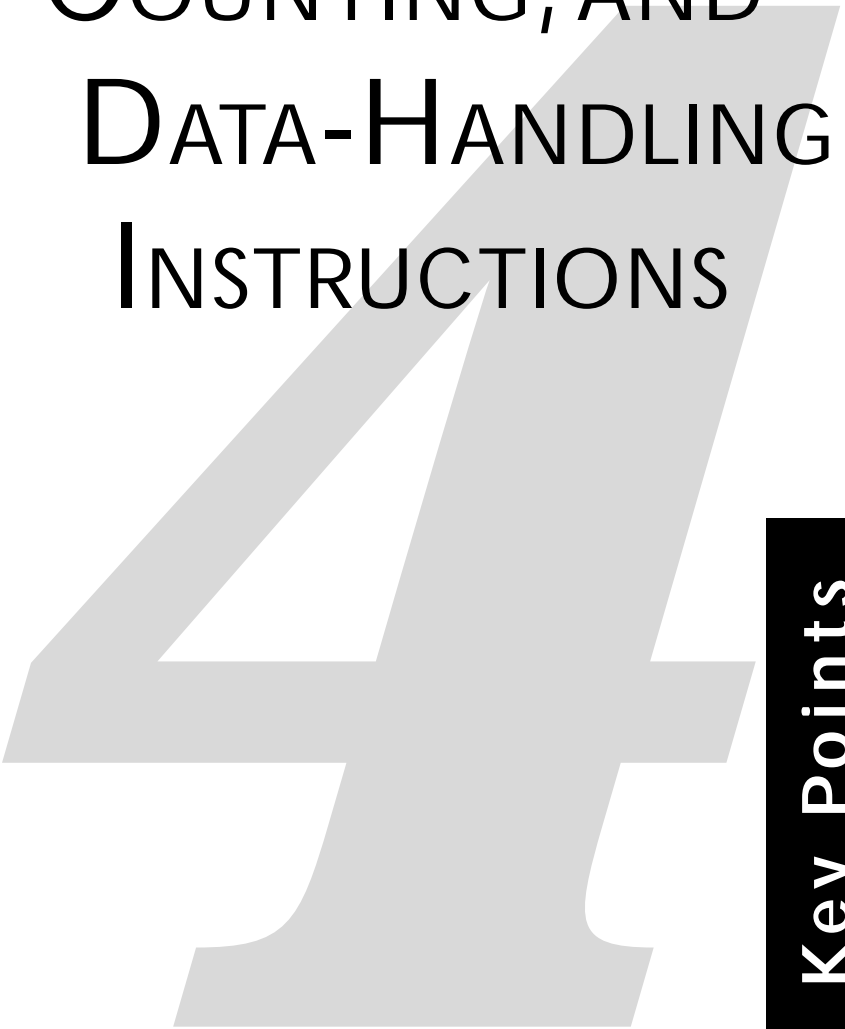


TIMING, COUNTING, AND DATA-HANDLING INSTRUCTIONS



This module is a further exploration of the MicroLogix 1000's programming instructions. Module 3 covered basic relay instructions, which perform simple ON/OFF operations. All of those instructions use a basic ladder format. The three sections of this module discuss programming instructions that are represented in block format. These sections are:

1. Timing instructions
2. Counting instructions
3. Data-handling instructions

Key Points

After finishing this module, you will:

- understand the three timing instructions used in a MicroLogix 1000—timer ON-delay, timer OFF-delay, and retentive timer—as well as the values and special programming issues associated with each
- understand the count up and count down counting instructions and the reset instruction, including the values and special programming issues associated with each
- know how to use data-handling instructions to move and convert data in a MicroLogix 1000 PLC

4-1 Timing Instructions

Timing instructions are programming instructions that replace the need for electromechanical timers in a control system. Timing instructions perform the same function as electromechanical timers, but they are more accurate, do not cost extra, and save space.

At the end of this section, you will know:

- timer basics, including timer values and addresses
- the operation of a timer ON-delay instruction
- the operation of a timer OFF-delay instruction
- how a retentive timer instruction works
- how to use and implement the trapping of instantaneous timer contacts in a PLC

General Timer Information

Timer Values. A timer instruction has three important values associated with it:

- the time base
- the preset value
- the accumulated value

Time Base. The **time base** is the unit of time used by a timer to time an event. A MicroLogix 1000's timers can have a time base of either 0.01 seconds or 1 second. A timer instruction times an event by counting the number of times the time base has occurred since the instruction was energized. For example, if a MicroLogix has a time base of 1 second and it is timing something that is 2 seconds long, the PLC will wait until the time base has occurred 2 times before the timer times out (see Figure 4-1).

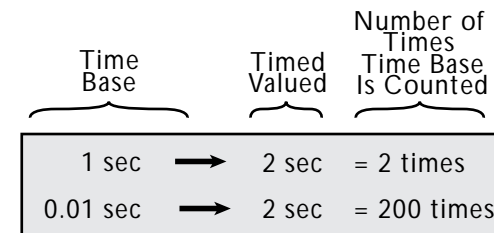


Figure 4-1. Time base illustration.

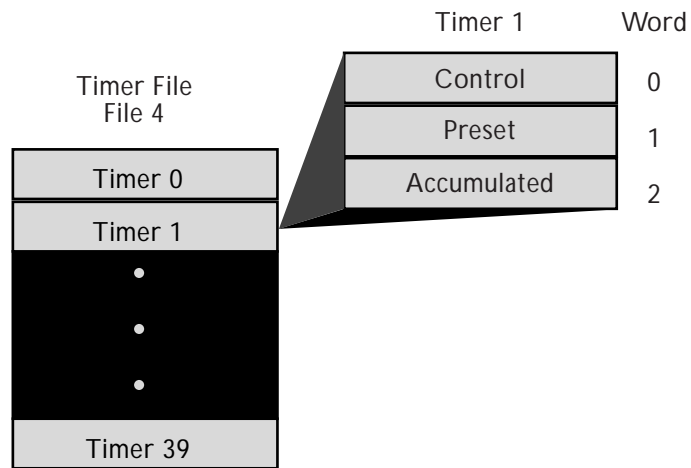


Figure 4-2. The timer file showing the three words associated with each timer.

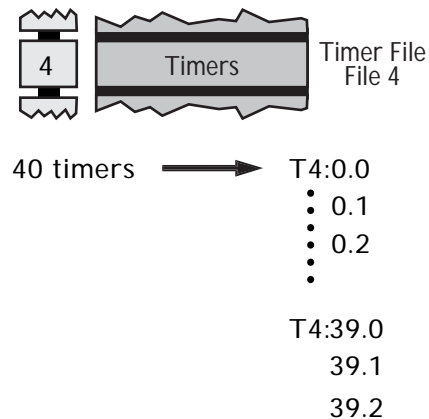


Figure 4-3. MicroLogix 1000 timer addressing.

Conversely, if the PLC's time base setting is 0.01 seconds, it will wait until the time base has occurred 200 times before timing out. The selection of the time base depends on what is most appropriate for the application.

Preset Value. The **preset value** of a timer works in conjunction with the time base by specifying the number of times that the timer must count the time base. This preset value, which is also referred to as the number of *ticks*, is predetermined and preprogrammed by the user. Thus, in the previous example of a timer with a 0.01 time base and a target value of 2 seconds, the preset value would be 200. This value indicates that the timer must wait 200 time bases before timing out.

Accumulated Value. The final value associated with a timer is the **accumulated value**. This value keeps track of how many times the time base has occurred since the timer instruction was energized. When the accumulated value equals the preset value, the timer will time out because it has reached its target timing value. So if a timer has a time base of 0.01 and a preset value of 200, the accumulated value will increase by one every 0.01 seconds until the accumulated value equals 200. At that point, the timer instruction will time out.

Addressing. A MicroLogix 1000 stores data about timers in file 4 of its data file section. This file can store the data of up to 40 timers, numbered 0 through 39. Each of these timers has three words associated with it (see Figure 4-2). Therefore, the available addresses in the timer file range from T4:0.0 to T4:39.2 (see Figure 4-3).

Each of the three words associated with a timer holds a specific kind of data (see Figure 4-4):

- *Word 0* holds control data about the status of the timer's enable output, whether the timer is actively timing, and the status of the timer's done output. The control word stores this information in bits 15, 14, and 13, respectively.
- *Word 1* stores the timer's preset value. This is the target timing value specified in memory.
- *Word 2* holds the accumulated value. This value indicates how much time has actually elapsed since the timer was energized.

In the RSLogix software, the labels PRE and ACC are used to denote timer words 1 and 2, respectively. Thus, timer words T4:0.0, T4:0.1, and T4:0.2 are represented as T4:0, T4:0.PRE, and T4:0.ACC in the RSLogix software.

Timer ON-Delay Instruction

The **timer ON-delay instruction** is a block-format instruction that is represented by the symbol shown in Figure 4-5. This block has two outputs:

- an enable output coil
- a done output coil

Inside the block is information about the timer's address, time base, preset value, and accumulated value. A timer ON-delay instruction energizes its done output after the timer block's input turns on and a specified delay has occurred. Consequently, this instruction is sometimes called a *timer ON-delay energize instruction*.

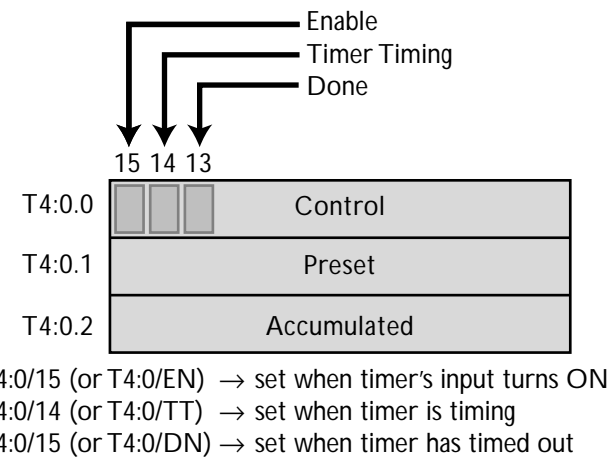


Figure 4-4. The data stored in each word of a timer's address.

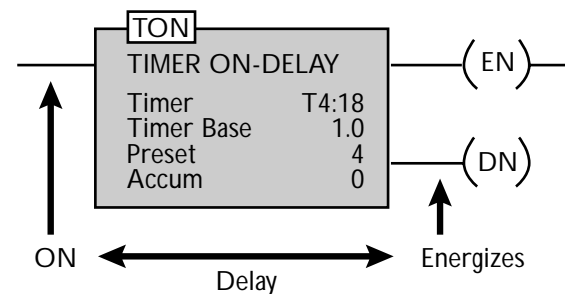


Figure 4-5. A timer ON-delay instruction.

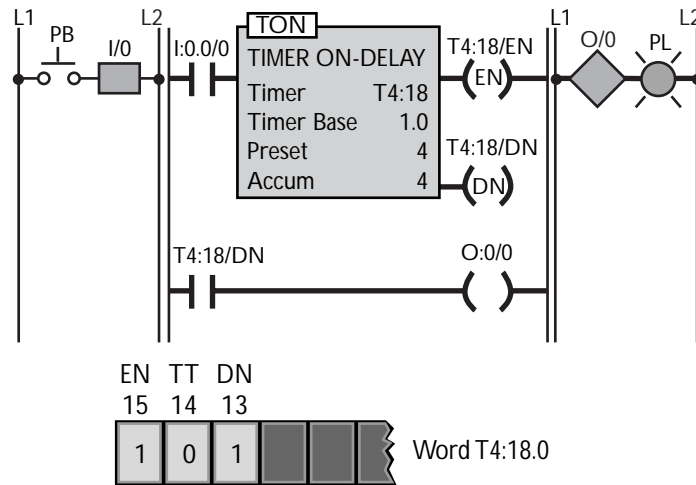


Figure 4-6. The operation of a timer ON-delay block in a control program.

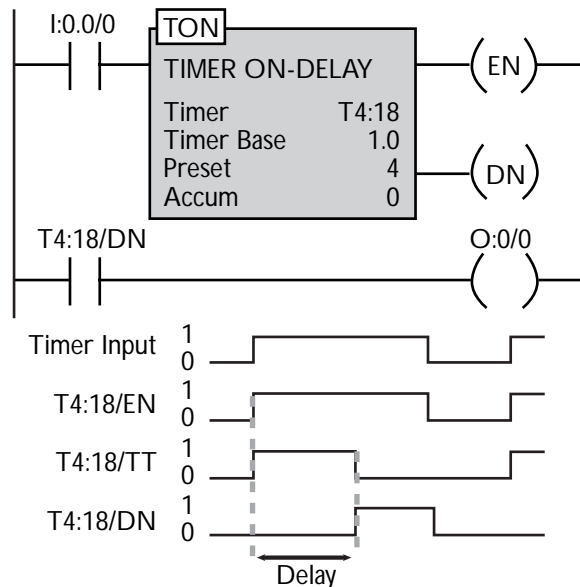


Figure 4-7. A timer ON-delay block and its associated timing diagram.

Figure 4-6 illustrates how a timer ON-delay instruction works. When the timer block's input has logic continuity, the block's enable output will turn on. As a result, a 1 will be stored in bit 15 of the timer's control word. Once the timer is enabled, it will start to time. Thus, a 1 will be stored in bit 14, which is the timer timing bit. As the timer times, the accumulated value increases until it equals the preset value. At that point, the timer timing bit will become a 0, and the done bit will become a 1, meaning that the done output will turn on. This done output is the timer's delay action contact.

The timer-ON delay instruction's enable output will remain on as long as the input logic to the block remains energized. However, the timer will stop timing as soon as the accumulated value equals the preset value. The timer's input logic must turn off and then on again before the timer will start timing again. The timer's done output can be referenced throughout the program by XIC and XIO contacts to implement the time delay.

In the ladder program shown in Figure 4-6, the pilot light output will turn on four seconds after the push button input is pressed. In the ladder diagram, the input logic to the pilot light is a contact that references the done output coil of the timer block. The timer's address is T4:18, its preset value is 4, and its time base is 1 second.

Figure 4-7 shows the operation of the same ladder diagram, using a timing diagram to keep track of when the input and outputs turn on and off. The ladder diagram operates like this:

- When the timer's input turns on, it will cause the timer's enable output to turn on. When this happens, the timer will start timing, but the done output coil will remain off. The time between the timer being energized and the done output being energized is the four-second delay implemented by the timer.

- The timer will stop timing as soon as the accumulated value equals the preset value.
- When the accumulated and preset values are equal, the done output will turn on, causing the output coil that drives the pilot light to turn on.
- The done output—and hence, the pilot light—will stay on until the timer block’s input turns off. At that time, everything in the ladder rung will turn off, and the timer’s accumulated value will be reset to 0.

Timer OFF-Delay Instruction

Figure 4-8 illustrates a **timer OFF-delay instruction**. This instruction looks much like a timer ON-delay instruction in that it has two outputs—done and enable—and includes information about the timer’s preset and accumulated values. Although a timer OFF-delay instruction may look like an ON-delay instruction, it works a little differently. A timer OFF-delay instruction de-energizes its done output after the timer block’s input turns off and a specified delay has occurred. Thus, the timer OFF-delay instruction is also called a *timer OFF-delay de-energize instruction*.

The ladder program in Figure 4-9 uses a timer OFF-delay instruction. This circuit works as follows:

- The done output will be off when the program is first started and the timer’s input is off.
- When the input logic turns on, both the block’s enable output and done output will turn on. However, the timer will not start timing because it is waiting for an OFF signal instead of an ON signal.

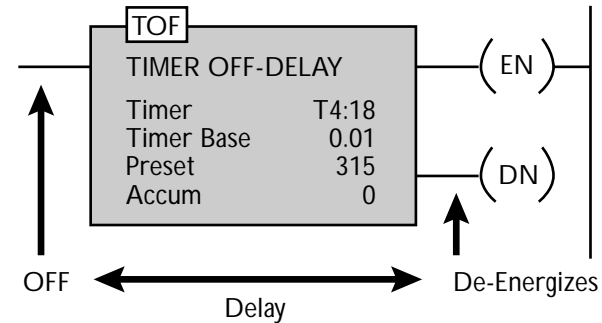


Figure 4-8. A timer OFF-delay instruction.

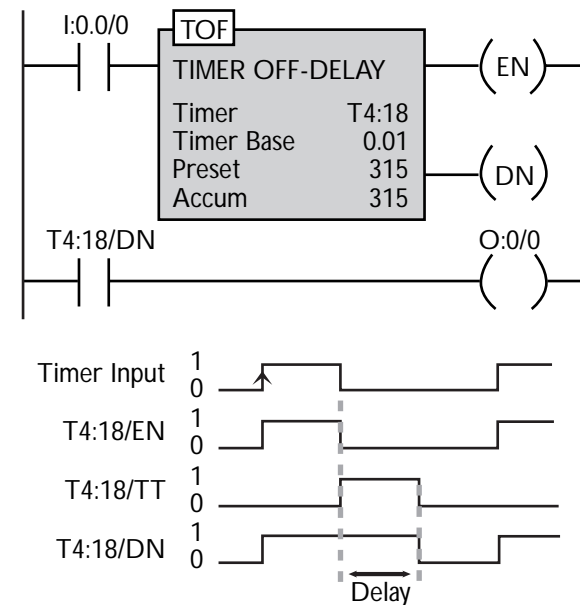


Figure 4-9. A timer OFF-delay block and its associated timing diagram.

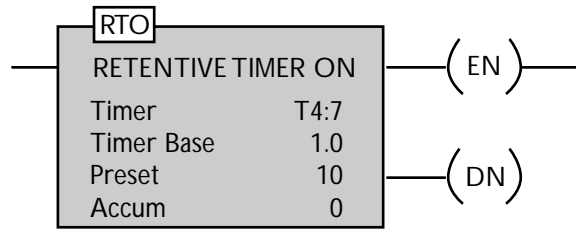


Figure 4-10. A retentive timer instruction.

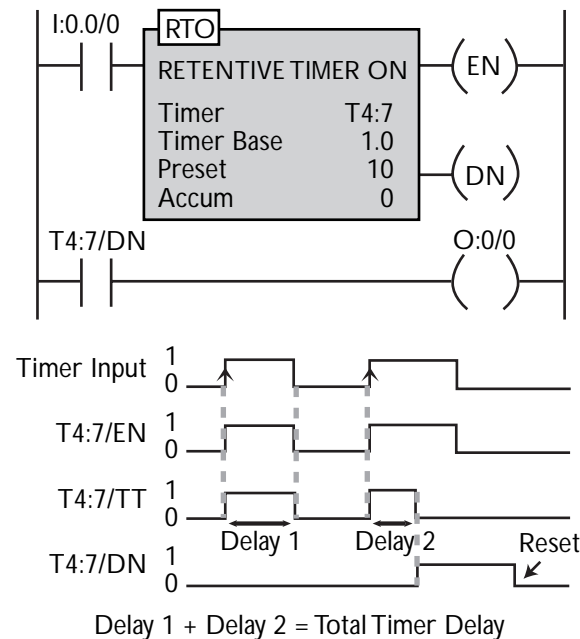


Figure 4-11. A retentive timer circuit and its associated timing diagram.

- When the block's input turns off, the enable output will turn off and the timer will start timing. The done output will stay on because it is waiting for the timer to time out before it will turn off.
- Once the accumulated value equals the preset value, the timer will stop timing and the done output will turn off, implementing the OFF-delay de-energize function.
- Therefore, the done bit's action follows the action of the timer's input signal, except that the done bit remains on for the specified delay period after the input turns off. All of the timer's outputs will now remain off until the input logic turns on again. At this point, the accumulated value is reset to 0.

Retentive Timer Instruction

A **retentive timer instruction**, pictured in Figure 4-10, operates much like a timer ON-delay instruction. A retentive timer, however, can stop timing and then start timing again without its accumulated value resetting to 0.

Figure 4-11 shows a retentive timer circuit and its timing diagram, which work as follows:

- When the input logic turns on, the enable output will turn on, and the timer will start timing.
- If the input logic turns off, the enable output will turn off, and the timer will stop timing. The accumulated value, however, will not reset to 0.
- When the timer starts timing again, it will pick up where it left off.
- When the accumulated value finally reaches the preset value, the done output will turn on.

Once a retentive timer has timed out, its done output will remain on even if its input logic and enable output turn off. A reset instruction must be used to turn the done output off and reset the timer's accumulated value. The operation of a reset instruction is explained in the counter section of this module.

Trapping

Trapping is a special timer programming issue. The electromechanical timers used in hardwired circuits have two kinds of contacts:

- time-delayed
- instantaneous

The **time-delayed contact** is used to turn on the output after the timer has timed out. The **instantaneous contact** is used to seal the timer's input so that, once the timer has started timing, it will continue to time even if its input logic turns off. This provides interlocking in the circuit.

Figure 4-12 illustrates how the two types of timer contacts are represented in an electromechanical diagram. An instantaneous contact is represented by a contact symbol, and a time-delayed contact is represented by a timer switch symbol. The symbol for an ON-delay timer's time-delayed contact has an arrow that points up. This indicates that the contact energizes, or closes, after the delay following the input's OFF-to-ON transition. In contrast, the time-delayed contact for an OFF-delay timer points down, indicating that it turns off, or opens, after the delay following the input's ON-to-OFF transition.

In contrast to electromechanical timers, PLC timers have only one type of contact—a time-delayed contact. This contact must not be confused with an instantaneous contact when replacing relay logic. Consequently, you must use trapping to implement

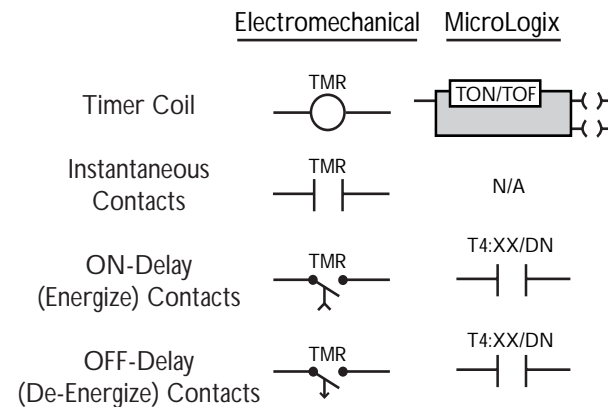


Figure 4-12. Instantaneous and time-delayed timer contacts as represented in both an electromechanical system and a MicroLogix system. The XX in the MicroLogix timer labels symbolizes the timer address.

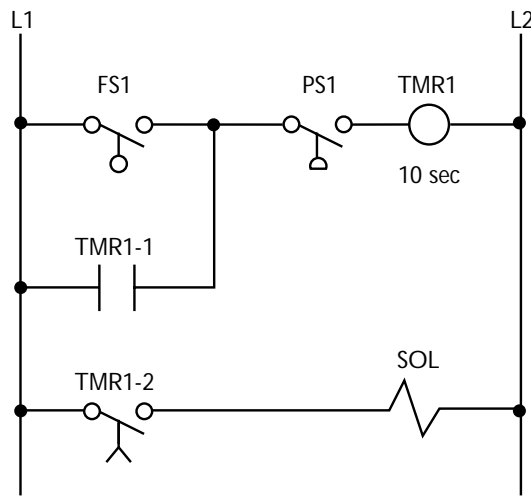


Figure 4-13. An electromechanical timer circuit to be implemented in a PLC.

an instantaneous contact in a PLC timer circuit, if the application requires it. When you trap a circuit, you use an internal contact and coil to seal the timer on. You can also use the enable output of the timer to trap the circuit.

Trapping Circuit—Internal Output. Figure 4-13 shows an electromechanical timer circuit that will be implemented in a PLC. In this circuit, the timer will start timing as soon as float switch FS1 and pressure switch PS1 close. The timer will continue to time even if the float switch turns off. This is because instantaneous contact TMR1-1 will seal the timer's input logic. After a 10-second delay, the timer will energize time-delayed contact TMR1-2, causing the solenoid to turn on.

The first step in making this a PLC circuit is to determine which devices will be connected to the PLC's I/O interfaces. In this case, only the float switch, pressure switch, and solenoid will be connected to the PLC. The rest of the circuit will be implemented through PLC instructions. The float switch will be connected to the MicroLogix's first input terminal, and the pressure switch to the second terminal. The solenoid will be connected to the first output terminal.

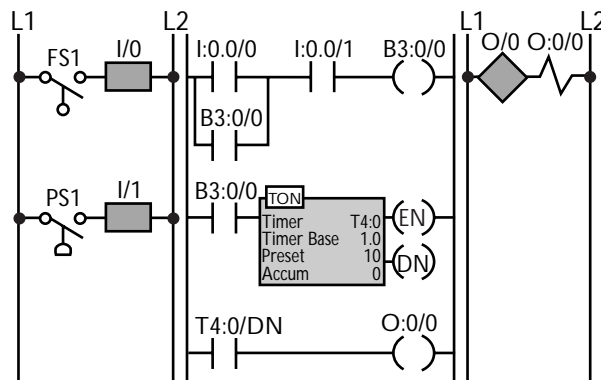


Figure 4-14. The timer circuit implemented in a ladder program using an internal output to trap the timer.

Figure 4-14 illustrates the ladder program that will implement the timer circuit in the PLC. This ladder program contains three rungs:

- The first rung traps the timer on.
- The second rung implements the timing function.
- The third rung implements the time-delayed output action.

Rung 1. The first rung in the ladder program consists of contacts that reference the float switch and pressure switch, along with an internal coil. It also contains an internal contact that references the internal coil, which implements the trap. When the

float switch and pressure switch turn on, the internal coil will turn on. Because contact B3:0/0 seals the input, the internal coil will stay on even if the float switch turns off. Thus, contact B3:0/0 performs the function of an instantaneous contact.

Rung 2. Rung two actually implements the timer. When the internal output coil in the first rung energizes, the timer will start to time because its input logic will be satisfied. Once the accumulated value equals the preset value, the timer's done output will turn on because the 10-second delay will be satisfied.

Rung 3. Rung three controls the solenoid output. When the done output in rung two turns on, the solenoid output will turn on because its input logic references the done output coil. Therefore, this PLC circuit implements both the instantaneous and time-delayed contacts of the hardwired circuit through the use of internal contacts.

Trapping Circuit—Enable Output. Another way to trap an instantaneous timer contact is to use a contact that references the timer's enable output. In this method, the enable contact is used to seal the timer's input, instead of an internal coil and contact. Figure 4-15 shows an example of this type of trapping. When the float switch and pressure switch turn on, the enable output and its corresponding contact will turn on. However, if the float switch opens, the timer will remain on because the enable contact will trap it.

Figure 4-16 shows a multispeed motor. In low speed, this motor operates in a delta configuration. In high speed, it operates in a wye configuration. In this motor, if the low push button is pressed, the motor will run at low speed. If the high push button is pressed after being in low speed, the motor will run at high speed. The timer in this circuit ensures that a 10.8-second delay occurs before the motor will run in high speed. Thus, if the high push button is pressed, the motor will first start out at low speed and

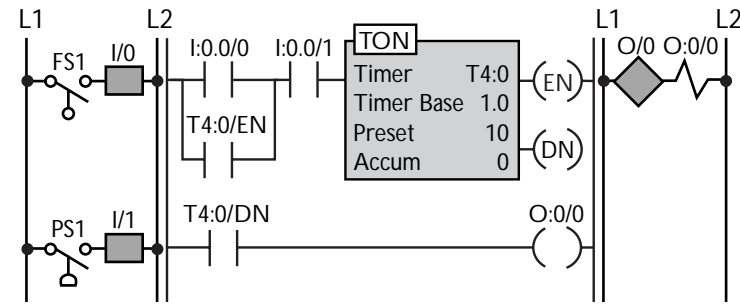


Figure 4-15. The timer circuit from Figure 4-13 implemented in a ladder program using the enable output to trap the timer.

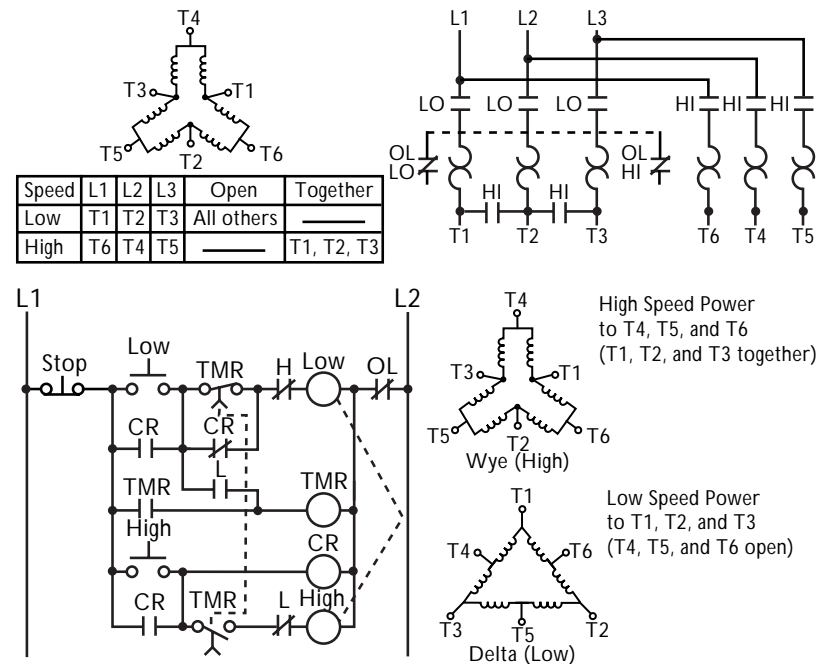


Figure 4-16. A multispeed motor circuit.

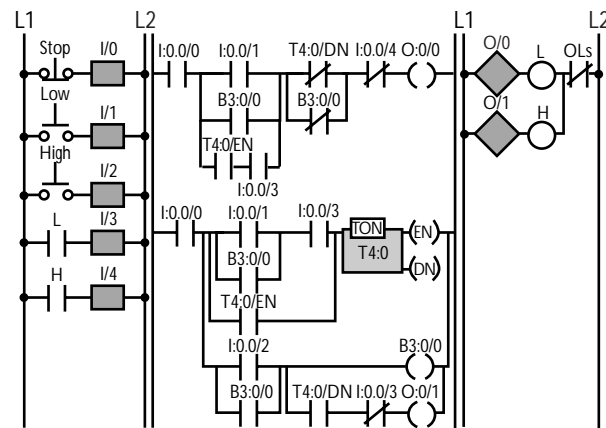


Figure 4-17. Multispeed motor ladder program using the timer's enable output to trap the timer.

rev up to high speed after 10.8 seconds. Notice that the circuit uses interlocking motor starter contacts so that the high-speed starter coil will not turn on until the time delay has occurred.

When this circuit is implemented in a MicroLogix, the stop, low, and high push buttons will be connected to the PLC as real inputs. The low (L) and high (H) motor starter contacts will also be brought in as inputs to provide low-voltage protection. The low- and high-speed starter coils will be connected as outputs. The rest of the circuit, including the timer, will be implemented using programming instructions.

The completed PLC program will look like Figure 4-17. The timer trap is implemented using the timer's enable output. Internal contact B3:0/0 is used to implement the control relay for the high-speed starter, which is driven by the high push button. The two rungs of this program perform the following functions:

- Rung one controls the low-speed motor starter.
- Rung two controls the high-speed motor starter and implements the 10.8-second delay through the use of a timer ON-delay instruction.

Rung 1. The first rung of the ladder program controls the low-speed starter coil and provides interlocking with the high-speed starter coil. When the low push button is pressed, the motor will start at low speed. At the same time, the timer will start timing and its done output will turn on after 10.8 seconds.

Rung 2. The second rung controls the high-speed starter coil. The fourth line of this rung turns the low-speed coil on when the high push button is pressed. It does this using an internal coil (B3:0/0) that bypasses the activation of the low push button contact I:0/0/1 in the first rung. Once the high push button has been pushed and the low-speed coil is on, the top three lines of the second rung enable the timer, which begins the 10.8-second

time delay. The timer is trapped using the T4:0/EN contact, which references the enable coil. When the timer times out, its done output will turn on. This will turn off the low-speed coil by breaking continuity to it. At the same time, the high-speed starter coil in line five of rung two will turn on because the done output will be on and the low-speed coil will be off. The high-speed push button in this rung is trapped on by contact B3:0/0 when the high push button is pushed. Thus, if the high-speed push button is pressed, the motor will start at low speed and then change to high speed after a 10.8-second delay.

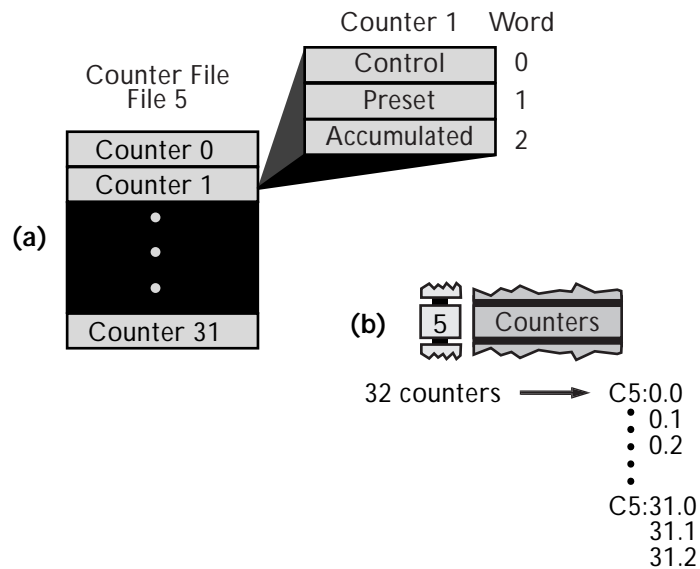


Figure 4-18. (a) The counter file and (b) its addressing scheme.

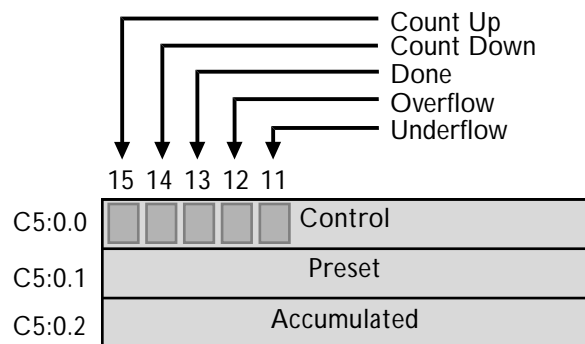


Figure 4-19. The data stored in each word of a counter's address.

4-2 Counting Instructions

Just as timing instructions replace the need for electromechanical timers in a PLC, counting instructions replace the need for electromechanical counters. This section discusses the different types of counting instructions available in a MicroLogix 1000.

At the end of this section, you will know:

- basic counter information
- the structure and operation of a MicroLogix's counting instructions
- special counter programming issues

General Counter Information

Counter Values. A counter instruction has two values associated with it:

- the preset value
- the accumulated value

These values perform the same function as they do in timer instructions. The preset value specifies the target number of counts, while the accumulated value indicates the actual number of counts that have already occurred. In a counter, the preset and accumulated values always increase or decrease in increments of one.

Addressing. Data about a MicroLogix 1000's counters is stored in file 5 of the data file section. The counter file can store the data of up to 32 counters, numbered 0 through 31 (see Figure 4-18). As with timers, each counter is allotted three words, which are numbered 0, 1, and 2. Each of these three words stores particular data about the counter instruction (see Figure 4-19):

- *Word 0* is the control word, which stores data about the counter block's operation and status. This word holds information about the status of the count up and count down outputs and data about the counter's done, overflow, and underflow status. This information is stored in bits 11 through 15 of the control word.
- *Word 1* stores the counter's preset value, which is the target count value.
- *Word 2* stores the counter's accumulated value, which is the actual count value. A counter's preset and accumulated words, words 1 and 2, are addressed with the labels PRE and ACC in the RSLogix software.

Counting instructions allow the implementation of several types of counter functions in a programmable controller. The three counting instructions found in a MicroLogix 1000 are:

- the count up instruction
- the count down instruction
- the reset instruction

Count Up Instruction

A **count up instruction** is represented by the symbol shown in Figure 4-20. The function of a count up instruction is to increase its accumulated value by one every time the block's input makes an OFF-to-ON transition. After a certain number of OFF-to-ON transitions have occurred, the count up instruction will energize its output. A count up block has two output coils:

- a count up output coil (CU), which indicates that the counter block is energized
- a done output coil (DN), which indicates that the count is complete

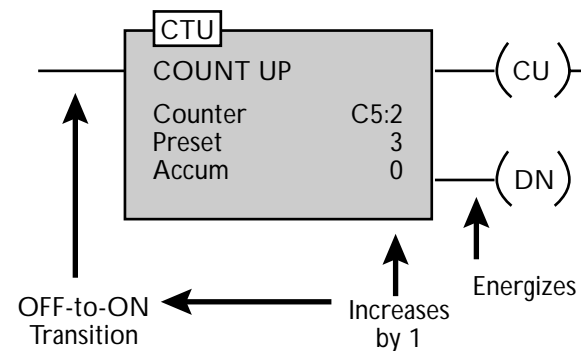


Figure 4-20. A count up instruction.

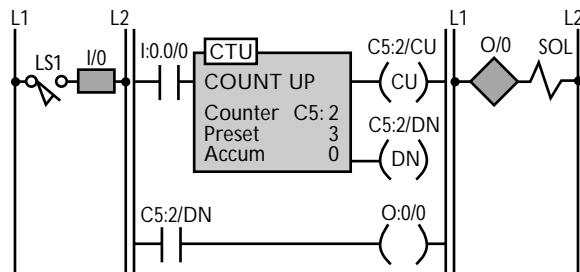


Figure 4-21. A count up circuit in which a limit switch and a solenoid are connected to a MicroLogix 1000.

Figure 4-21 shows a count up circuit in which a limit switch and a solenoid are connected to a MicroLogix 1000 controller. The solenoid should turn on after the limit switch has turned on three times. The circuit operates as follows:

- When the limit switch turns on for the first time, the count up output will be energized, and the accumulated value will increase to 1.
- When the limit switch turns off then on again, the accumulated value will increase to 2.
- When the switch makes its third OFF-to-ON transition, the accumulated value will increase to 3 and the done output will turn on because the accumulated value is equal to the preset value.
- When the done output turns on, the solenoid output in the second rung will be energized.

In a counter circuit, the counter will continue to count even after the accumulated value has reached the preset value. The done output will remain on as long as the accumulated count is greater than or equal to the preset count. The only way to reset the accumulated value and turn off the done output is to use a reset instruction, which will be discussed later in this section.

Count Down Instruction

A **count down instruction** (see Figure 4-22) decreases its accumulated value by one every time the block's input makes an OFF-to-ON transition. When the accumulated value becomes less than the preset value, the count down instruction de-energizes its output. When the counter's accumulated value is greater than or equal to its preset value, the counter's output will be on.

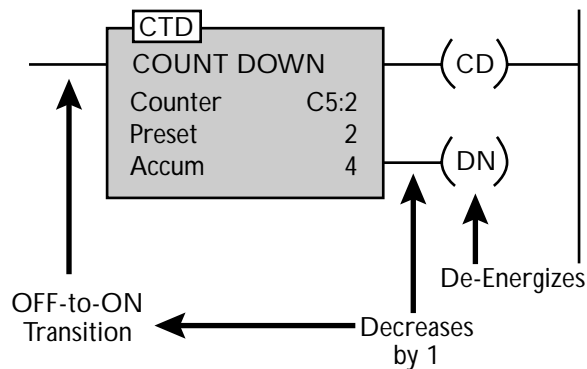


Figure 4-22. A count down instruction.

Like a count up instruction, a count down instruction also has two outputs:

- a count down output, which indicates that the counter is energized
- a done output, which signals that the target count value has been reached

Figure 4-23 shows a count down circuit, which works as follows:

- In this circuit, the count down block's done output will already be on because the accumulated value is greater than the preset value.
- When the block's input turns from OFF to ON, the accumulated value will decrease to 3.
- When the block's input makes this OFF-to-ON transition again, the accumulated value will decrease to 2.
- When the input makes one more OFF-to-ON transition, the accumulated value will drop to less than the preset value and the done output will turn off, de-energizing the done output and output O:0/0.

In practice, a count down instruction is most often used with a count up instruction to form an up/down counter. In the up/down counter shown in Figure 4-24, both counters share the same address and the same preset and accumulated values. As a result, the up counter increases the accumulated value every time a certain event occurs, while the down counter decreases the same accumulated value if another event occurs.

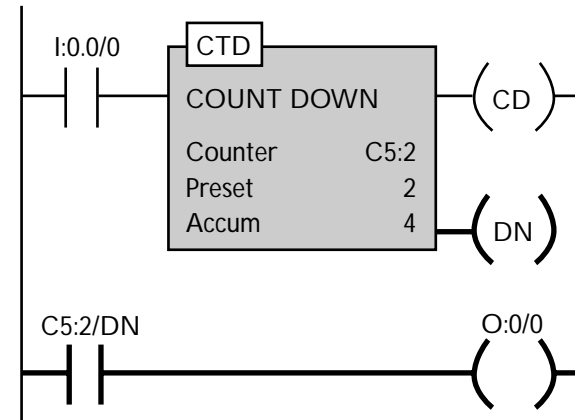


Figure 4-23. A ladder program containing a count down circuit.

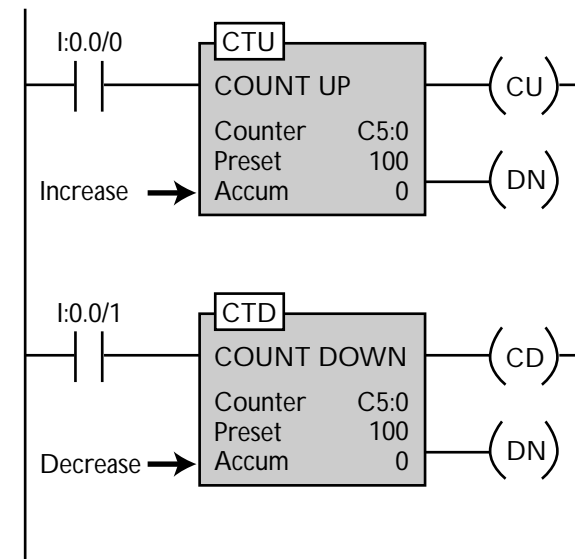


Figure 4-24. Up/down counter configuration.

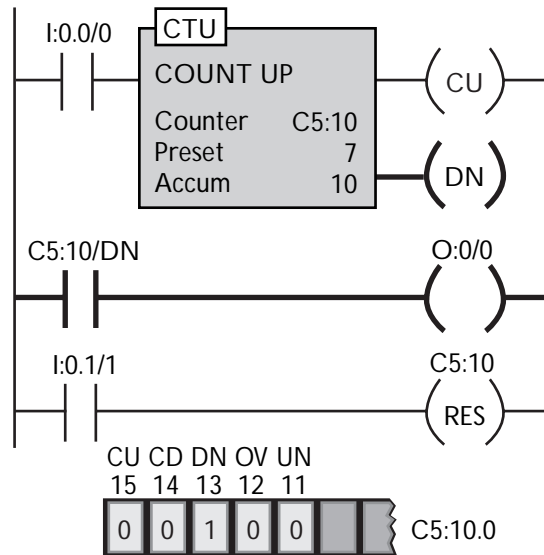


Figure 4-25. A reset instruction being used to reset a count up instruction.

Reset Instruction

A **reset instruction** is a coil instruction that can reset either a timing or counting instruction. When a reset instruction is energized, it sets the accumulated value of its corresponding timer or counter to 0. It also resets all of the control bits in word 0 of the timer or counter's memory location.

The ladder program shown in Figure 4-25 illustrates a reset instruction being used to reset a count up instruction. The reset coil shares the count up instruction's address—C5:10. The count up instruction has already counted up to 10, which is several counts past its preset value. Consequently, the counter's done output is on. When the reset coil's input is energized, the reset instruction will set the up counter's accumulated value to 0. At the same time, it will reset all of the bits in the counter's control word. This will turn the done output off.

A reset instruction can be used with all types of timing and counting instructions except a timer OFF-delay instruction. It cannot be used with a timer OFF-delay instruction because a reset instruction resets the done, timer timing, and enable bits of the timer's control word. If the status of these bits is altered while a timer OFF-delay instruction is timing, a machine malfunction could occur.

Special Programming Issues

When using counter instructions in a MicroLogix PLC, you must consider some special programming issues:

- using a reset instruction to implement a self-resetting counter
- counting past the maximum count
- reading fast input signals

Self-Resetting Counter. A **self-resetting counter** is a counter that resets itself in the same scan after the accumulated value reaches the preset value. Often a reset instruction is used in a counter circuit to implement a self-resetting action. However, this should be avoided in a MicroLogix 1000 unless certain precautions are taken, because the result will be an incorrect count value. Following is an explanation of why.

Figure 4-26 shows a reset instruction used to implement a self-resetting counter. When the counter's input turns on, the accumulated count value will increase to 1. At the same time, the counter's count up bit, bit 15, will turn on because its action follows that of the counter's input. Since the count up bit reflects the status of the input signal, the PLC uses it to determine if the input signal has made an OFF-to-ON transition. It does this by comparing the current status of the input signal to the value stored in the count up bit address.

Figure 4-27 shows the self-resetting counter circuit after several subsequent scans. If the input remains on in the scan following the first OFF-to-ON transition (point A), the MicroLogix will compare this 1 value to the value stored in count up bit 15 in scan 1. Since the count up value is already a 1, the PLC detects that the input has not made an OFF-to-ON transition. The controller will continue to make this same comparison every scan (points B and C). Therefore, when the input signal makes an off-to-on transition (point D), the MicroLogix will know it because the PLC will detect that the current status of the input is 1 and that the previous status of the count up bit was 0. Since the PLC senses an OFF-to-ON transition, it will increase its accumulated count value by one. In this circuit, the done bit will turn on since the accumulated value now equals the preset value.

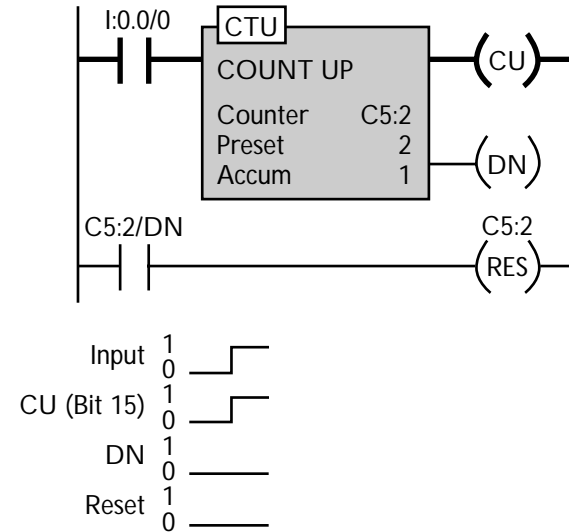


Figure 4-26. A reset instruction used to implement a self-resetting counter.

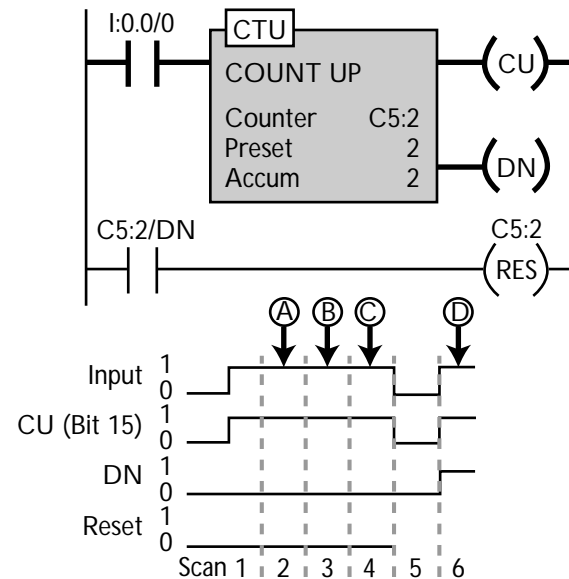


Figure 4-27. The self-resetting counter circuit after several subsequent scans.

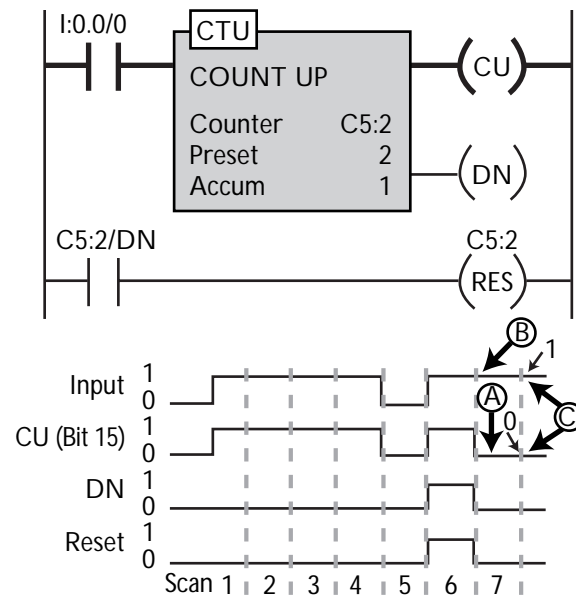


Figure 4-28. An illustration of what will happen after the count up instruction's accumulated value is reset.

Figure 4-28 shows what will happen after the counter's done bit turns on. When the done output turns on, the reset bit will also turn on since the done bit provides the input logic to the reset coil. The reset instruction will reset the accumulated value, as well as the count up and done bits, to 0 at the end of the scan. The reset instruction sets the count up bit to 0 (point A), but the input signal has not turned off (point B). This means that in the next scan the PLC will sense an OFF-to-ON transition as it compares the input signal to the count up value (point C), even though no transition has occurred. As a result, the PLC will increase the counter's accumulated value, despite the fact that no actual input transition has occurred.

Thus, using a reset instruction to implement a self-resetting counter will result in an inaccurate accumulated count value. To avoid this situation, you can use one of the following programming methods to create a self-resetting counter:

- Use a clear instruction instead of a reset instruction to set the counter's accumulated value to 0.
- Use a move instruction to move a value of 0 into the accumulated word at the end of the scan.
- Use a reset instruction, but with a one-shot rising instruction programmed at the input to the counter. This one-shot instruction will ensure that the input must turn off and then on again before the PLC will increment its count value.

Job Aid 4-1 provides examples of each of these self-resetting counter programming methods.

Counting Past The Maximum Count Value. A counter instruction's accumulated value has a range from $-32,768$ to $+32,767$. Once a counter reaches a count of $+32,767$, it cannot

go any higher. Therefore, it wraps the accumulated count back around to $-32,768$ and starts counting up again. To count past the $+32,767$ count value, you must cascade two counters, making sure that they self-reset in each scan.

When two counters are cascaded, they are programmed so that one counter provides the input to the other counter (see Figure 4-29). This way, the second counter counts how many times the first one has reached its preset value. Figure 4-30 shows two cascaded counters that implement a count to 100,000. These cascaded counters have addresses C5:10 and C5:11, and their programming works as follows:

- The input to the first counter is the event to be counted, while the input to the second counter is a contact that references the first counter's done bit.
- The first counter will increase its count every time the input event occurs. The second counter will increase its count every time the first counter's done output turns on—that is, every time the first counter's accumulated value equals its preset value.
- If the first counter's preset value is set to 1000 and the second counter's preset value is set to 100, they will implement a count to 100,000.
- Internal output B3:0/0 indicates when the count has reached 100,000 because this internal turns on when the second counter's done output turns on.
- The clear instruction resets the contents of the first counter's accumulated word to 0 every time its done bit is enabled, so that the first counter will reset to 0 every time it reaches a count of 1000.

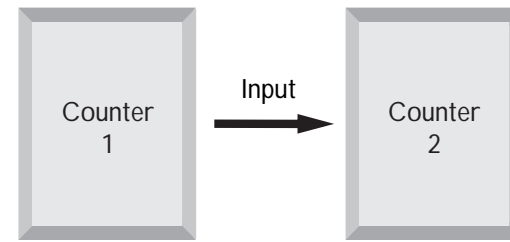


Figure 4-29. Cascaded counter, where counter 1 provides the input to counter 2.

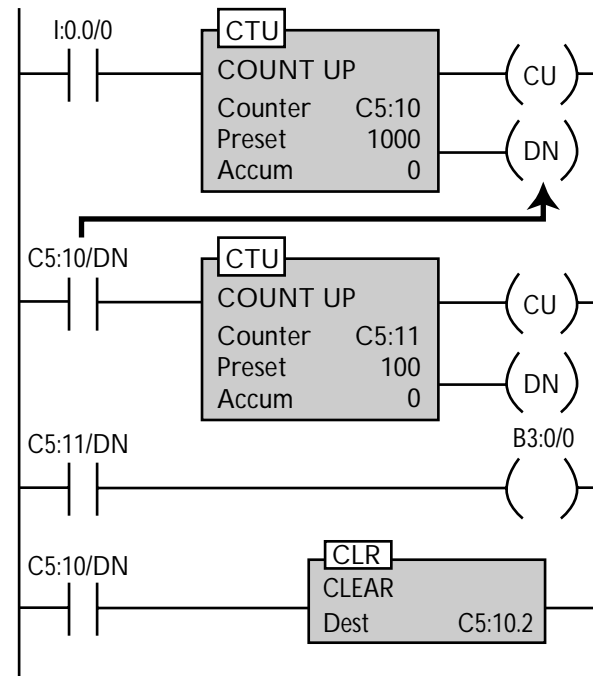


Figure 4-30. Two cascaded counters that implement a count to 100,000.

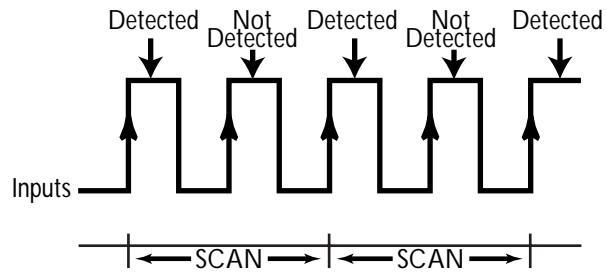


Figure 4-31. If the input events to be counted are happening at a rate faster than the scan, some of the inputs will not be counted.

Job Aid 2-2 provides more information about cascading counters to count past the maximum count.

Reading Fast Input Signals. If the input events to be counted are happening at a rate faster than the scan, some of the inputs will not be counted (see Figure 4-31). This is because a PLC only detects inputs that are valid at the beginning of each scan. It will not detect inputs that occur during the scan. If an application requires the counting of fast inputs, you must use a high-speed counter instruction to count them. This instruction is designed to count fast input signal pulses at a frequency of up to 6.6 kilohertz.

4-3 Data Handling Instructions

This section discusses data-handling instructions. Data-handling instructions are used to convert and move data within a Micro-Logix PLC. Data-handling instructions are often used to interface with field devices that supply or require data in BCD (binary coded decimal) form.

At the end of this section, you will know:

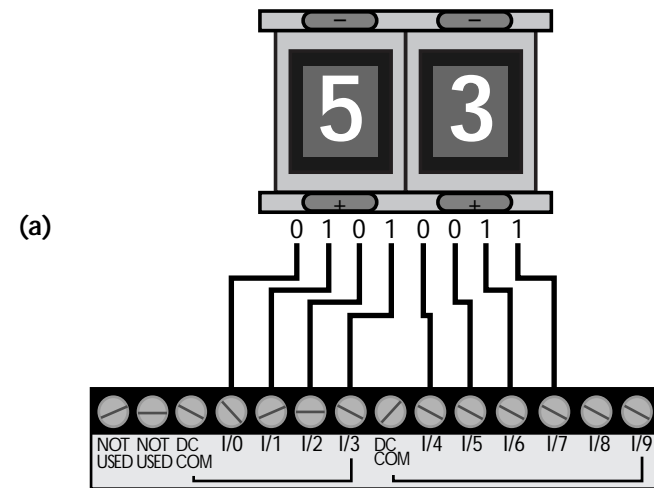
- BCD data-handling information
- how to use a convert-from-BCD instruction
- how to use a convert-to-BCD instruction
- how a move instruction operates
- how a masked move instruction operates
- how to apply ladder logic filtering to a BCD application

BCD Data-Handling Information

Before you can understand how BCD data-handling instructions work, you must first understand two fundamental BCD topics:

- how BCD input data is sent from an input field device to a PLC
- how BCD output data is sent from a PLC to an output field device

Reading BCD Input Data. A BCD input device communicates a decimal value to a PLC in binary coded decimal form. To communicate this data, the device uses a 4-bit code containing 1s and 0s (see Figure 4-32). To send this code to the controller, the device requires 4 input connections to the PLC's input inter-



(b)

Decimal	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Figure 4-32. (a) Two BCD thumbwheel switches communicating decimal values to a PLC in binary coded decimal form and (b) a decimal-to-BCD conversion table.

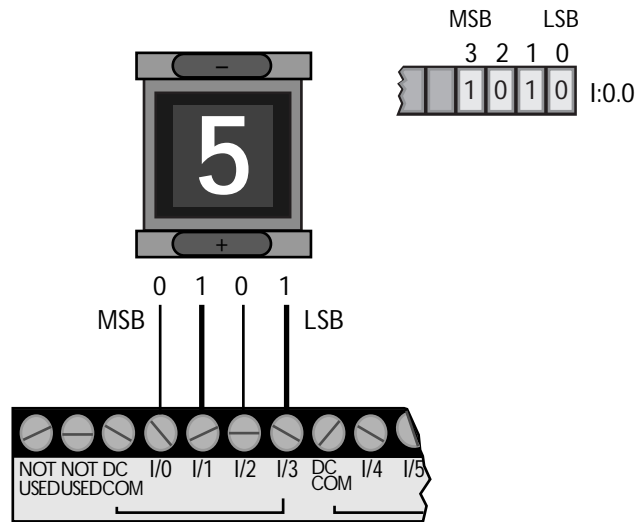


Figure 4-33. A thumbwheel switch connected to the first four input terminals of a MicroLogix PLC.

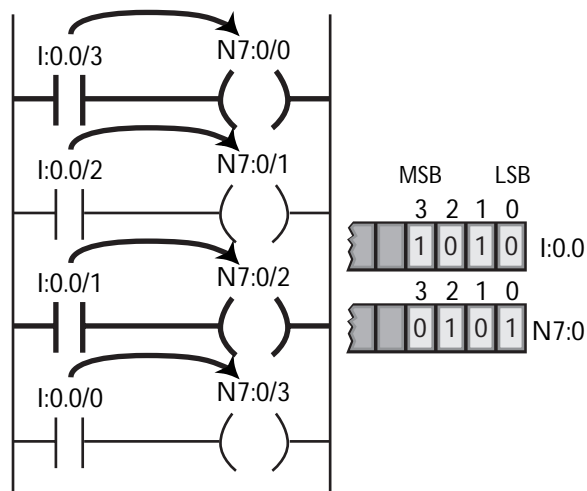


Figure 4-34. Ladder program used to store the BCD data in the integer file in the correct order of significance.

face—one connection for each BCD code bit. If more than one input device is used, then each device requires its own four separate input connections.

Figure 4-33 shows a thumbwheel switch connected to the first four input terminals of a MicroLogix PLC. This thumbwheel switch will transmit the BCD-equivalent value of the number 5 to the controller by providing a voltage to terminals 1 and 3, but not to terminals 0 and 2. Note that the PLC will interpret this number as 1010 instead of 0101, which is the actual BCD equivalent of the decimal number 5. This occurs because of the way the switch is wired. The BCD data will be stored in the input file in reverse order, with the most significant bit of the BCD value in the least significant input bit position and vice versa.

After the PLC has received the thumbwheel switch's BCD data, the data must be stored in the integer file in the correct order of significance. The ladder program shown in Figure 4-34 performs this task. Each rung of this program contains an examine-if-closed instruction that references one of the inputs. Each rung also contains an internal coil that references a bit in the integer file, which is where the input data will be transferred. This program operates as follows:

- The thumbwheel's inputs are programmed to transfer their data to the appropriate bit of the integer word to maintain the proper bit significance (i.e., input 3 to bit 0, input 2 to bit 1, input 1 to bit 2, and input 0 to bit 3).
- When the MicroLogix receives the BCD code equivalent to the number 5, the contacts referencing inputs 1 and 3 will energize. As a result, the internal outputs corresponding to integer word bits 0 and 2 will be on.
- At the same time, inputs 0 and 2, corresponding to internal outputs 1 and 3, will be off.

- Consequently, bits 0 through 3 of the integer file will store the value 0101, which is the BCD equivalent of the number 5.

Writing BCD Output Data. A BCD output device works the opposite of a BCD input. Figure 4-35 shows a seven-segment indicator connected to the output interface of a MicroLogix 1000. Like an input, a BCD output device requires four output connections to receive a binary coded decimal value from a PLC. The indicator is wired so that its least significant bit is wired to terminal 5 and its most significant bit is wired to terminal 2.

Figure 4-36 shows the ladder program used to transfer the BCD data from the integer file to the output device in the correct order. This program uses internal contacts to reference the BCD data stored in the integer file word. The rungs in this program energize based on the status of their reference bits. This transfers the BCD data to the output file via the output coils. The ladder program reverses the data from the integer file so that it is in the appropriate order and place in the output file.

BCD I/O Utilization. Using BCD input and output devices can tie up a MicroLogix 1000's I/O interfaces. If an application requires five thumbwheel switches and three seven-segment indicators, 20 input terminals and 12 output terminals must be used for just the BCD devices alone. This leaves no room for other inputs or outputs.

As an alternative to BCD I/O devices, you can use a MicroView operator interface instead. This MicroView interface inputs BCD data directly to the controller, replacing the need for thumbwheel switches, seven-segment indicators, and other similar BCD devices. This interface connects directly to the MicroLogix's RS-232 communication channel, meaning that it does not utilize any of the I/O terminals. The MicroView interface comes in both a handheld and a panel-mounted model.

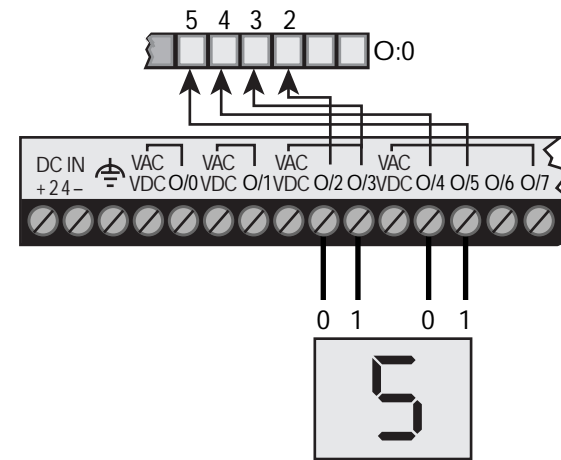


Figure 4-35. A seven-segment indicator connected to a MicroLogix's output interface.

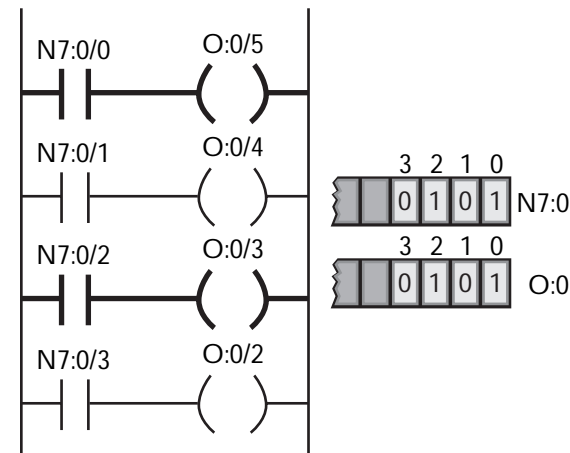


Figure 4-36. Ladder program used to transfer the BCD data from the integer file to the output device in the correct order.

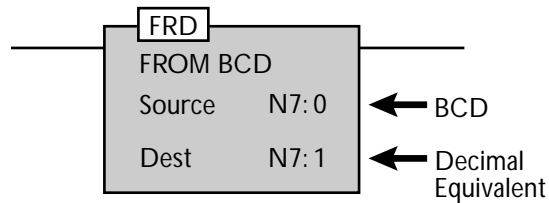


Figure 4-37. A convert-from-BCD instruction.

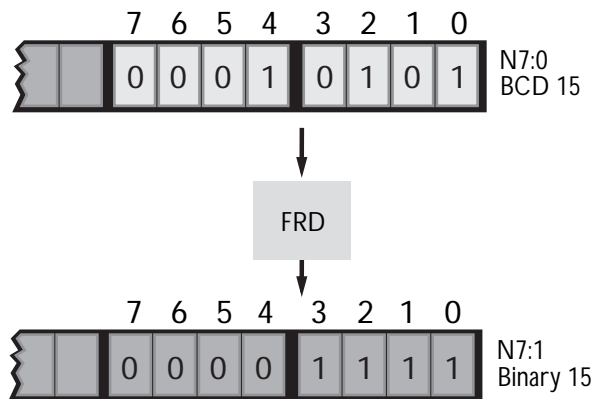


Figure 4-38. A convert-from-BCD instruction used to convert the BCD number 15 into the binary equivalent of the decimal number 15.

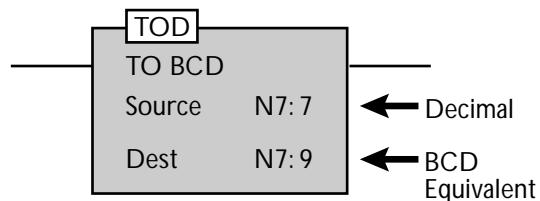


Figure 4-39. A convert-to-BCD instruction.

Convert-From-BCD Instruction

A **convert-from-BCD instruction** is a block instruction that converts the BCD data stored in a MicroLogix's data file into its equivalent decimal value (see Figure 4-37). This instruction block, which is abbreviated by the letters FRD, contains two pieces of information:

- a source location
- a destination location

The source location indicates where the BCD data to be converted is located. The destination location indicates where the decimal-equivalent value should be stored.

Figure 4-38 shows the first eight bits of integer file word 0, which contains the BCD-equivalent of the decimal number 15. This data is represented as two sets of 4-bit codes, with one set—bits 4 through 7—being the BCD equivalent of the number 1 (0001) and the other set—bits 0 through 3—being the BCD equivalent of the number 5 (0101).

Although the data in word 0 is supposed to represent the decimal number 15, the MicroLogix does not interpret it that way. Because a PLC is a straight binary machine, it interprets the data in word 0 as the binary number 00010101, which is actually the decimal number 21. As a result, you must use a convert-from-BCD instruction to convert the BCD value 15 (00010101) into the binary equivalent of the value 15 (00001111) and store this converted value in a new word location (N7:1).

Convert-To-BCD Instruction

A **convert-to-BCD instruction** looks like a convert-from-BCD instruction, but it is abbreviated TOD (see Figure 4-39). A convert-to-BCD instruction converts data stored in decimal form into its

equivalent BCD value. This instruction's source word contains the decimal value to be converted, while its destination word indicates where the converted BCD-equivalent value should be stored.

A convert-to-BCD instruction performs the opposite function of a convert-from-BCD instruction. Figure 4-40 shows a convert-to-BCD instruction that takes the binary-equivalent decimal value 15 (00001111) and converts it into the BCD-equivalent form of the number 15 (00010101). It then stores this BCD number in a new word. Thus, the convert-to-BCD instruction converts the decimal data into its BCD-equivalent value, which can then be sent to a BCD output device.

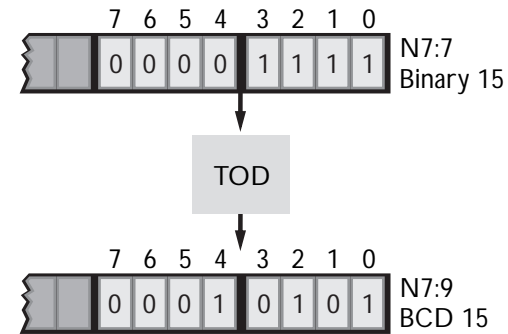


Figure 4-40. A convert-to-BCD instruction used to convert the binary-equivalent decimal value 15 (00001111) into the BCD equivalent form of the number 15 (00010101).

Move Instruction

Like BCD instructions, a **move instruction** also comes in a block format with both a source and a destination parameter (see Figure 4-41). A move instruction, however, simply moves data from the source word and puts it into the destination word. It does not convert or manipulate the data in any way. The data in a move block's source parameter can be either a variable value stored in a word, which changes during program execution, or a fixed constant value, which is entered during programming.

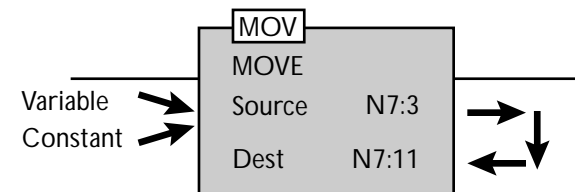


Figure 4-41. A move instruction.

Masked Move Instruction

A **masked move instruction** is used to manipulate data as it is moved (see Figure 4-42). A masked move instruction operates like a regular move instruction, except that a masked move lets you filter out data that you do not want to move. The mask parameter specified in the instruction block is what performs this filtering process. This mask parameter can be either a word address location or a hexadecimal constant.

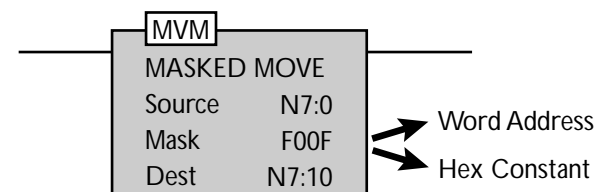


Figure 4-42. A masked move instruction.

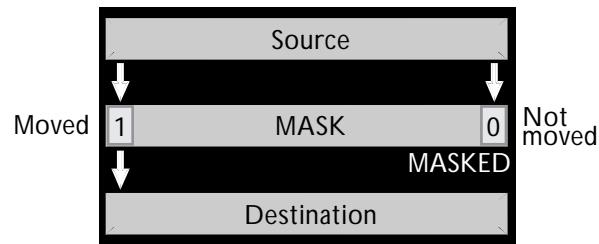


Figure 4-43. Mask parameter of a masked move instruction.

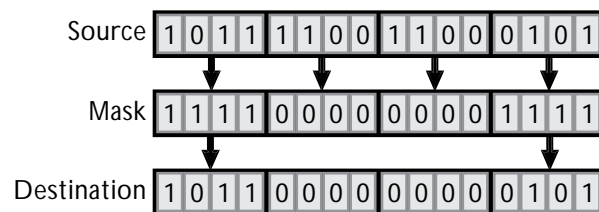


Figure 4-44. A masked move instruction's source, mask, and destination values.

The mask parameter of a masked move instruction specifies which bits in the source word are to be moved to the destination word and which bits in the source word are to be masked—that is, not moved (see Figure 4-43). If a bit in the mask word contains a 1, then the corresponding bit in the source word will be moved to the destination word. Conversely, if a bit in the mask contains a 0, then the corresponding bit in the source word will not be moved to the destination word.

As an example, Figure 4-44 shows a masked move instruction that moves only the first four and last four bits of data in the source word to the destination word. The instruction operates like this:

- The first four and last four bits of the mask word are set to 1, while the other bits are left as 0.
- When the masked move instruction is energized, only the first four and last four bits of data will be moved to the destination word.
- The middle bits in the destination word will not be replaced. They will remain in their previous state, which in this case was all zeros.

The bits in the mask register are set to either 1 or 0 in groups of four using the hexadecimal equivalent of each group's binary pattern. Thus, you would use the letter *F* to indicate that a group of four bits is set to one. You would use a 0 to indicate that a group of four bits is set to zero. This notation is what you see as the mask value in the masked move instruction. Job Aid 4-3 provides more information on using mask codes to filter data and input BCD information.

Ladder Logic Filtering

Ladder logic filtering prevents BCD conversion errors due to the difference in the BCD device and PLC processing speeds. Figure 4-45 shows a thumbwheel switch that is sending the BCD number 7 to a MicroLogix 1000. The PLC sees this number 7 as the BCD value 0111. Note that the thumbwheel switch is wired so that its least significant bit corresponds to the least significant input word bit and its most significant bit corresponds to the most significant input word bit.

If the thumbwheel switch's value changes from 7 to the number 8, which has the BCD binary pattern of 1000, the device will send the new number to the PLC. Because of its mechanical nature, however, a BCD device operates slowly as compared to a MicroLogix 1000. Thus, the BCD device may not be able to send all of its new data to the PLC within the period of one scan. In fact, it may take the device several scans to provide the BCD pattern for the selected number. During this time, the output of the BCD device may specify an invalid BCD bit pattern.

If the switch sends its terminal 3 data to the input terminals first, before it sends any of its other new data (see Figure 4-46), and then the PLC performs a scan, the controller will read the BCD number 1111. This is not a valid BCD value. Thus, if the MicroLogix tries to perform a BCD conversion on this data once it has been input to the controller, an overflow will occur in bit 1 of status file word 0 (the math overflow bit). This overflow will cause an error fault at the end of the scan, halting the PLC's operation. As a result of this invalid BCD number, the PLC will store a +32,767 in the destination word of the convert-from-BCD instruction.

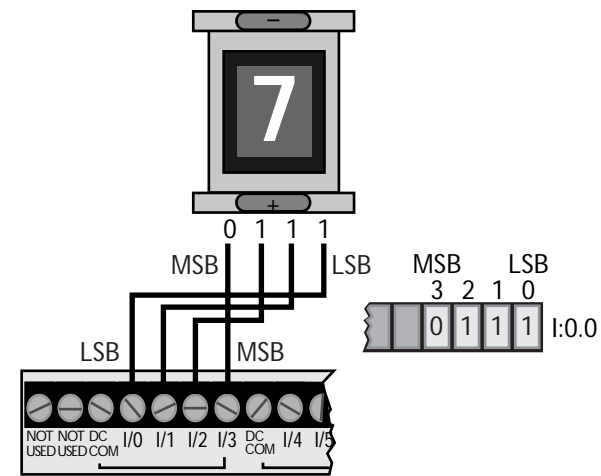


Figure 4-45. A thumbwheel switch sending the number 7 to a MicroLogix.

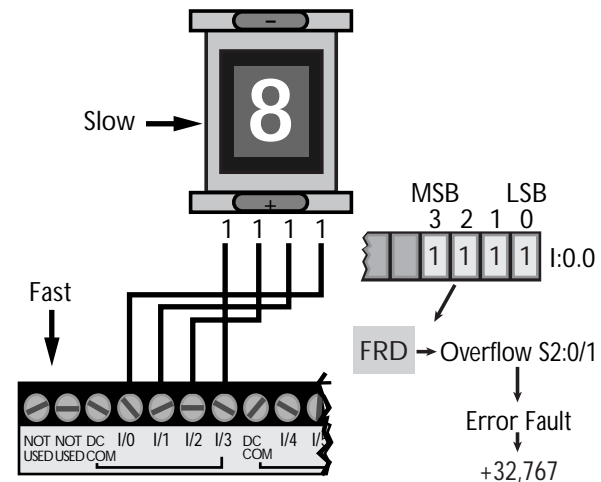


Figure 4-46. The thumbwheel switch's value changing from 7 to the number 8.

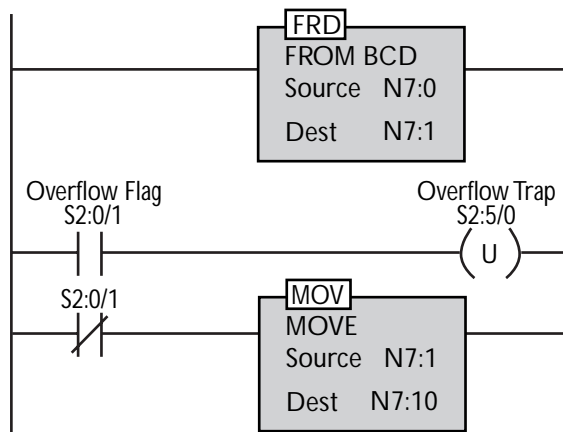


Figure 4-47. A circuit with ladder logic filtering.

To avoid this situation, you must apply ladder logic filtering to the circuit. Ladder logic filtering ensures that the data received from the BCD device is a valid BCD code. This way, the PLC will wait until it has received all of the updated BCD input data before it performs a BCD-to-binary conversion.

Figure 4-47 shows a circuit with ladder logic filtering. It contains three rungs, which perform the following functions:

- The first rung contains a convert-from-BCD block that converts the BCD data in word N7:0 to its decimal binary equivalent and stores it in word N7:1.
- The second rung contains an examine-if-closed contact that references the MicroLogix's status overflow bit. If this contact is on, the convert-from-BCD instruction in rung one has read an invalid BCD code in its source register. This contact drives an unlatch coil that resets the overflow condition so that the MicroLogix will not fault at the end of the scan. This will allow the PLC to keep reading the input data.
- The third rung contains an examine-if-open instruction that also references the overflow bit. If the convert-from-BCD instruction has received a valid BCD code, then the overflow bit will not be on. Accordingly, this rung will energize, moving the converted BCD value in word N7:1 to word N7:10.

When this type of ladder logic programming is used, the destination word of the move instruction (in the previous case, word N7:10) will hold the converted valid BCD value. This destination word should be used when referencing the converted BCD number in the ladder program.

4-4 Review

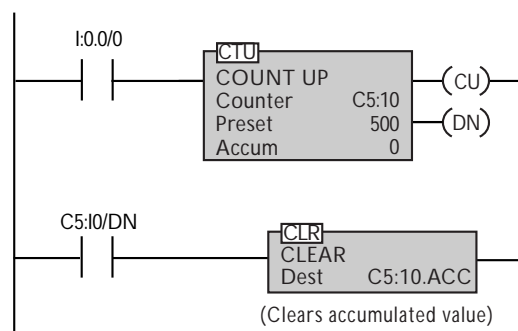
- A timer ON-delay instruction energizes its output after its input turns on and a certain amount of time has elapsed.
- A timer OFF-delay instruction de-energizes its output after its input turns off and a certain amount of time has elapsed.
- A retentive timer instruction works like a timer ON-delay instruction, except that its accumulated value is retained even if the timer's input turns off.
- A trapping circuit is used to implement the instantaneous timer contact in a PLC program.
- A count up instruction increases its accumulated value by one every time its input makes an OFF-to-ON transition.
- A count down instruction decreases its accumulated value by one every time its input makes an OFF-to-ON transition.
- A count down instruction is usually used with a count up instruction to form an up/down counter.
- A reset instruction is used to reset the accumulated value and control bits of counter instructions, as well as timer ON-delay and retentive timer instructions.
- A reset instruction cannot be used alone (unless some precautions are taken) to create a self-resetting counter circuit because an incorrect count value will result.
- A cascaded counter circuit must be created for a counter to count past its maximum count value.
- A high-speed counter instruction must be used to count fast input signals.
- BCD input devices require four input terminal connections and four bits of memory, to send their BCD data to the PLC.
- BCD output devices require four output terminal connections and four bits of memory, to receive BCD data from a PLC.
- A convert-from-BCD instruction takes the BCD value stored in the source word and stores it in the destination word in binary equivalent form.
- A convert-to-BCD instruction takes the binary data stored in the source word and stores it in the destination word in BCD-equivalent form.
- A move instruction moves data from one word location to another without manipulating it.
- A masked move instruction moves only the source word data bits indicated by ones in the mask to the destination word.
- Ladder logic filtering is a special type of circuit that prevents the PLC control program from halting operation due to an overflow fault resulting from an invalid BCD value.

4-5 Job Aids

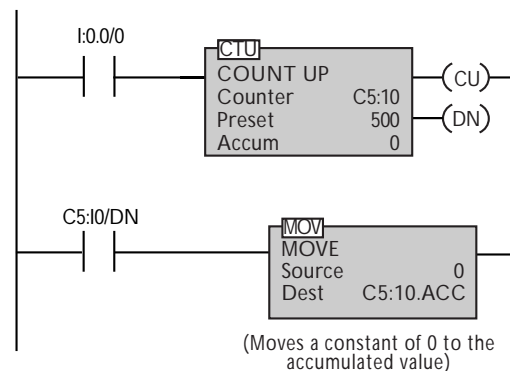
Job Aid 4-1: Self-Resetting Counter Programming Methods

Because a reset instruction cannot be used alone to implement a self-resetting counter circuit, you must use another programming method to implement this type of circuit. Following are three methods that can be used to create a self-resetting counter.

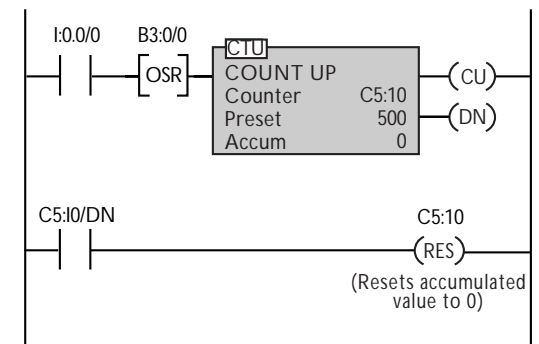
Method 1: Use a clear instruction to set the counter's accumulated count value to 0.



Method 2: Use a move instruction to move a source value of 0 into the counter's accumulated word at the end of the scan.

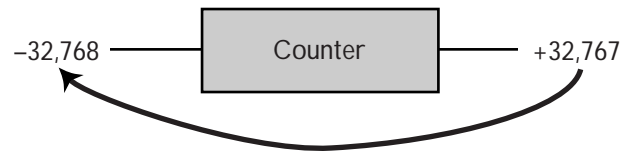


Method 3: Use a reset instruction in conjunction with a one-shot rising instruction programmed at the input to the counter.



Job Aid 4-2: Counting Past the Maximum Count

When a counter instruction reaches the end of its range of count values ($-32,768$ to $+32,767$), it wraps the accumulated count value back around and starts counting from the other side. The following graphic illustrates what occurs when a counter reaches its maximum count of $+32,767$:

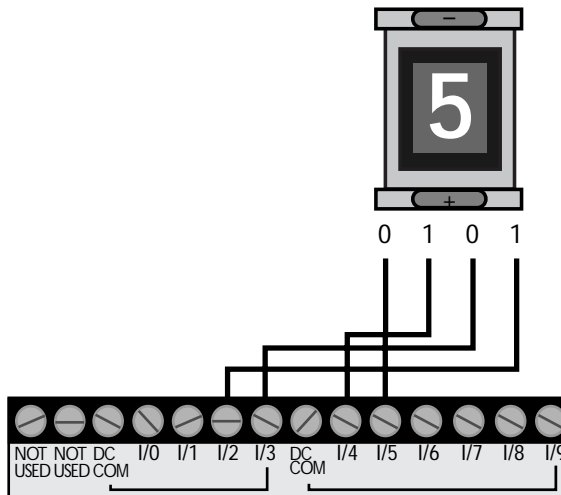


To count past the maximum count, you must cascade two counters in a ladder circuit. You cannot try to trick a counter by setting its preset value to the opposite end of the range because the done bit will not behave properly. For example, you could try to count to $+32,770$ by setting a counter's preset value to $-32,766$, since this value is three counts past the maximum limit once the counter wraps its accumulated value around. However, this will not work for the following reasons:

- Once the counter wraps back around to $-32,768$, its overflow bit will turn on.
- When the overflow bit turns on, the counter's done output will turn on, even though its accumulated value is less than its preset value.
- At this point, the done bit will stay on until the overflow bit is reset or the counter counts back down to $+32,767$.

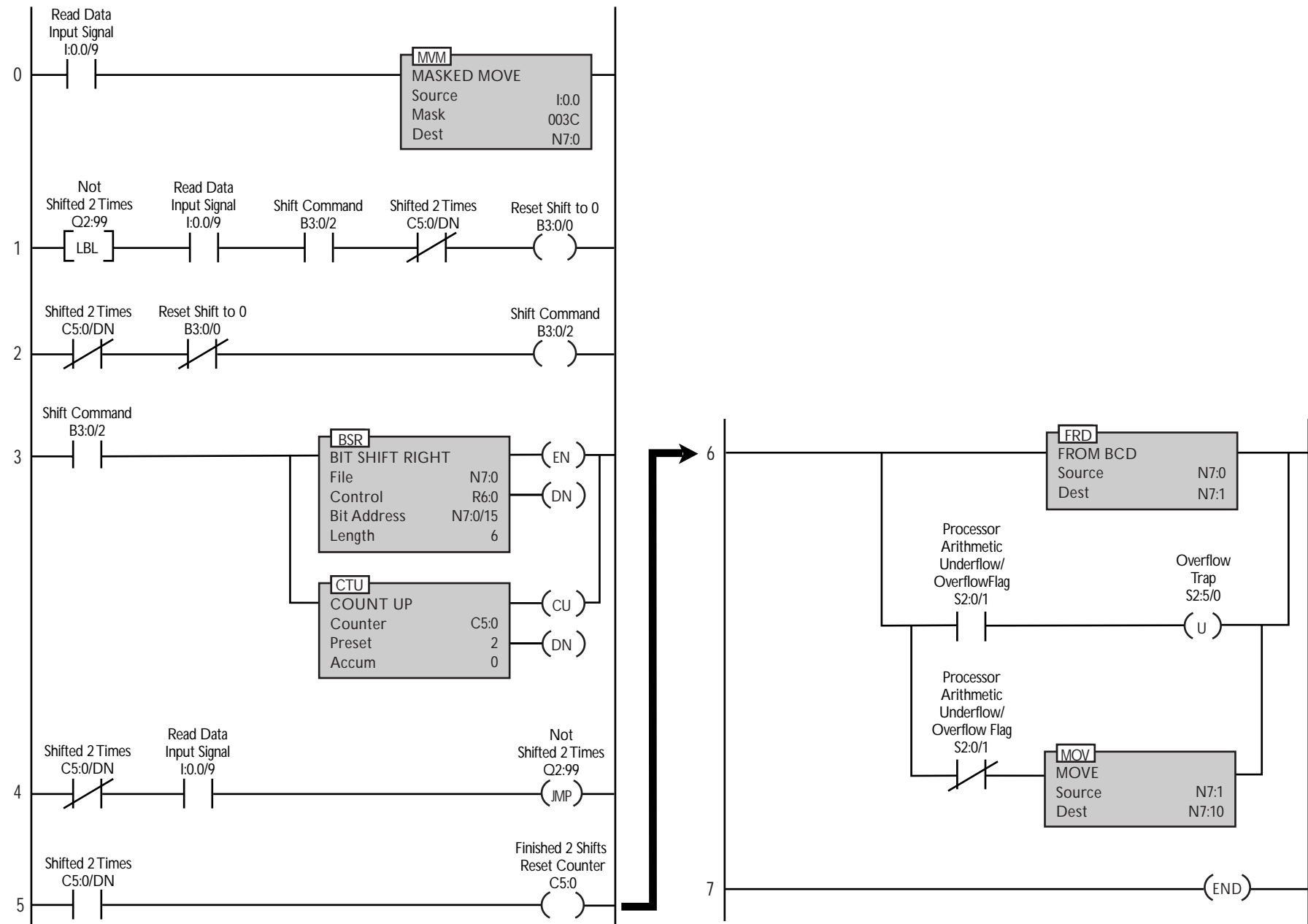
Job Aid 4-3: Using Mask Codes to Filter BCD Data

You can use a masked move instruction to input data to a PLC from BCD devices connected to “odd” input terminal bit locations. For example, the following diagram shows a thumbwheel switch that is connected to input terminals 2, 3, 4, and 5 of a 16 I/O MicroLogix 1000:



To filter and move this BCD data, you could use a masked move instruction to mask out all but the thumbwheel switch's input data and move it to an integer word location. You could then use a bit shift right instruction to shift the BCD data into the integer word's first four bits before performing a convert-from-BCD instruction.

The next page shows a ladder program that would implement this action in a MicroLogix PLC. This ladder diagram contains a masked move (MVM) instruction in rung 0 to read the thumbwheel switch's input data (inputs I/2 through I/5). The MVM instruction uses a hexadecimal mask value of 003C to move only the desired bits into word N7:0. The bits in this word must then be shifted two positions to the right for the word to contain the correct BCD number in the correct position. The shifting of the two bits is accomplished in rungs 1 through 5. Rung 6 implements a BCD-to-decimal conversion.



Following is a detailed explanation of the function of each of the ladder rungs:

- *Rung 0:* The XIC contact I:0.0/9 in this rung references an input event that will trigger the MVM instruction, which reads and transfers the thumbwheel switch's BCD input data to word N7:0. If the input data is to be read continuously, rather than conditionally, this XIC contact should be omitted. If this is the case, the I:0.0/9 contacts in rungs 1 and 4 should be omitted as well.
- *Rungs 1, 2, 4, and 5:* Rungs 1 and 2 implement an oscillating OFF-to-ON/ON-to-OFF input command for the bit shift right (BSR) instruction in rung 3 by referencing the output of the count up instruction, also located in rung 3. The counter keeps track of how many times the BSR instruction has been executed. These rungs work as follows:
 - If the BSR instruction has been executed less than two times, rung 4 will jump the program to rung 1, which resets rung 2 (if rung 2 is already energized). Subsequently, rung 2 will energize, meaning that the BSR instruction will be executed.
 - If the BSR instruction has already been executed two times, then rung 4 will not be energized, meaning that program execution will move to rung 5. Rung 5 resets the counter's accumulated value.
- *Rung 3:* The BSR instruction shifts the BCD data bits in the integer word. When the BSR block's input turns from OFF to ON, the block will shift the contents of word N7:0 one bit to the right. The block uses the following parameters to complete this operation:
 - The file parameter (N7:0) indicates where the data to be shifted is stored.
 - The control parameter (R6:0) stores control data about the BSR instruction (e.g., EN output, DN output, length, etc.).
 - The length parameter (6) specifies the file word bit into which data will be shifted—i.e., word N7:0, bit 5 (the sixth bit).
 - The bit address parameter (N7:0/15) specifies the location of the data to be shifted into the specified file word bit. This bit address will always contain a 0, so each time the BSR instruction is executed, the bits in word N7:0 will be shifted one bit to the right and a 0 will be stored in bit N7:0/5.The counter in this rung simply counts the number of times the BSR instruction has been executed. The counter's done output will be energized when the BSR instruction has been executed two times.
- *Rung 6:* This rung implements the BCD-to-decimal conversion of the data stored in word N7:0. Once the data has been shifted into the proper position, the convert-from-BCD block will convert the BCD number into its equivalent decimal value. This rung also contains ladder logic filtering to prevent an overflow fault due to an invalid BCD number. Thus, once the PLC determines that a valid BCD number has been converted, the move instruction will transfer the newly converted data to word N7:10, where it will be stored for use by the rest of the control program.

Note that this program could be programmed as a subroutine rather than as part of the main control program if you did not want to include it in the main program.

