have to enter an 80 as the compare value rather than 5. This is because the decimal value 80 translates into the binary value to be compared (i.e., the value 0101 located in bits 4 through 7). Because the com-

parison is true, the output will turn on. If you used the decimal value 5, it would generate the wrong binary comparison value (0101 located in bits 0 through 3). Thus, the MEQ instruction would not work correctly because the masked source value would not be identical to the compare value.

Limit Test Instruction

A **limit test instruction**, which is abbreviated as LIM, checks a value to see whether it is within a certain range (see Figure 5-10). It compares the test value to the low and high limit values. If the test value is between the high and low values or equal to them, the block's output energizes or de-energizes according to how the high and low parameters are defined.

In a limit test instruction, the test value can be either a constant or a word (see Figure 5-11). If the test value is a constant, the low and high limit values must be words. If the test value is a word, then the low and high limits can be either words or constants. The function of a limit test instruction depends on which is greater, the high limit value or the low limit value:

- If the high limit is greater than the low limit, the block's output will be on if the test value is between the two limits. The block's output will be off if the test value is lower than the low limit or greater than the high limit.
- If the low limit is greater than the high limit, the block's output will be off if the test value is between the two limits. The block's output will be on if the test value is greater than the low limit or lower than the high limit.

This high and low limit information is important to remember, especially when using word addresses for the high and low limit values. In this situation, the output continuity of the limit test block may reverse if the contents of the high and low limit words change.

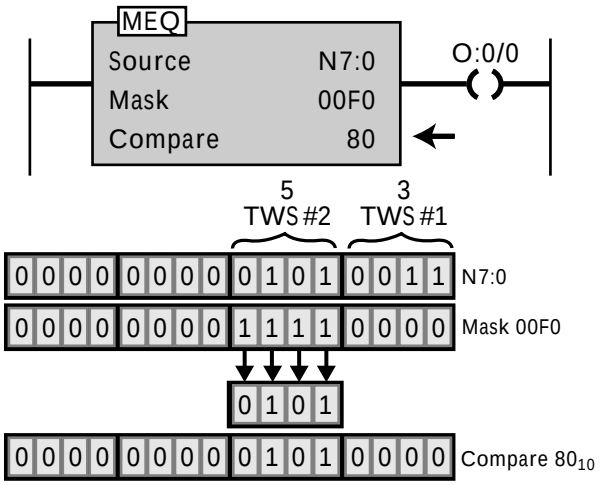

Figure 5-9. A ladder program in which output O:0/0 turns on when the pressure volume level is 5.

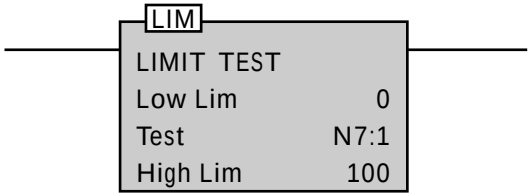

Figure 5-10. A limit test instruction.

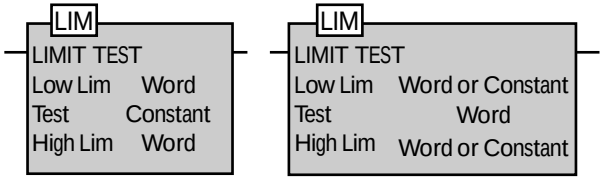

Figure 5-11. In a limit test instruction, the test value can be either a constant or a word.

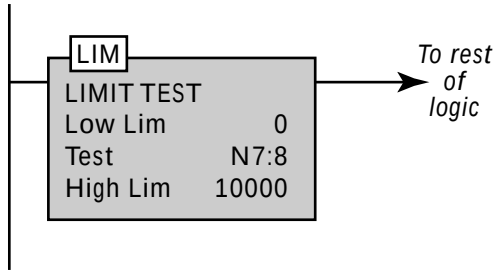

Figure 5-12. A limit test instruction used to ensure that an operator enters a valid parameter into the PLC.

Figure 5-12 illustrates is an example of how a limit test instruction can be used in a process application to ensure that an operator enters a valid parameter into the PLC. In this application, the limit test instruction's test value is the word location of the data entered by the operator (word N7:8). The low and high limit values are the minimum and maximum possible valid entries. Thus, if the test value falls within the valid entry range, the limit test instruction will provide continuity to the rest of the rung.

5-2 Flow Control Instructions—Alter Sequence

A MicroLogix 1000 uses three types of flow control instructions—those that alter the sequence of the control program's execution, those that halt its execution, and those that alter the I/O scan reading. This section covers the first type, those that alter the sequence of evaluation of the rungs in a ladder program. At the end of this section, you will know:

- the jump instruction
- the jump to subroutine instruction
- the master control reset instruction

Jump Instruction

A **jump instruction** (see Figure 5-13) is a coil instruction that jumps the PLC's program execution to a specified rung, thereby skipping those rungs programmed between the jump instruction and the destination rung. Because it does not reference a particular memory location, a jump instruction can have any numerical address between 0 and 999. A jump coil instruction works in conjunction with a label contact instruction, which specifies which rung to jump to.

Figure 5-14 illustrates how a jump instruction works. In this program, the output of the first rung is a jump coil with address 10. The fourth rung begins with a label contact that shares the jump coil's address. If input I:0.0/0 is true, jump coil 10 will be energized. This will cause the MicroLogix to jump to the rung containing the label instruction with address 10 and resume program execution from there. Thus, the jump instruction causes the PLC to skip the execution of rungs two and three. If the jump instruction is not energized, then the PLC will evaluate all the rungs of the ladder program as it would normally.

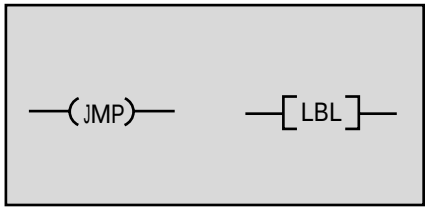

Figure 5-13. A jump instruction and a label instruction

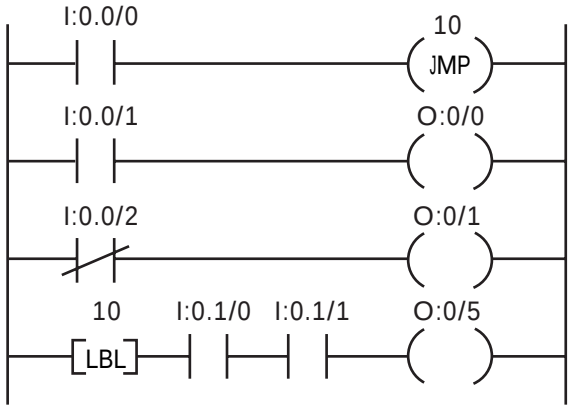

Figure 5-14. A ladder program containing a jump instruction.

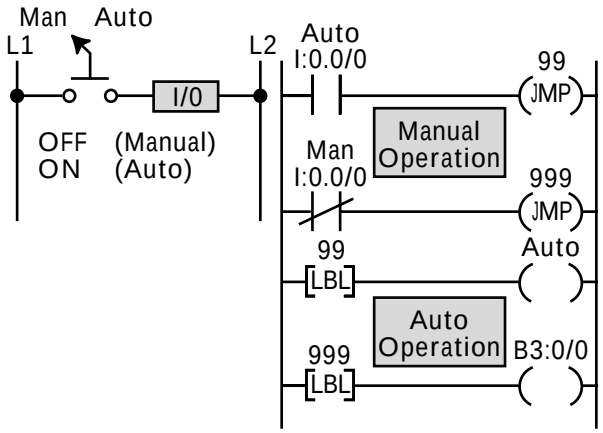

Figure 5-15. A jump instruction used to implement manual/automatic machine operation.

Figure 5-15 illustrates how a jump instruction can be used to control the manual or automatic operation of a machine. In this example, if the selector switch connected to input 0 is off, the machine will be in manual mode. If the switch is on, the machine will be in automatic mode. By using jump instructions, you can program two sets of instructions—one set for when the machine is in manual mode and another set for when the machine is in automatic mode. This avoids having to interlock the manual and automatic programming in all of the ladder instructions. Following is a description of how this program works:

- If the selector switch is on (automatic mode), the jump 99 coil in rung one will be energized.
- When the jump 99 coil is energized, the PLC will jump the program execution to the rung with label 99.
- When the controller jumps to label 99, it will start executing the automatic control program, thereby skipping the manual control program.
- Conversely, if the selector switch is off (manual mode), the controller will execute the manual control program because it will not jump over it.
- When the PLC finishes the manual program, the controller will encounter a jump 999 instruction that will cause it to jump over the automatic control program.
- When the controller jumps over the automatic program, it will jump to the rung with label 999. This rung contains a “dummy” coil that does not control anything. The purpose of this rung is to give the controller some place to jump to when it skips the automatic operation at the end of the program.

Jump instructions affect program execution in a variety of ways. A jump instruction that jumps forward reduces the program scan time, since it omits the execution of part of the ladder program. Jumping backwards has the opposite effect. It increases the scan time, since it causes the PLC to repeat part of the control program. Both jumping forward and backward are valid uses for jump instructions. In fact, you can even jump forward or backward several times to the same label using multiple jump instructions. Nevertheless, you should be careful not to jump backwards an excessive number of times. If you do, the processor's watchdog timer may time out and cause the controller to fault.

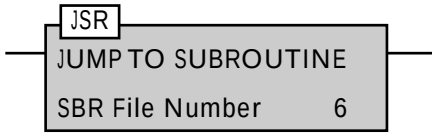

Figure 5-16. A jump to subroutine instruction.

Jump To Subroutine Instruction

A **jump to subroutine instruction** (see Figure 5-16) is used to call a subroutine from the main ladder program. When a jump to subroutine instruction in the main ladder program is enabled, it causes the program to jump to the specified subroutine located in the subroutine storage area. The controller then executes this subroutine until it finds a return or end instruction. At that point, it jumps back to the main program and resumes program execution with the instruction immediately following the jump to subroutine instruction.

Three instructions are associated with a jump to subroutine instruction (see Figure 5-17). These are:

- the subroutine instruction
- the return instruction
- the end instruction

Subroutine Instruction. A **subroutine instruction** should be programmed at the beginning of the subroutine being called. Although this instruction is not technically necessary, it is a good practice to use it for programming clarity.

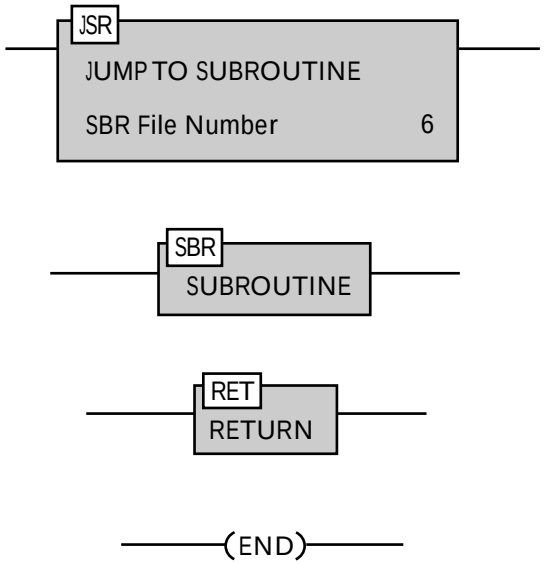

Figure 5-17. A jump to subroutine instruction and its three associated instructions—the subroutine instruction, the return instruction, and the end instruction.

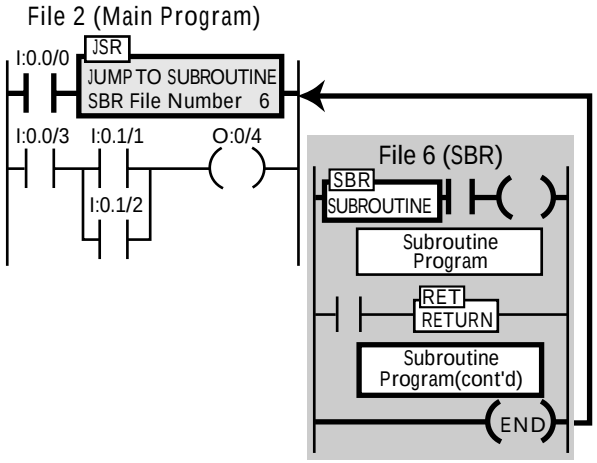

Figure 5-18. A jump to subroutine instruction that jumps to subroutine 6.

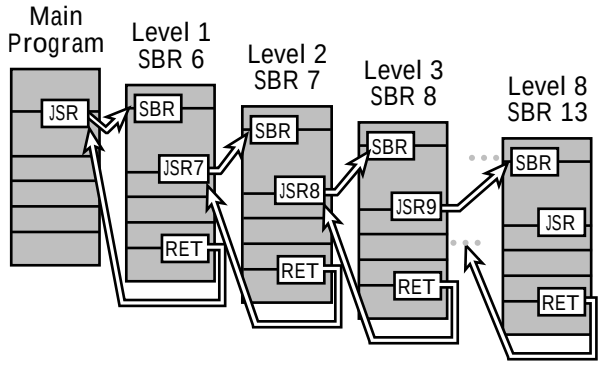

Figure 5-19. Nested subroutines in a MicroLogix 1000 program.

Return Instruction. A **return instruction** is used to tell the PLC to stop executing the subroutine and return to the main program. This instruction terminates a subroutine, either conditionally or unconditionally, before the whole subroutine program has been executed. If a subroutine does not contain a return instruction, the controller will execute the subroutine until it reaches the end instruction in the subroutine's file.

End Instruction. An **end instruction** is always present as the last instruction in a subroutine file, just as it is in the main ladder program and other program files. This instruction lets the PLC know that it has finished the subroutine.

Jump to Subroutine Operation. When a jump to subroutine instruction is enabled, the controller will jump to the subroutine specified and start executing it (see Figure 5-18). If the PLC encounters an energized return instruction in the subroutine, it will jump back to the main program and pick up where it left off. If it does not find an energized return instruction, the controller will wait until it gets to the end of the subroutine before jumping back to the main ladder program. When a subroutine is executed, its outputs remain in their last state, either on or off, until the subroutine is executed again.

As explained in Module 1, a MicroLogix 1000 stores the main ladder program in file 2 of the program file section. It stores the subroutines in files 6 through 15 of this same section. Each of the subroutine files can store one subroutine, for a total of 10 subroutines. Although these subroutines do not need to be programmed in the order in which they are called, you should do so anyway.

In a MicroLogix 1000, you can nest subroutines (see Figure 5-19). **Nesting** involves using one subroutine to call another subroutine. You can do this up to eight times in a MicroLogix program,

meaning that the subroutine calls can be eight levels deep. The controller will generate a *subroutine stack overflow error* if more than eight subroutines are nested in a program. Conversely, the controller will generate a *subroutine stack underflow error* if the program contains more return instructions than jump to subroutine instructions. Note that you can only nest three levels of subroutines if you are using the selectable timed interrupt and high-speed counter files to store additional subroutines.

To use subroutines to implement the manual/automatic selector switch program discussed earlier, you would use a rung containing two jump to subroutine instructions (see Figure 5-20). The first JSR instruction references subroutine 6, which stores the manual control program. The second JSR instruction references subroutine 7, which stores the automatic control program. Thus, in this control program, when the selector switch is off, the controller will jump to the manual subroutine and execute it. When the selector switch is on, the controller will jump to the automatic subroutine and execute it instead.

A jump to subroutine instruction can reduce the program scan time just as a jump instruction can, since it avoids the unnecessary evaluation of certain parts of the program. A jump to subroutine instruction can also simplify the main ladder program by allowing complex routines to be performed outside the main program. Moreover, subroutines can be used to program tasks that occur many times in the main ladder program. A lookup table routine is a good example of this kind of task. By using jump to subroutine instructions to go to the subroutine for this repetitive task, you can avoid having to program the task over and over again in the main control program.

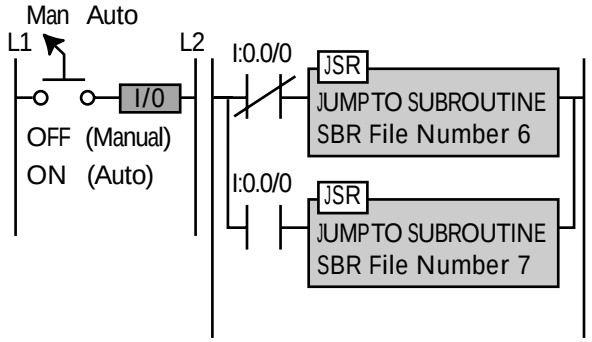

Figure 5-20. Jump to subroutine instructions used to control the manual or automatic operation of a machine.

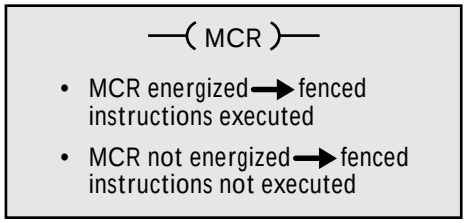

Figure 5-21. A master control reset instruction.

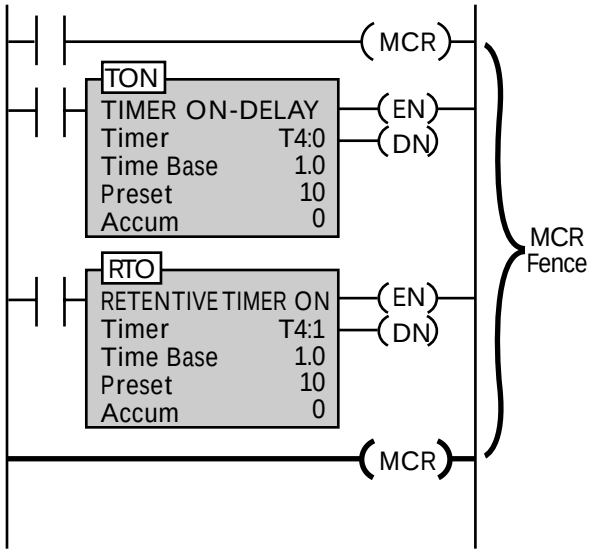

Figure 5-22. A ladder program with two fenced timers.

Master Control Reset Instruction

A **master control reset instruction**, which is also called an MCR instruction (see Figure 5-21), creates a fence around a group of ladder rungs. If the MCR instruction is energized, then the controller will execute the fenced instructions. If not energized, the controller will not execute the fenced instructions.

Master control reset instructions are always used in pairs to form a conditional fence around a group of rungs. If the input logic to the first MCR is energized, the programmed logic within the fence will be executed. If the input logic to the first MCR is not satisfied, then the controller will skip the fenced logic and resume program execution after the second MCR instruction. This second MCR instruction must be unconditional, meaning that it is always active because it has no driving input logic.

When an MCR fence is deactivated, all of the nonretentive outputs within the MCR fence will turn off, regardless of the status of their input conditions. Only the retentive outputs will retain their last status, either ON or OFF.

Figure 5-22 shows a ladder program with two fenced timers to demonstrate how MCRs work. One of these timers is an ON-delay timer; the other is a retentive timer. Both timers are located within the MCR fence. This means that neither timer will start timing, even if its input is on, unless the first MCR instruction is on. When the first MCR instruction turns on, both timers will be enabled if their inputs are on. Thus, the timers will start timing, and if the MCR zone stays on long enough, both timers will time out. When the first MCR eventually turns off, the ON-delay timer's output will turn off, since all nonretentive outputs in an MCR fence are reset when the fence turns off. The retentive timer's output will remain on.

The circuit in Figure 5-22 works much the same way if the timers start timing and then the MCR fence turns off before either timer has timed out. In this situation, the timers will start timing when the first MCR instruction turns on. However, when the MCR turns off, the timers will turn off too, even though their inputs may still be on. As a result, the accumulated value of the ON-delay timer will be reset to 0. The retentive timer, however, will retain its accumulated value. To reset the retentive timer's accumulated value, you would need to add a reset instruction outside of the MCR fence (see Figure 5-23). If this reset instruction was located inside the fence, it could only reset the retentive timer when the MCR zone was activated.

When using MCR instructions, you should never use a jump instruction to jump inside an MCR zone. If you do, the logic you jump to inside the MCR fence will be evaluated, regardless of whether the MCR is on. This can cause a hazardous situation. Also, you should be aware that although nesting is permitted with subroutines, it is not allowed with MCR zones.

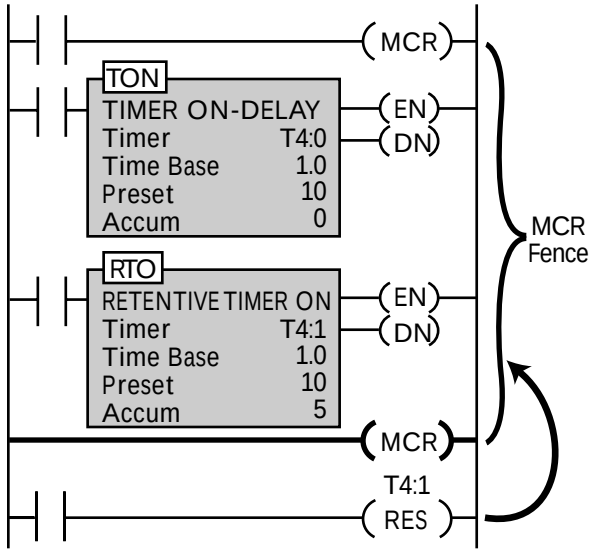

Figure 5-23. A ladder program with two fenced timers that uses a reset instruction to reset the RTO instruction.

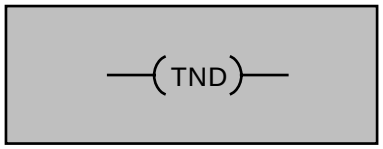

Figure 5-24. A temporary end instruction.

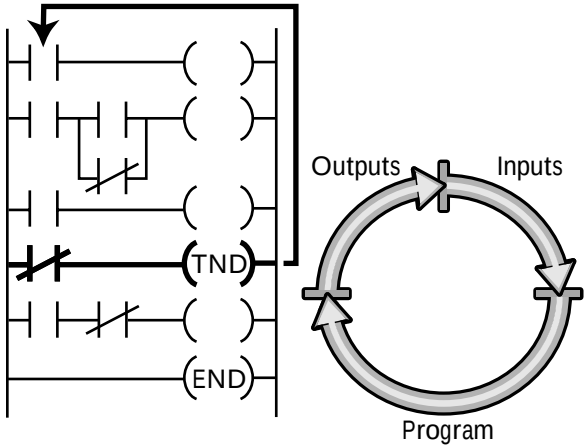

Figure 5-25. The operation of a temporary end instruction.

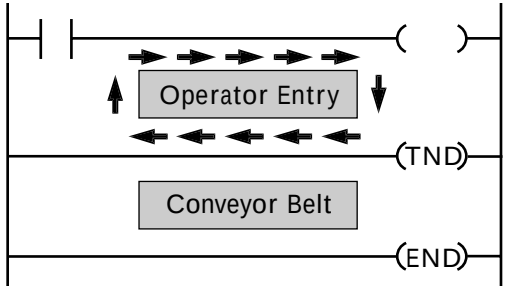

Figure 5-26. A control program with two sections—one section that evaluates an operator's entry values and another section that runs a conveyor belt.

5-3 Flow Control Instructions—Halt Execution

This section discusses flow control instructions that interrupt the execution of the control program. Two flow control instructions perform this type of function. They are:

- the temporary end instruction
- the suspend instruction

Temporary End Instruction

A **temporary end instruction** (see Figure 5-24) is a coil instruction that stops the execution of the control program at the point where the temporary end coil is located. It works as follows (see Figure 5-25):

- If the temporary end coil is on, the PLC will stop executing the control program at the temporary end instruction and perform an update of the outputs.
- The PLC will then begin its next scan by reading the inputs and executing the control program again, starting from the top.
- If the temporary end coil is not on as the PLC performs this next scan, the controller will continue with its scan until it reaches either the end of the program or another energized temporary end instruction.

Temporary end instructions are often used to test or troubleshoot certain parts of a control program. They allow you to test part of the control program with the inputs and outputs on-line, without having to run through the rest of the program. For example, Figure 5-26 shows a control program with two sections—one section that evaluates an operator's entry values and another section that runs a conveyor belt. If you want to test the

operator entry section without turning on the conveyor, then you would program an unconditional temporary end after the first section. This allows you to run just the top section over and over again until you know that it is working properly. After that, you can take out the temporary end instruction and run the whole program.

You can use a temporary end instruction in the main program and any of its subroutines stored in files 6 through 15. However, you cannot use a temporary end instruction in the user-error fault routine (file 3), the high-speed counter interrupt routine (file 4), or the selectable timed interrupt routine (file 5). If you do, the PLC will generate a fault.

Suspend Instruction

Like a temporary end instruction, a **suspend instruction** (see Figure 5-27) is used for testing or troubleshooting a control program. A suspend instruction causes the controller to stop executing the control program and enter an idle mode. When it does this, the controller de-energizes all outputs. However, it does not clear the bits in the status file. This allows you check the status bits to retrieve information about the controller's operation and why it stopped.

A suspend instruction has an ID number that can range between –32,768 and +32,767. When a suspend instruction is activated, its ID number is stored in word 7 of the status file (see Figure 5-28). This indicates where the controller was in the control program when its operation was suspended.

A suspend instruction is useful, for example, for halting the control program when an overflow occurs to allow troubleshooting. To do this, you would program a suspend instruction, such as the one in Figure 5-29, which operates as follows:

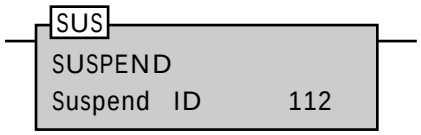

Figure 5-27. A suspend instruction.

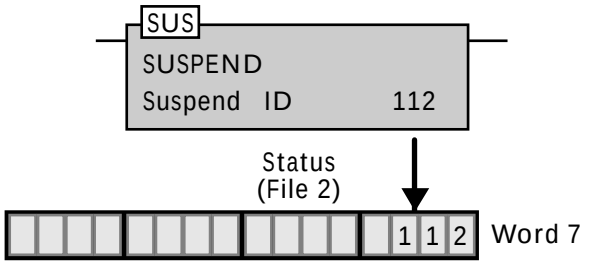

Figure 5-28. When a suspend instruction is activated, its ID number is stored in word 7 of the status file.

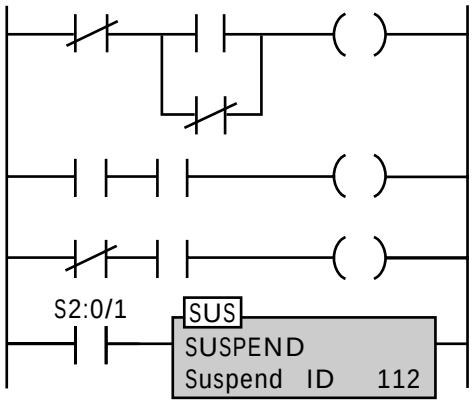

Figure 5-29. A suspend instruction used to halt the control program when an overflow occurs to allow troubleshooting.

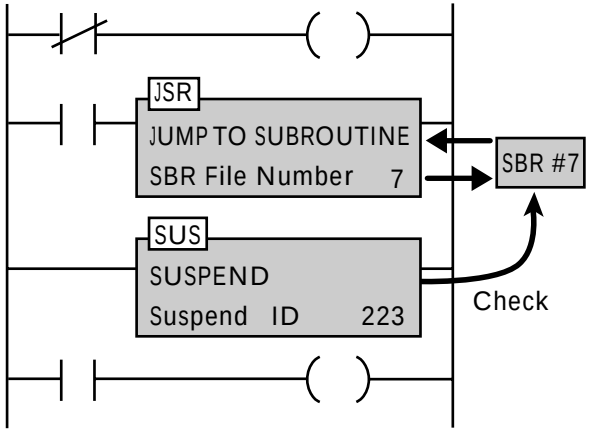

Figure 5-30. A suspend instruction used after a subroutine to allow a check of the subroutine's data before resuming main program execution.

- The suspend instruction's input logic is a contact referencing a math overflow bit (S:0/1).
- If this overflow bit is energized, the suspend instruction will be energized. Hence, the controller will halt the program execution.
- At this point, you can review the logic programmed before the suspend instruction to see why the overflow occurred.
- The ID number, which is 112 in this case, indicates that the suspend was triggered by a math overflow. Other suspend instructions may have different IDs to indicate other reasons for the program halt.

You can also use a suspend instruction after a subroutine to allow you to check the subroutine's data before resuming the main program (see Figure 5-30). In some critical applications, you may also use one or more inputs to drive one or more suspend instructions. This kind of programming turns off the outputs if certain critical error conditions occur.

5-4 Flow Control Instructions—Alter I/O Scan

This section discusses two flow control instructions that alter the I/O scan evaluation of the controller. These are:

- the immediate input with mask instruction
- the immediate output with mask instruction

Immediate Input With Mask Instruction

An **immediate input with mask instruction** , which is abbreviated as IIM, interrupts program execution to update the specified input data (see Figure 5-31). When energized, an immediate input with mask instruction masks the data in the specified input word to obtain just the data to be updated. Then it interrupts normal program execution to store this data to the input file. Thus, an immediate input instruction updates input data without having to wait until the beginning of the next scan.

In an immediate input instruction, the slot parameter indicates the input word to be updated. This will be either I:0.0 or I:0.1. The mask specifies which bits will be masked. This mask, which is expressed in hexadecimal, works like the mask value used by the masked move instruction.

An immediate input with mask instruction works as follows (see Figure 5-32):

- When energized, an immediate input with mask instruction checks the inputs mapped to the specified word for their current status.
- Then it performs a mask to filter out all but the input bits specified.

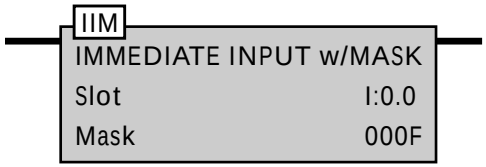

Figure 5-31. An immediate input with mask instruction.

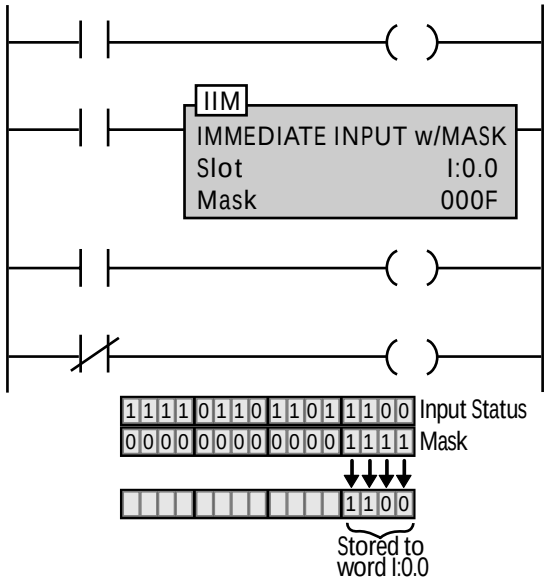

Figure 5-32. The operation of an immediate input with mask instruction.

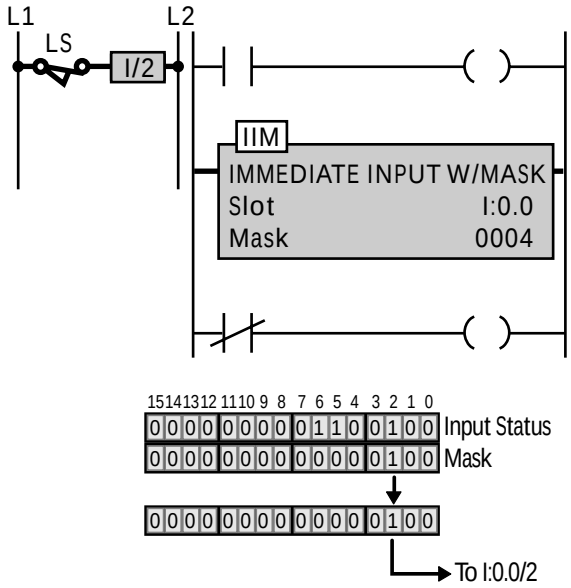

Figure 5-33. An immediate input with mask block used to update the status of a limit switch connected to input terminal 2 of a MicroLogix 1000.

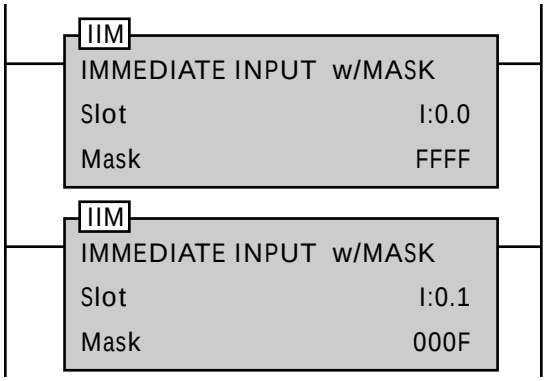

Figure 5-34. Two consecutive immediate input with mask instructions used to update all the input bits of a 32 I/O MicroLogix.

- It then stores this data to the input word denoted by the slot parameter. The input bits that are not masked are not updated; thus, these bits are left in their previous state.
- The PLC uses the updated input data to evaluate all the rungs located after the immediate input instruction.

Figure 5-33 shows an example of an IIM application in which a limit switch is connected to input terminal 2 of a MicroLogix 1000. To update the data about this input during the ladder program, the program contains an IIM instruction whose slot parameter is I:0.0, since this is the input word that maps the limit switch. The IIM instruction's mask value is 0004 to mask out all but the data for bit 2. Thus, if the limit switch was off when the PLC started its scan but has since turned on, the immediate input instruction will interrupt the scan to read the current status of the limit switch. It will then update the new data about the limit switch's logic status in address I:0.0/2. The rest of the program will now use this updated input data.

The mask address specified in an IIM instruction determines how many inputs will be updated by the instruction. To update all of the inputs for a 16 I/O MicroLogix, you must specify a slot parameter of I:0.0 and a mask value of 03FF. This will update all 10 input bits of the MicroLogix's input file. To update all the input bits of a 32 I/O MicroLogix, you must program two consecutive immediate input instructions (see Figure 5-34). The first instruction must have a slot parameter of I:0.0 and a mask of FFFF to update all the bits of this word. The second instruction must have a slot parameter of I:0.1 and a mask of 000F, since a 32 I/O MicroLogix only uses the first four bits of word I:0.1 for inputs.

Immediate Output With Mask Instruction

An **immediate output with mask instruction**, which is abbreviated as IOM, interrupts program execution to update the specified output data, which is located in output word O:0 (see Figure 5-35). When energized, an immediate output instruction updates the outputs specified by the mask. It interrupts the normal program execution to store this data to the output file. Like an immediate input instruction, an immediate output instruction specifies slot and mask parameters. The slot parameter indicates the output word to be updated. The mask indicates which outputs will be updated.

To illustrate how an IOM instruction works, Figure 5-36 shows two pilot lights that are connected to output terminals 0 and 1 of a MicroLogix 1000. At the end of the last scan, both lights were on, but now, according to the current logic, they should both be off. To update these outputs during the scan, you can program an immediate output with mask instruction with a slot parameter of O:0 and a mask of 0003. When energized, the IOM instruction will update the status of the output file and mask out all but the data for bits 0 and 1. It will then send this data to PL1 and PL2, turning them off immediately instead of waiting until the update output section of the scan.

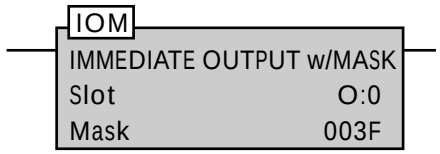

Figure 5-35. An immediate output with mask instruction.

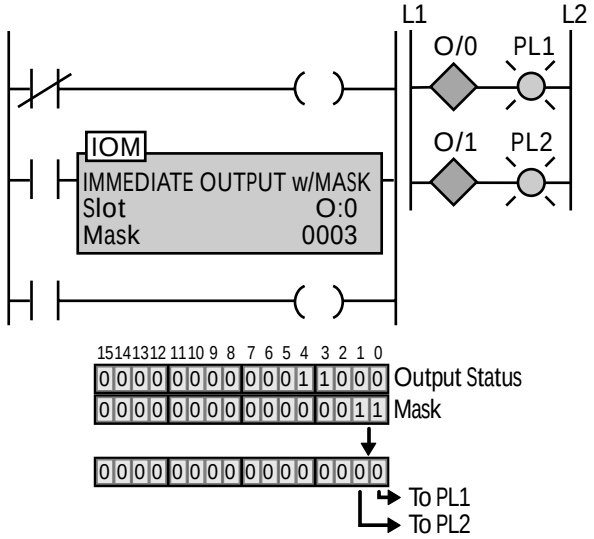

Figure 5-36. An immediate output with mask instruction used to update the status of two pilot lights.

5-5 Review

- An equal instruction compares two values to see if they are equal to each other.
- Other basic comparison instructions (not equal, less than, less than or equal, greater than, and greater than or equal) also compare two values, but they each test for a different comparison condition.
- The masked comparison for equal instruction uses a mask value to compare part of the data in the source location to a comparison value.
- A limit test instruction checks to see if a test value falls within a certain range of values.
- A jump instruction causes a controller to jump over the execution of a certain set of ladder rungs.
- A jump to subroutine instruction causes the controller to stop executing the main control program and start executing a subroutine program.
- A master control reset instruction forms a conditional fence around a set of ladder rungs.
- A temporary end instruction stops program execution before the end of the whole ladder program.
- A suspend instruction halts program execution and turns off the outputs while maintaining the status file data.
- An immediate input with mask instruction forces the PLC to halt program execution and immediately update the status of the input devices.
- An immediate output with mask instruction forces the PLC to halt program execution and immediately update the status of the output devices.