AutoMax
Enhanced Ladder Language

Instruction Manual J2-3094-4

RELIANCE ELECTRIC

The information in this user's manual is subject to change without notice.

DANGER

ONLY QUALIFIED ELECTRICAL PERSONNEL FAMILIAR WITH THE CONSTRUCTION AND OPERATION OF THIS EQUIPMENT AND THE HAZARDS INVOLVED SHOULD INSTALL, ADJUST, OPERATE, OR SERVICE THIS EQUIPMENT. READ AND UNDERSTAND THIS MANUAL AND OTHER APPLICABLE MANUALS IN THEIR ENTIRETY BEFORE PROCEEDING. FAILURE TO OBSERVE THIS PRECAUTION COULD RESULT IN SEVERE BODILY INJURY OR LOSS OF LIFE.

WARNING

THE USER MUST PROVIDE AN EXTERNAL, HARDWIRED EMERGENCY STOP CIRCUIT OUTSIDE THE CONTROLLER CIRCUITRY. THIS CIRCUIT MUST DISABLE THE SYSTEM IN CASE OF IMPROPER OPERATION. UNCONTROLLED MACHINE OPERATION MAY RESULT IF THIS PROCEDURE IS NOT FOLLOWED. FAILURE TO OBSERVE THIS PRECAUTION COULD RESULT IN BODILY INJURY.

1.0 Relay Instructions

Use the relay instructions to simulate contacts (input instructions) and coils (output instructions). Choose from these input relay instructions:

- Normally Open Contact (NOI)
- Normally Closed Contact (NCI)
- Positive Transition Contact (PTI)
- Negative Transition Contact (NTI)
- Always True Contact (ATI)
- Always False Contact (AFI)

Choose from these output relay instructions:

- Coil (CO)
- Set (Latch) Coil (SCO)
- Reset (Unlatch) Coil (RCO)

The thick black bar shown at the right-hand margin of this page will be used throughout this instruction manual to signify new or revised text or figures.

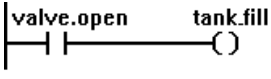
1.1 Normally Open Contact (NOI)

Use this input instruction to examine whether a Boolean variable is on (1) or off (0). When the variable is on, the instruction is true. Otherwise, the instruction is false.

The supported variables are:

- simple Boolean
- Boolean array element
- bit-indexed integer or double integer
- bit-indexed integer or double integer array element
- timer/counter status bits

Example of a Normally Open Contact (NOI)

This rung shows that when the variable *valve.open* is true, the variable *tank_fill* is set true.	

1.2 Normally Closed Contact (NCI)

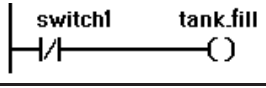
Use this input instruction to examine whether a Boolean variable is on (1) or off (0). When the variable is off, the instruction is true. Otherwise, the instruction is false.

The supported variables are:

- simple Boolean
- Boolean array element
- bit-indexed integer or double integer
- bit-indexed integer or double integer array element
- timer/counter status bits

Example of a Normally Closed Contact (NCI)

This rung shows that when *switch1* is false, the variable *tank_fill* is set true.



1.3 Positive Transition Contact (PTI)

Use this input instruction to examine a Boolean variable for a rising edge. When the variable changes from being off to on, the PTI instruction becomes true for one scan; otherwise, the instruction is false.

The supported variables are:

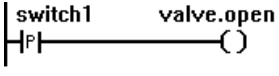
- simple Boolean
- Boolean array element
- timer/counter status bits

Do **not** use these variables:

- bit-indexed integer or double integer variables
- bit-indexed integer or double integer array elements

Example of a Positive Transition Contact (PTI)

This rung shows that when *switch1* transitions from off to on the variable *valve.open* is set true for one scan.



1.4 Negative Transition Contact (NTI)

Use this input instruction to examine a Boolean variable for a falling edge. When this variable changes from being on to off, the NTI instruction becomes true for one scan; otherwise, the instruction is false.

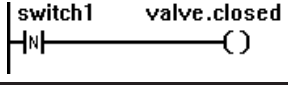
The supported variables are:

- simple Boolean
- Boolean array element
- timer/counter status bits

Do **not** use these variables:

- bit-indexed integer or double integer variables
- bit-indexed integer or double integer array elements

Example of a Negative Transition Contact (NTI)

This rung shows that when *switch1* transitions from on to off the variable *valve.closed* is set true for one scan.	

1.5 Using Transition Contacts

This section explains the methods for using transition contacts PTI and NTI and how the transitions will be interpreted. The methods explained in this section are the following:

- using a variable only on a transition contact
- using a variable on a coil and on a transition contact
- using a variable on more than one coil and on a transition contact
- forcing or setting variables used on transition contacts
- using a variable on a set (SCO) coil and reset coil (RCO) pair and on a transition contact
- using transition contacts in a program with a Jump and Label construct

1.5.1 Using a Variable Only on a Transition Contact

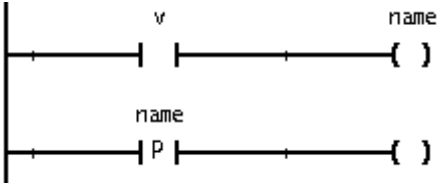
When you use a variable only on transition contacts and not on any other coil within a program, the transition contact behaves as follows:

- If the variable is local, setting or forcing the variable causes the transition contact to always evaluate true. Therefore always use a coil for the local variable somewhere else in the program.
- If the variable is global, another task determines whether a transition is detected by the transition contact, since only another task can change the variable's value.

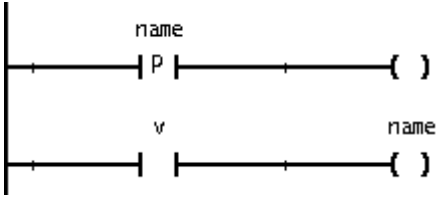
1.5.2 Using a Variable on a Coil and on a Transition Contact

The location of the coil in the program in relation to the transition contact using the same variable name helps determine when the transition is detected. The coil can appear before or after the transition contact.

If the transition contact is placed **after** the coil, any transition is detected on the **current** program scan.



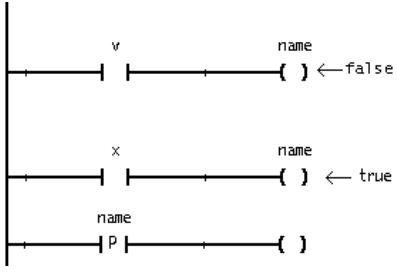
If the transition contact is placed **before** coil the, any transition is detected during the **next** scan.



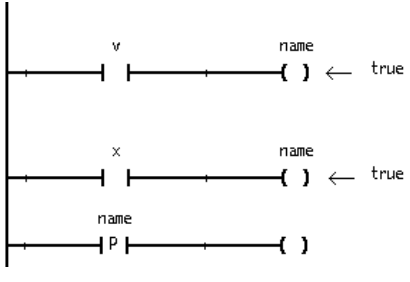
1.5.3 Using a Variable on More Than One Coil and On a Transition Contact

When you use the same variable on more than one coil and on a transition contact, the state of the variable for the contact is determined by the most recently executed coil.

For example, a transition would be detected in this case:



However, a transition would **not** be detected in this case, since the variable name is true for both coils:



1.5.4 Forcing or Setting Variables Used on Transition Contacts

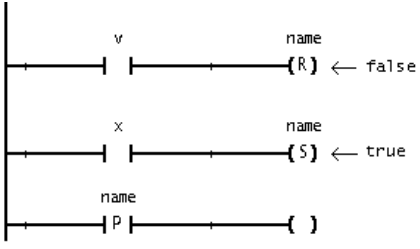
When setting or forcing a variable used on a transition contact, you must be aware of how the transition contact is affected. The transition contact is also affected if the variable being set or forced is used on a coil as well. This table summarizes the effect setting or forcing a variable has on transition contacts with and without a coil being in the program.

Variable Scope	Coil Present	Normal Execution	Variable Is Set	Variable Is Forced
local	no	no effect; nothing changing the variable	transition contact **continuously** true	transition is detected
	one coil before the transition contact	transition is detected	transition is **not** detected	transition is **not** detected
	one coil after the transition contact	transition is detected	transition is detected	transition is detected
global	no	transition is detected	transition is detected	transition is detected
	one coil before the transition contact	transition is detected	transition is **not** detected	transition is **not** detected
	one coil after the transition contact	transition is detected	transition is detected	transition is detected

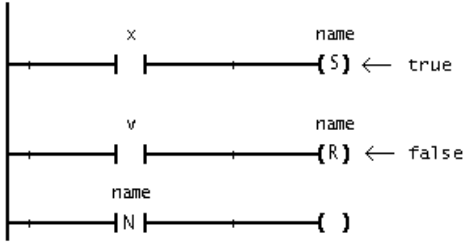
1.5.5 Using a Variable on a Set (SCO) Coil and Reset Coil (RCO) Pair and on a Transition Contact

When you use the same variable name on an RCO and SCO pair and on a transition contact, the state of the variable for the contact is determined by the most recently executed coil.

For example, a PTI instruction would see a transition in this case:



and, an NTI would see a transition in this case:



1.5.6 Using Transition Contacts in a Program with a Jump and Label Construct

When using Jump and Label constructs in the same program as transition contacts and coils using the same variable, keep in mind that the last executed coil determines the state of a transition contact. Also, by jumping over rungs some coils may not be executed.

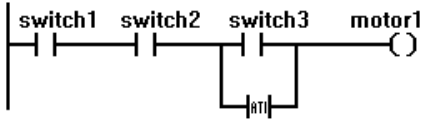
1.6 Always True Contact (ATI)

Use this instruction whenever you need a contact that will always evaluate true. For example, use it when you are debugging a program and wish to bypass some logic, but you do not want to delete the original logic.

Using a variable name is optional. However, if one is used, the variable must be a simple Boolean.

Example of an Always True Contact (ATI)

This rung shows that the state of *switch3* has no affect on the state of *motor1* because of the ATI instruction.

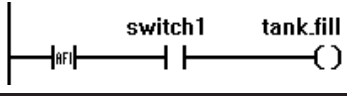


1.7 Always False Contact (AFI)

Use this input instruction whenever you need a contact that will always evaluate false. For example, use it when you are debugging a program and wish to disable some logic, but you do not want to delete the original logic.

Using a variable name is optional. However, if one is used, the variable must be a simple Boolean.

Example of an Always False Contact (AFI)

This rung is placed in a false state by the AFI instruction.	

1.8 Coil (CO)

Use this output instruction to store the state of the rung in the Boolean variable specified in this instruction.

When the rung is true, a value of 1 is stored in the Boolean variable specified for this coil instruction.

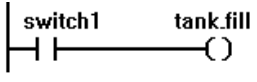
When the rung is false, a value of 0 is stored in the Boolean variable specified for this coil instruction.

The supported variables are:

- simple Boolean
- Boolean array element
- bit-indexed integer or double integer
- bit-indexed integer or double integer array element

Example of a Coil (CO)

This rung shows that the state of the variable *tank_fill* follows the state of the variable *switch1*.



1.9 Set (Latch) Coil (SCO)

Use this output instruction to set the Boolean variable on (1) when the input condition is true.

The SCO instruction is an instruction that can only turn on a bit (it cannot turn off a bit). This instruction is usually paired with an RCO (reset unlatch) instruction, with both instructions addressing the same bit.

When enabled, the latch instruction turns on the addressed bit. After this, the bit remains on (regardless of the rung condition) until the bit is turned off, typically by an RCO instruction in another rung.

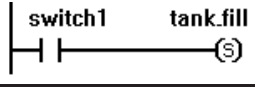
If the rung is:	Then the instruction turns the bit:
true	on
false	no change

The supported variables are:

- simple Boolean
- Boolean array element
- bit-indexed integer or double integer
- bit-indexed integer or double integer array element

Example of a Set (Latch) Coil (SCO)

This rung shows that when *switch*1 is on, the variable *tank_fill* is on until it is reset by an RCO instruction.



1.10 Reset (Unlatch) Coil (RCO)

Use this output instruction to reset a Boolean variable to off (0) when the input condition is true.

The RCO instruction is a retentive output instruction that can only turn off a bit (it cannot turn on a bit). This instruction is usually paired with an SCO (set latch) instruction, with both instructions addressing the same bit.

When enabled, the unlatch instruction turns off the addressed bit. After this, the bit remains off (regardless of the rung condition) until it is turned on, typically by an SCO instruction in another rung.

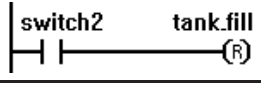
If the rung is:	Then the instruction turns the bit:
true	off
false	no change

The supported variables are:

- simple Boolean
- Boolean array element
- bit-indexed integer or double integer
- bit-indexed integer or double integer array

Example of a Reset (Unlatch) Coil (RCO)

This rung shows that when *switch2* is on the variable *tank_fill* will be off until it is set by an SCO instruction.



1.11 Errors Caused by the Relay Instructions

These errors can occur when you are using the relay instructions in a program. They are logged in the error log.

If this error occurs:	Then:	Do the following:
The bit number is negative.	Bit 0 of the variable will be used for the instruction's operation.	Specify a number of 0-15 for integers and 0-31 for double integers.
The bit number is too large.	The largest bit number of the variable will be used for the instruction's operation.	Specify a number of 0-15 for integers and 0-31 for double integers.
The array index is negative.	Element 0 of the array variable will be used for the instruction's operation.	Specify a valid array element.
The array index is too large.	The largest element number of the array variable will be used for the instruction's operation.	Specify a valid array element.

2.0 Counter Instruction

Use counter instruction (CTUD) to count activities as they occur, like products passing over a switch on a conveyor belt or pushes of a button. The counter instruction uses the counter data type to control the counter instruction.

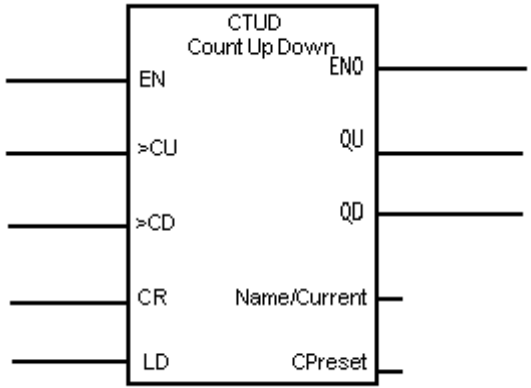
The value in Current can count up past the preset value and down past zero. However, the value in Current cannot exceed the upper limit or go below the lower limit of a double integer.

The QU and QD outputs do not change state unless the instruction is executed, even if the CPreset has been modified to be less than or equal to the current value.

IMPORTANT

Global counters must be entered into the variable configurator as five-element, double-integer, non-volatile arrays.
Example: COUNTER1!(4).

2.1 Count Up Down (CTUD)



Use this instruction to increment or decrement a counter.

When EN is true, the instruction:

- increments the double integer value stored in Current for every false-true transition of the CU input
- decrements the double integer value stored in Current for every false-true transition of the CD input

The QD output becomes true when the value stored in Current is less than or equal to zero. The QU output becomes true when the value stored in Current is greater than or equal to the value stored in CPreset.

2.1.1 Input Parameters for the Count Up Down Instruction

This table lists the inputs for the CTUD instruction and the variable type and data type/range that each input supports.

Parameter	Description	Variable Type	Data Type/Range
EN	When this input is true, the instruction is enabled. When EN is false, the instruction is not executed and ENO, QU, and QD are set false.	Connect a Boolean input or output.	
CU	When the instruction is enabled and CR and LD are false, a false-true transition of this input causes the value in Current to be incremented by one. This input is false when not connected.		
CD	When the instruction is enabled and CR and LD are false, a false-true transition of this input decrements the value in Current by one. This input is false when not connected.		
CR	When this input is true, the value in Current is reset to zero. This input is false when it is not connected. See “Resetting a Counter,” section 2.2.		
LD	When this input is true and CR is false, the value in Current is set equal to the value in CPreset. This input is false when not connected.		

Parameter	Description	Variable Type	Data Type/Range
Name	Enter the name of the counter variable you want to use for this instruction.	data structure	counter See also “About Counter Variables”
CPreset	Enter the value you want the counter to count up to or count down from. This value is stored in *name.CPreset*. See “Changing the Preset Value of a Counter Instruction by Using Ladder Logic,” section 2.3.	constant	double integer

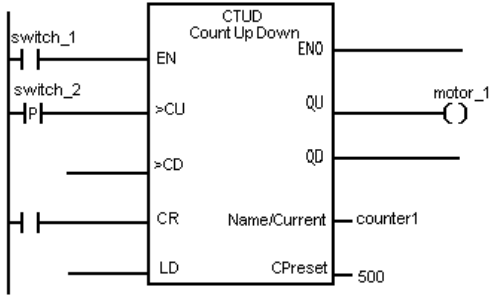
2.1.2 Output Parameters for the Count Up Down Instruction

This table lists the outputs for the CTUD instruction. To use them, connect them to a contact, coil, or Boolean input of another instruction.

Parameter	Description
ENO	Use this output as the input to another instruction for easily chaining multiple instructions. This output follows the state of EN.
QU	This output is true when the value in Current is greater than or equal to the preset value (CPreset).
QD	This output is true when the value in Current is less than or equal to 0.

2.1.3 Example of a Count Up Down Instruction

This logic shows that *counter1* counts up to 500 as long as *switch1* is on and *switch2* transitions from false to true. When the counter reaches 500, QU is set true, and the variable *motor1* becomes true.



2.2 Resetting a Counter

To reset a counter, attach a Boolean parameter to CR so that when the Boolean parameter is true, the value stored in Current is reset to 0. Any transitions to CU or CD are ignored.

2.3 Changing the Preset Value of a Counter Instruction by Using Ladder Logic

You can change the preset value of a counter instruction without having to edit the instruction in the AutoMax Ladder Editor. This is useful for frequently loading different preset values into a counter.

To change the preset value by using ladder logic

Step 1. Place a Move Source Data to Destination (MOVE) instruction in the ladder program.

Step 2. In the In input, enter the value you want to use as a new counter preset value.

Step 3. In the Out output, enter the name of the counter's preset input (*name.*CPreset).

Step 4. Condition the EN input of the MOVE instruction so that the new counter preset is loaded into the Counter instruction.

Tip

You can also use other instructions that have a double integer output to change a counter preset. For example, you can use an ADD instruction to calculate a new preset and place the result in counter's preset input (*name*.CPreset).

3.0 Timer Instructions

Use the Timer instruction to enable and disable activities at pre-defined times. For example, set a timer to turn on a valve or shut off a furnace.

Choose from these timer instructions:

Use this instruction:	To:
Retentive Timer On (RTO)	track accumulated time
Timer Off Delay (TOF)	stop an activity at a preset time interval
Timer On Delay (TON)	start an activity at a preset time interval
Timer Pulse (TP)	enable an output for a preset time interval

The timer instructions use the timer data type for the variable in the Name parameter.

The maximum time interval is 248.5 days (5965 hours). You can specify time in increments of 0.01 seconds.

How Timer Instructions Operate

Once the instruction is enabled, the elapsed value is updated at the start of a program's scan. For example, if a program is scheduled to run every second, the elapsed value for a timer in that program is incremented by 100 each time the program runs. Although the timer is specified in units of 0.01 seconds, the actual duration may be affected by the program's scan time. For example, if the scan time is 0.1 seconds and you set a timer to 0.01 seconds, the actual timer elapsed time will be 0.1seconds, because the timer output will only be updated when the timer executes at 0.1 seconds intervals (the program's scan time).

The T and Q outputs of the timer instructions do not change unless the timer instruction is executed, even if the correct amount of time has elapsed.

Guidelines for Programming Timer Instructions

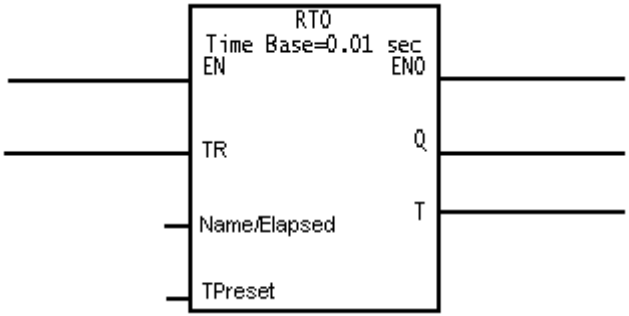
- When inserting timer instructions into a program, assign unique timer variables to each timer instruction. Do not use the same timer variable on more than one instruction in a program. However, you can use individual timer elements (name.Q, name. Elapsed, etc.) as variables on other contacts and instructions.
- When you remove a timer instruction while testing your edits on an online, active program, the Editor considers the timer instruction as being disabled. Therefore, should you ever re-instate that timer instruction, it is inserted into the program in a reset state. This means that the timer has lost its accumulated time.

IMPORTANT

Observe the following programming practices when creating ladder programs:

- Avoid using timer instructions in programs that are not executed at periodic intervals because their behavior will be unpredictable.
- Avoid skipping timer instructions using a JMP instruction because the timer's output will not be set unless the timer instruction is executed.
- Do not use timer instructions that use the same global, timer data structure in multiple programs because the timer will gain time.
- You must enter global timers into the variable configurator as five-element, double integer, non-volatile arrays. Example: TIMER1!(4).

3.1 Retentive Timer On (RTO)



Use this instruction to set an event at a preset interval. This instruction retains the Elapsed value after EN goes false and resumes keeping time when EN is true again. The timer can stop and start without the Elapsed value being reset.

When EN is true, the instruction begins incrementing the value in Elapsed. When the value in Elapsed equals the value in TPreset, output Q becomes true. Reset this instruction by setting input TR.

The value in Elapsed is in increments of 0.01 seconds.

3.1.1 Input Parameters for the RTO Instruction

This table lists the inputs for the RTO instruction and the variable type and data type/range that each input supports.

Parameter	Description	Variable Type	Data Type/Range
EN	While this input is true, the instruction executes. When this input is false, the instruction is not executed and ENO is false.	Connect a Boolean input or output.	
TR	To reset the counter to 0, set this input to true. Output Q becomes false.		
Name	Enter the name of the timer variable you want to use for this timer.	data structure	timer See also “About Timer Variables”
TPreset	Enter the value that Elapsed must reach before the instruction sets output Q. Enter a value in increments of 0.01 seconds up to 248.5 days. This value is stored in *name*.TPreset.	constant	double integer (0 to 2147483647) 0 to 248.5 days in 0.01 second intervals

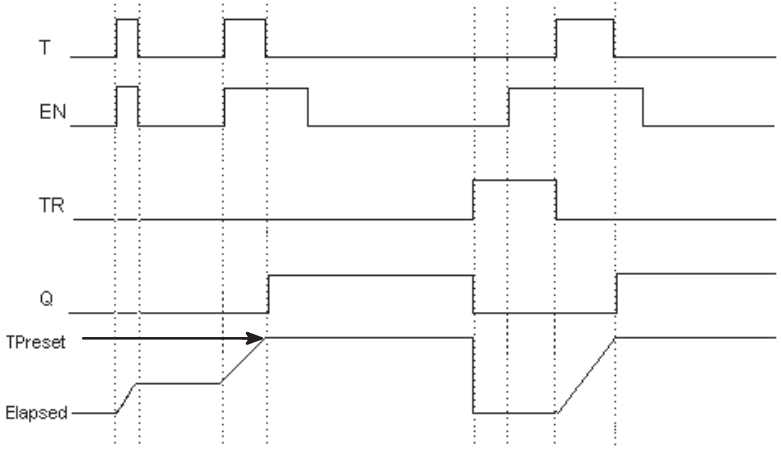
3.1.2 Output Parameters for the RTO Instruction

This table lists the outputs for the RTO instruction and the variable type and data type/range that each output supports.

Parameter	Description	Variable Type	Data Type/Range
ENO	Use this bit as the input to another instruction for easily chaining multiple instructions. This output follows the state of EN.	Connect a contact, coil, or Boolean input to another instruction.	
Q	This output is true when the value in Elapsed equals the value in TPreset. Q is false when R is true.	Connect a Boolean input or a coil.	
T	This output is true when all of the following conditions are met: • the instruction is enabled • the value in Elapsed is less than the value in TPreset • TR is false T is false when the instruction is disabled or the value in Elapsed equals the value in TPreset.		
Elapsed	This value is specified in increments of 0.01 seconds. This value is incremented when EN is true and until its value equals that of TPreset. The value in Elapsed is reset only when TR is true. The value in Elapsed is stored in the element *name*.Elapsed and will not exceed the TPreset value when the instruction is enabled.	constant	double integer (0 to 2147483647) 0 to 248.5 days in 0.01s intervals

3.1.3 RTO Timing Diagram

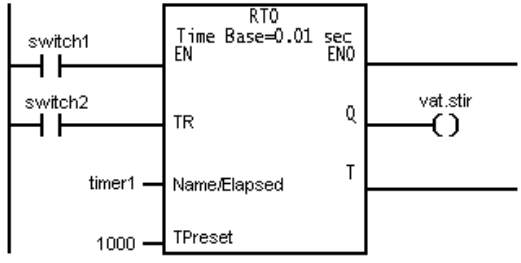
The following diagram shows the interaction between the RTO instruction inputs and outputs at various time intervals.



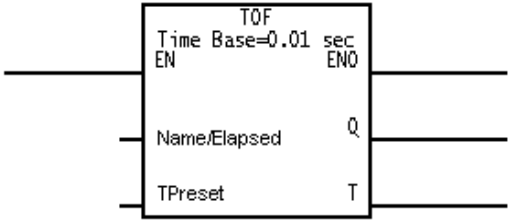
RTO Timing Diagram

3.1.4 Example of an RTO Instruction

This logic shows that as long as *switch1* is true and *switch2* is false, the timer increments toward 1000. When 10 seconds have elapsed, Q is set true, causing the variable *vat.stir* to be set true. When *switch2* is true, TR is set true, which sets the elapsed value to 0 and sets Q false. The variable *vat.stir* is then set false.



3.2 Timer Off Delay (TOF)



Use this instruction to disable an activity at a preset interval. When EN is false, the instruction increments the value in Elapsed until it reaches the value you defined in TPreset. When the value in Elapsed equals the value in TPreset, output Q goes false.

The value in Elapsed is in increments of 0.01 seconds.

3.2.1 Input Parameters for the TOF Instruction

This table lists the inputs for the TOF instruction and the variable type and data type/range that each input supports.

Parameter	Description	Variable Type	Data Type/Range
EN	While this input is false, the instruction begins the timer operation. When EN is true, the value in Elapsed is reset to zero.	Connect a Boolean input or output.	
Name	Enter the name of the timer variable you want to use for this timer.	data structure	timer See also “About Timer Variables”
TPreset	Enter the value that Elapsed must reach before the instruction resets output Q. Enter a value in increments of 0.01 seconds up to 248.5 days. This value is stored in *name*.TPreset.	constant	double integer (0 to 2147483647) 0 to 248.5 days in 0.01 second intervals

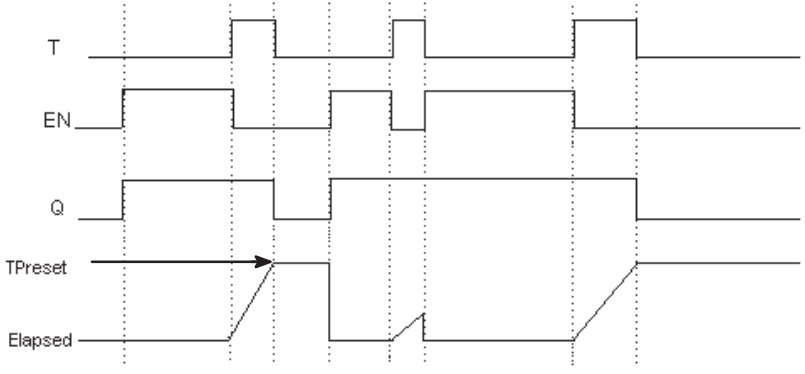
3.2.2 Output Parameters for the TOF Instruction

This table lists the outputs for the TOF instruction and the variable type and data type/range that each output supports.

Parameter	Description	Variable Type	Data Type/Range
ENO	Use this output as the input to another instruction for easily chaining multiple instructions. ENO follows the state of EN.	Connect a contact, coil, or Boolean input to another instruction.	
Q	This output is false when the value in Elapsed equals the value in TPreset. This output is true while EN is true and Elapsed is less than TPreset.		
T	This output is true while EN is false and the value in Elapsed is less than the value in TPreset. T is false while EN is true or the value in Elapsed equals the value in TPreset.		
Elapsed	This value is specified in increments of 0.01 seconds. This value is incremented when EN is false, until its value equals that of TPreset. The value in Elapsed is reset to zero when EN is true. The value in Elapsed is stored in the element *name*.Elapsed and will not exceed the preset value when the instruction is enabled.	constant	double integer (0 to 2147483647) 0 to 248.5 days in 0.01 second intervals

3.2.3 TOF Timing Diagram

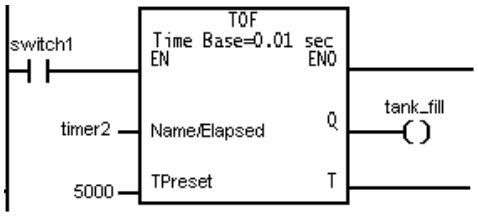
The following diagram shows the interaction between the TOF instruction inputs and outputs at various time intervals.



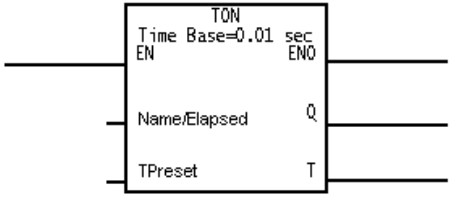
TOF Timing Diagram

3.2.4 Example of a TOF Instruction

This logic shows that as long as *switch1* is false, *timer2* increments to 5000. When 50 seconds have elapsed, the timer sets Q false, causing the variable *tank_fill* to be set false.



3.3 Timer On Delay (TON)



Use this instruction to enable an activity at a preset interval. While EN is true, the instruction increments the value in Elapsed until it reaches the value you defined in TPreset. When the value in Elapsed equals the value in TPreset, output Q is set.

The value in Elapsed is in increments of 0.01 seconds.

3.3.1 Input Parameters for the TON Instruction

This table lists the inputs for the TON instruction and the variable type and data type/range that each input supports.

Parameter	Description	Variable Type	Data Type/Range
EN	While this input is true, the instruction begins the timer operation. When EN is false, the value in Elapsed is reset to zero.	Connect a Boolean input or output.	
Name	Enter the name of the timer variable you want to use for this timer.	data structure	timer See also “About Timer Variables”
TPreset	Enter the value that Elapsed must reach before the instruction sets the output Q. Enter a value in increments of 0.01 seconds up to 248.5 days. This value is stored in *name*.TPreset.	constant	double integer (0 to 2147483647) 0 to 248.5 days in 0.01 second intervals

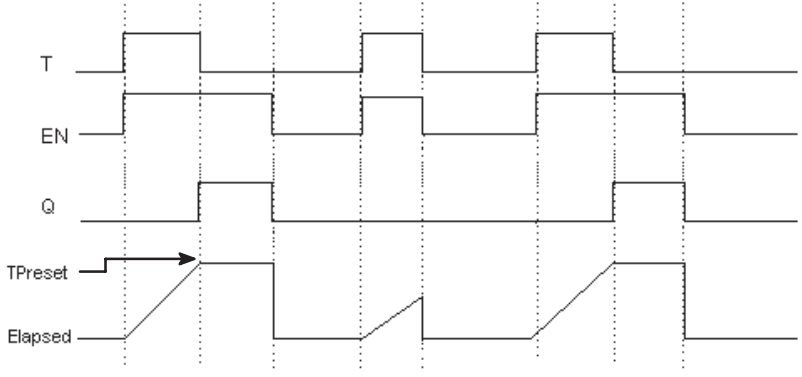
3.3.2 Output Parameters for the TON Instruction

This table lists the outputs for the TON instruction and the variable type and data type/range that each output supports.

Parameter	Description	Variable Type	Data Type/Range
ENO	Use this output as the input to another instruction for easily chaining multiple instructions. ENO follows the state of EN.	Connect a contact, coil, or Boolean input to another instruction.	
Q	This output is true when the instruction is enabled and Elapsed is equal to TPreset. Q is false when EN is false or when Elapsed in less than TPreset.		
T	This output is true while EN is true and the value in Elapsed is less than the value in TPreset. T is false when EN is false or the value in Elapsed is equal to the value in TPreset.		
Elapsed	This value is specified in increments of 0.01 seconds. This value is incremented when EN is true, until its value equals that of TPreset. The Elapsed value is reset to zero when EN is false. The value in Elapsed is stored in the element *name*.Elapsed and will not exceed the TPreset value when the instruction is enabled.	constant	double integer (0 to 2147483647) 0 to 248.5 days in 0.01 second intervals

3.3.3 TON Timing Diagram

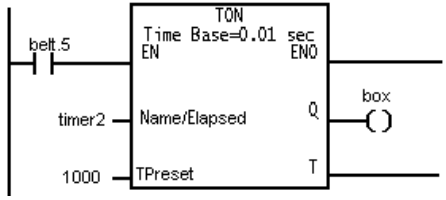
The following diagram shows the interaction between the TON instruction inputs and outputs at various time intervals.



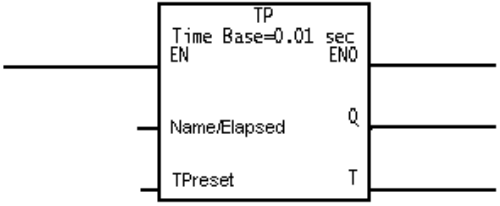
TON Timing Diagram

3.3.4 Example of a TON Instruction

This logic shows that as long as *belt.5* is true *timer2* increments to 1000. When 10 seconds have elapsed, the timer sets Q true, causing the variable *box* to be set true.



3.4 Timer Pulse (TP)



Use this instruction to enable an output for a preset amount of time. This instruction guarantees that an output remains on for a preset time, regardless of the state of the EN input.

When EN is true, the instruction sets outputs Q and T and begins counting towards the preset value. The timer increments regardless of EN's state until the value in Elapsed equals that of TPreset. When these values are equal, outputs Q and T become false. The value in Elapsed is reset when EN is false and the value in Elapsed equals that of TPreset. The value in Elapsed is in increments of 0.01 seconds.

3.4.1 Input Parameters for the TP Instruction

This table lists the inputs for the TP instruction and the variable type and data type/range that each input supports.

Parameter	Description	Variable Type	Data Type/Range
EN	While this input is true, the instruction begins the timer operation. When EN is false and TPreset is equal to Elapsed, the value in Elapsed is reset to zero.	Connect a Boolean input or output.	
Name	Enter the name of the timer variable you want to use for this timer.	data structure	timer See also “About Timer Variables”
TPreset	Enter the value that Elapsed must reach before the instruction resets outputs Q and T. Enter a value in increments of 0.01 seconds up to 248.5 days. This value is stored in *name.*TPreset.	constant	double integer (0 to 2147483647) 0 to 248.5 days in 0.01 second intervals

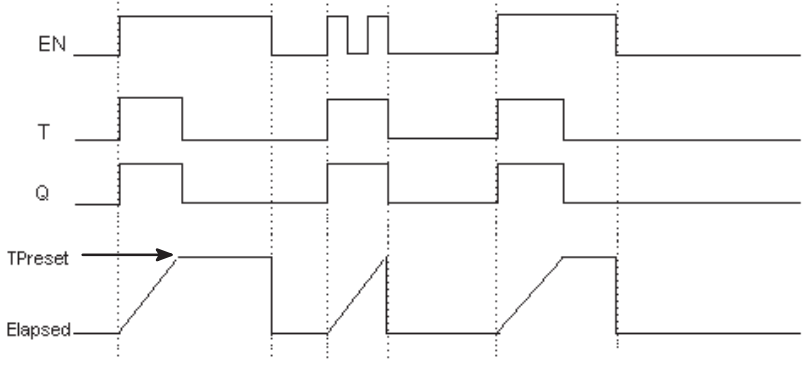
3.4.2 Output Parameters for the TP Instruction

This table lists the outputs for the TP instruction and the variable type and data type/range that each output supports.

Parameter	Description	Variable Type	Data Type/Range
ENO	Use this output as the input to another instruction for easily chaining multiple instructions. ENO follows the state of EN.	Connect a contact, coil, or Boolean input to another instruction.	
Q, T	These outputs are set true when EN is true and the value in Elapsed is less than that in TPreset. Q and T remain true while the value in Elapsed is less than the value in TPreset. These outputs are set false when the value in Elapsed equals the value in TPreset.		
Elapsed	This value is specified in increments of 0.01 seconds. This value is incremented when EN is true, until its value equals that of TPreset. The Elapsed value is reset to zero when EN is false and Elapsed and TPreset are equal. The value in Elapsed is stored in the element *name*.Elapsed and will not exceed the TPreset value when the instruction is enabled.	constant	double integer (0 to 2147483647) 0 to 248.5 days in 0.01 second intervals

3.4.3 TP Timing Diagram

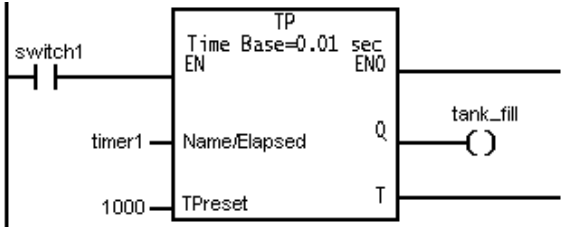
The following diagram shows the interaction between the TP instruction inputs and outputs at various time intervals.



TP Timing Diagram

3.4.4 Example of a TP Instruction

This logic shows that *timer1* begins incrementing when *switch1* is true. The variable *tank_fill* will be on until 100 seconds have elapsed, at which time Q is set false. As soon as the timer begins to increment, the T and Q outputs are set true. The timer continues to count until 10 seconds have elapsed, at which time T, Q, and the variable *tank_fill* are set false.



3.5 Changing the Preset Value of a Timer Instruction by Using Ladder Logic

You can change the preset value of a timer instruction without having to edit the instruction in the AutoMax Ladder Editor. This is useful for frequently loading different preset values into a timer. You can change the preset value of a timer by using a ladder logic instruction with a double integer output for global and local timer presets.

To change the preset value by using ladder logic

Step 1. Place a Move Source Data to Destination (MOVE) instruction in the ladder program.

Step 2. In the In input, enter the value you want to use as a new timer preset value.

Step 3. In the Out output, enter the name of the timer's preset input (*name.*TPreset).

Step 4. Condition the EN input of the MOVE instruction so that the new timer preset is loaded into the timer instruction.

Tip

You can also use other instructions that have a double integer output to change a timer preset. For example, you can use an ADD instruction to calculate a new preset and place the result in timer's preset input (*name*.TPreset).

4.0 Compare Instructions

Use the Compare instructions to compare two or three integer or double integer variables. Choose from these instructions:

- Equal To (EQ)
- Greater Than or Equal To (GE)
- Greater Than (GT)
- Less Than or Equal To (LE)
- Less Than (LT)
- LIMIT (Limit)
- Mask Compare (MSK)
- Not Equal To (NE)

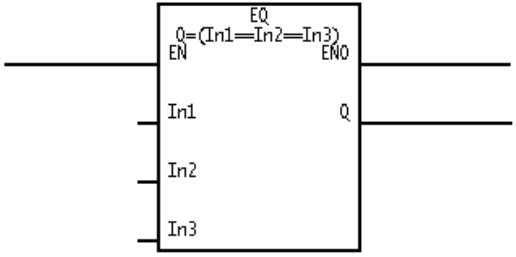
The supported parameters are:

- simple integers and double integers
- integer and double integer constants
- elements of integer and double integer arrays
- Timer variables (*name*.TPreset and *name.*Elapsed)
- Counter variables (*name*.CPreset and *name.*Current)

See each input and output parameter description for each instruction for specific information.

When integer and double integer variable types are mixed within a compare instruction, integer values are converted to signed, double integer values before the compare operation is performed. An exception is the MSK instruction, which converts integer values to unsigned double integers.

4.1 Equal To (EQ)



Use this instruction to test whether two or three values are equal. While EN is true, the instruction determines whether In1, In2, and In3 are equal. If the values are equal, Q becomes true.

4.1.1 Input Parameters for the Equal To Instruction

This table lists the inputs for the EQ instruction and the variable type and data type/range that each input supports.

Parameter	Description	Variable Type	Data Type/Range
EN	When this input is true, the instruction executes. When this input is false, the instruction is not executed and ENO is false.	Connect a Boolean input or output.	
In1 In2 In3	Enter the parameters you want to compare in these inputs. You must specify something for at least In1 and In2. In3 is optional.	• simple • constant • element of an array	• integer • double integer • timer (*name*.TPreset and *name*.Elapsed) • counter (*name*.CPreset and *name*.Current)

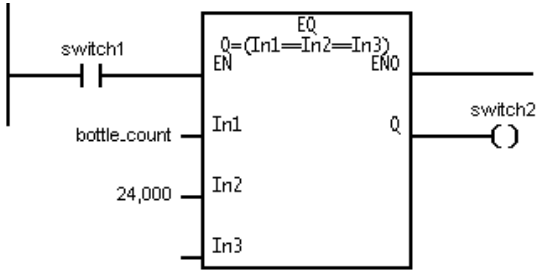
4.1.2 Output Parameters for the Equal To Instruction

This table lists the outputs for the EQ instruction. To use these outputs, connect them to a contact, coil, or Boolean input of another instruction.

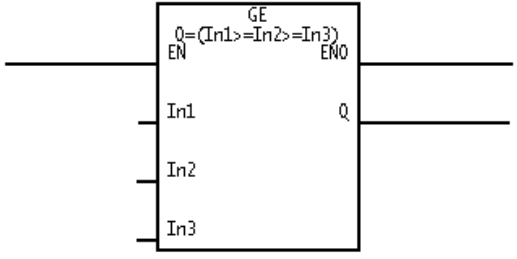
Parameter	Description
ENO	Use this output as the input to another instruction for easily chaining multiple instructions. This output follows the state of input EN unless an error occurs.
Q	This output is true when all of the following conditions occur: • the instruction is enabled • In1 is equal to In2 • if In3 is used, In2 is equal to In3 Otherwise, Q is false.

4.1.3 Example of an Equal To Instruction

This logic shows that when *switch1* is true the variable *bottle_count* will be compared to determine if it is equal to 24,000. If so, Q becomes true, which sets *switch2* true.



4.2 Greater Than Or Equal To (GE)



Use this instruction to test whether two or three values are greater than or equal to each other.

While EN is true, the instruction determines whether In1 is greater than or equal In2. If In3 is used, the instruction determines whether In2 is greater than or equal to In3. If the tested values are greater than or equal, Q becomes true.

4.2.1 Input Parameters for the Greater Than Or Equal To Instruction

This table lists the inputs for the GE instruction and the variable type and data type/range that each input supports.

Parameter	Description	Variable Type	Data Type/Range
EN	When this input is true, the instruction executes. When this input is false, the instruction is not executed and ENO is false.	Connect a Boolean input or output.	
In1 In2 In3	Enter the parameters you want to compare in these inputs. You must specify something for at least In1 and In2. In3 is optional.	• simple • constant • element of an array	• integer • double integer • timer (*name*.TPreset and *name*.Elapsed) • counter (*name*.CPreset and *name*. Current)

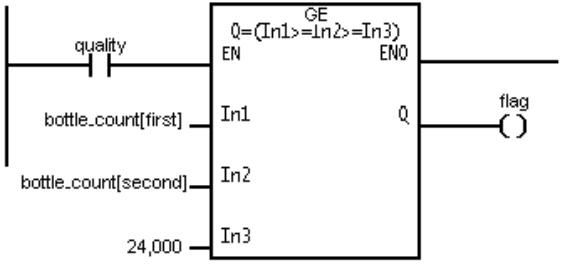
4.2.2 Output Parameters for the Greater Than Or Equal To Instruction

This table lists the outputs for the GE instruction. To use these outputs, connect them to a contact, coil, or Boolean input of another instruction.

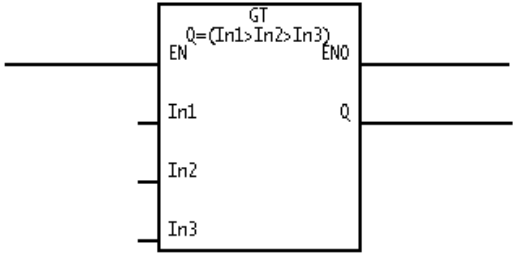
Parameter	Description
ENO	Use this output as the input to another instruction for easily chaining multiple instructions. This output follows the state of EN unless an error occurs.
Q	This output is set true when all of the following conditions occur: • the instruction is enabled • In1 is greater than or equal to In2 • if In3 is used, In2 is greater than or equal to In3 Otherwise, Q is false.

4.2.3 Example of a Greater Than Or Equal To Instruction

This logic shows that when the variable *quality* is true, the instruction compares the value of *bottle_count[first]* to the value of *bottle_count[second]*. The value of *bottle_count[second]* is then compared to 24,000. If each of these values is greater than or equal to the next, Q is set true, setting the variable *flag* true.



4.3 Greater Than (GT)



Use this instruction to test whether two or three values are greater than each other. While EN is true, the instruction determines whether In1 is greater than In2. If In3 is used, the instruction determines whether In2 is greater than In3. If the tested values are greater than each other, Q is true.

4.3.1 Input Parameters for the Greater Than Instruction

This table lists the inputs for the GT instruction and the variable type and data type/range that each input supports.

Parameter	Description	Variable Type	Data Type/Range
EN	When this input is true, the instruction executes. When this input is false, the instruction is not executed, and ENO is false.	Connect a Boolean input or output.	
In1 In2 In3	Enter the parameters you want to compare in these inputs. You must specify something for at least In1 and In2. In3 is optional.	• simple • constant • element of an array	• integer • double integer • timer (*name*.TPreset and *name.*Elapsed) • counter (*name*.CPreset and *name.*Current)

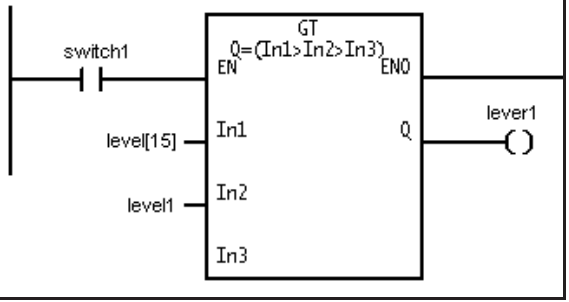
4.3.2 Output Parameters for the Greater Than Instruction

This table lists the outputs for the GT instruction. To use these outputs, connect them to a contact, coil, or Boolean input of another instruction.

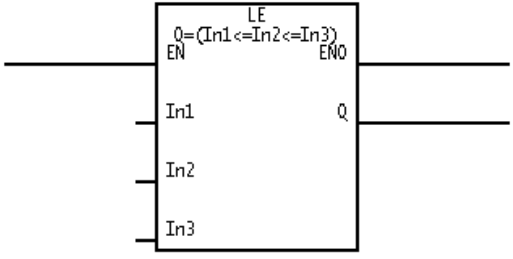
Parameter	Description
ENO	Use this output as the input to another instruction for easily chaining multiple instructions. This output follows the state of EN unless an error occurs.
Q	This output is set true when all of the following conditions occur: • the instruction is enabled • In1 is greater than In2 • if In3 is used, In2 is greater than In3 Otherwise, Q is false.

4.3.3 Example of a Greater Than Instruction

This logic shows that when the variable *switch1* is set true, the instruction tests the variable *level[15]* to determine if its value is greater than the variable *level1*. If so, Q is set true, which sets *lever1* true.



4.4 Less Than Or Equal To (LE)



Use this instruction to test whether two or three values are less than or equal to each other. While EN is true, the instruction determines whether In1 is less than or equal to In2. If In3 is used, the instruction determines whether In2 is less than or equal to In3. If the tested values are less than or equal, Q is true.

4.4.1 Input Parameters for the Less Than Or Equal To Instruction

This table lists the inputs for the LE instruction and the variable type and data type/range that each input supports.

Parameter	Description	Variable Type	Data Type/Range
EN	When this input is true, the instruction executes. When this input is false, the instruction is not executed and ENO is false.	Connect a Boolean input or output.	
In1 In2 In3	Enter the parameters you want to compare in these inputs. You must specify something for at least In1 and In2. In3 is optional.	• simple • constant • element of an array	• integer • double integer • timer (*name*.TPreset and *name*.Elapsed) • counter (*name*.CPreset and *name*.Current)

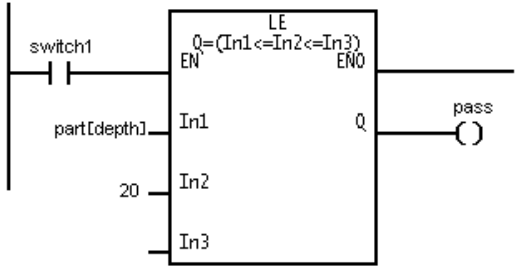
4.4.2 Output Parameters for the Less Than Or Equal To Instruction

This table lists the outputs for the LE instruction. To use these outputs, connect them to a contact, coil, or Boolean input to another instruction.

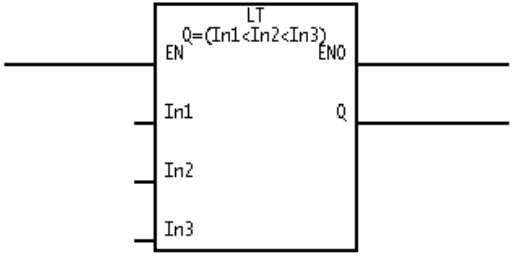
Parameter	Description
ENO	Use this output as the input to another instruction for easily chaining multiple instructions. This output follows the state of EN unless an error occurs.
Q	This output is set true when all of the following conditions occur: • the instruction is enabled • In1 is less than or equal to In2 • if In3 is used, In2 is less than or equl to In3 Otherwise, Q is false.

4.4.3 Example of a Less Than Or Equal To Instruction

This logic shows that when *switch1* is true, the instruction compares the variable *part[depth]* to 20. If *part[depth]* is less than or equal to 20, Q is set true, which sets the variable *pass* true.



4.5 Less Than (LT)



Use this instruction to test whether two or three values are less than each other. While EN is true, the instruction determines whether In1 is less than In2. If In3 is used, the instruction determines whether In2 is less than In3. If they are, Q is true.

4.5.1 Input Parameters for the Less Than Instruction

This table lists the inputs for the LT instruction and the variable type and data type/range each inputs supports.

Parameter	Description	Variable Type	Data Type/Range
EN	When this input is true, the instruction executes. When this input is false, the instruction is not executed and ENO is false.	Connect a Boolean input or output.	
In1 In2 In3	Enter the parameters you want to compare in these inputs. You must specify something for at least In1 and In2. In3 is optional.	• simple • constant • element of an array	• integer • double integer • timer (*name*.TPreset and *name.*Elapsed) • counter (*name*.CPreset and *name.*Current)

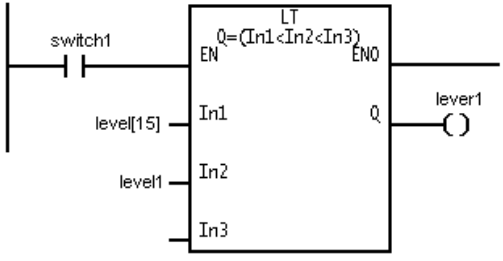
4.5.2 Output Parameters for the Less Than Instruction

This table lists the outputs for the LT instruction. To use these outputs, connect them to a contact, coil, or Boolean input of another instruction.

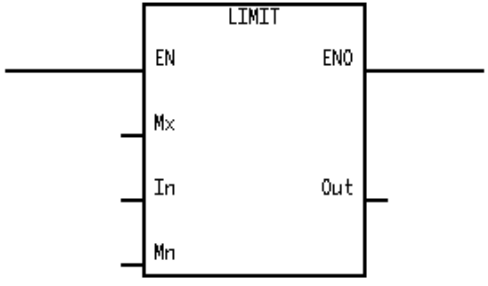
Parameter	Description
ENO	Use this output as the input to another instruction for easily chaining multiple instructions. This output follows the state of EN unless an error occurs.
Q	This output is set true when the following conditions occur: • the instruction is enabled • In1 is less than In2 • if In3 is used, In2 is less than In3 Otherwise, Q is false.

4.5.3 Example of a Less Than Instruction

This logic shows that when the variable *switch1* is true, the variable *level[15]* is tested to determine if its value is less than the variable *level1*. If so, Q is set true, which sets *lever1* true.



4.6 Limit (LIMIT)



Use this instruction to clamp values that are outside a specified range. While EN is true, the instruction determines whether In falls within the maximum and minimum limits specified for Mx and Mn. Out contains the value of In unless its value does not fall within the limits.

If the value of In is:	**Then Out contains the value of:**
greater than Mx	Mx
less than Mn	Mn

4.6.1 Input Parameters for the Limit Instruction

This table lists the inputs for the LIMIT instruction and the variable type and data type/range that each input supports.

Parameter	Description	Variable Type	Data Type/Range
EN	When this input is true, the instruction executes. When this input is false, the instruction is not executed and ENO is false.	Connect a Boolean input or output.	
Mx	Enter the upper limit for In.	• simple • constant • element of an array	• integer • double integer • timer (*name*.TPreset and *name.*Elapsed) • counter (*name*.CPreset and *name.*Current)
In	Enter the variable that you want to limit.		
Mn	Enter the lower limit for In.		

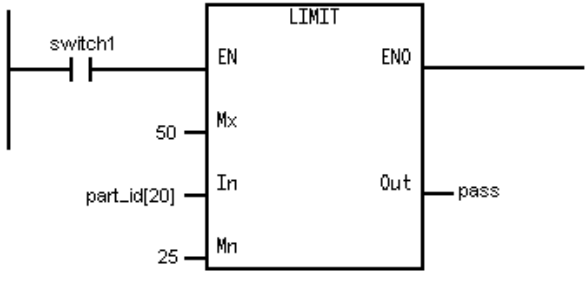
4.6.2 Output Parameters for the Limit Instruction

This table lists the outputs for the LIMIT instruction and the variable type and data type/range that each output supports.

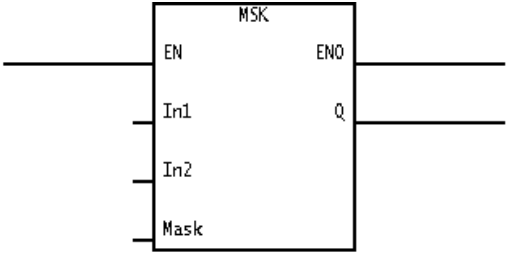
Parameter	Description	Variable Type	Data Type/Range
ENO	Use this output as the input to another instruction for easily chaining multiple instructions. This output follows the state of EN unless an error occurs.	Connect a contact, coil, or a Boolean input to another instruction.	
Out	When the instruction is enabled, Out contains the value in In when In falls within the defined limits. If the value in In is less than the lower limit, Out contains the value in Mn. If the value in In is greater than the value in Mx, Output contains the value in Mx.	• simple • element of an array	• integer • double integer • timer (*name*.TPreset and *name.*Elapsed) • counter (*name*.CPreset and *name.*Current)

4.6.3 Example of a Limit Instruction

This logic shows that when the variable *switch1* is true the variable *part_id[20*] is checked to determine if its value falls between 50 and 25. The variable *pass* contains the limited value.



4.7 Mask Compare (MSK)



Use this instruction to apply a bit-mask to two variables and then have them compared to determine if the values are equal.

While EN is true, the value of Mask is logically ANDED with the variables In1 and In2. After applying the bit-mask, the instruction then compares In1 and In2 to determine if they are equal. If the variables are equal, Q is true.

In1, In2, and Mask are treated as unsigned integers. If any of these parameters is converted to double integers, the most significant 16 bits are set to zero.

4.7.1 Input Parameters for the Mask Compare Instruction

This table lists the inputs for the MSK instruction and the variable type and data type/range that each input supports.

Parameter	Description	Variable Type	Data Type/Range
EN	When this input is true, the instruction executes. When this input is false, the instruction is not executed, and ENO is false.	Connect a Boolean input or output.	
In1 In2	Enter the variables to which you want to apply the bit mask. These variables are treated as unsigned integers.	• simple • constant • element of an array	• integer • double integer • timer (*name*.TPreset and *name*.Elapsed) • counter (*name*.CPreset and *name*.Current)
Mask	The Mask specifies which bits to pass or block. Enter the parameter that you want to use as the bit-mask. The mask passes bits equal to 1 and blocks bits equal to 0.	• constant • element of an array	• integer (0 to FFFF) • double integer (0 to FFFFFFFF)

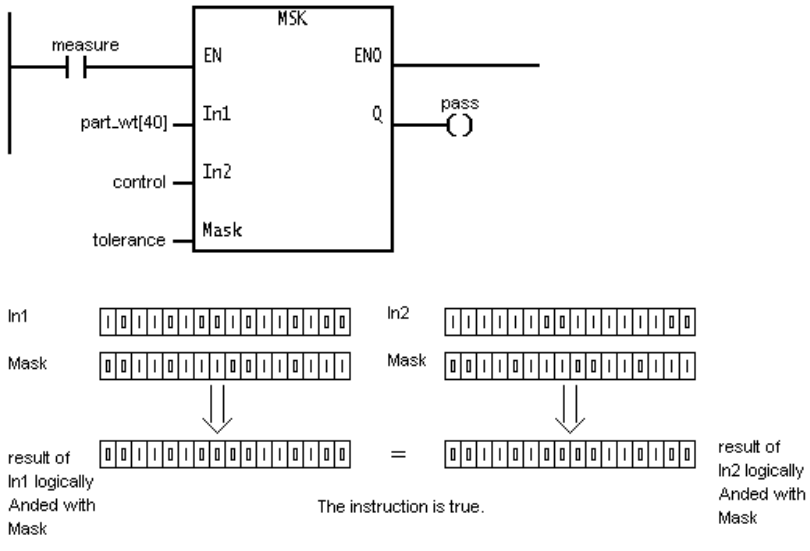
4.7.2 Output Parameters for the Mask Compare Instruction

This table lists the outputs for the MSK instruction. To use these outputs, connect them to a contact, coil, or Boolean input.

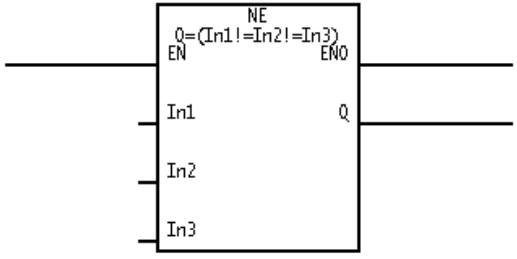
Parameter	Description
ENO	Use this output as the input to another instruction for easily chaining multiple instructions. This output follows the state EN unless an error occurs.
Q	When the instruction is enabled, this output is true after the bit mask has been applied to In1 and In2 and the results were compared and determined to be equal. Otherwise, Q is false.

4.7.3 Example of a Mask Compare Instruction

This logic shows that when the variable *measure* is true a bit mask identified as *tolerance* is applied to the variables *part_wt[40]* and *control*. The results are then compared to determine if they are equal. If so, Q and the variable are set true.



4.8 Not Equal To (NE)



Use this instruction to test whether two or three values are not equal. While EN is true, the instruction determines whether the values in In1, In2, and In3 are not equal. If the values are not equal, Q becomes true.

4.8.1 Input Parameters for the Not Equal To Instruction

This table lists the inputs for the NE instruction and the variable type and data type/range that each input supports.

Parameter	Description	Variable Type	Data Type/Range
EN	When this input is true, the instruction executes. When this input is false, the instruction is not executed, and ENO is false.	Connect a Boolean input or output.	
In1 In2 In3	Enter the parameters you want to compare in these inputs. You must specify something for at least In1 and In2. In3 is optional.	• simple • constant • element of an array	• integer • double integer • timer (*name*.TPreset and *name.*Elapsed) • counter (*name*.CPreset and *name.*Current)

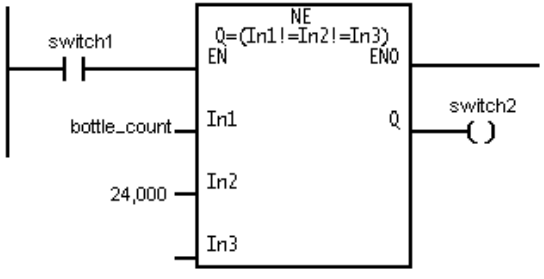
4.8.2 Output Parameters for the Not Equal To Instruction

This table lists the outputs for the NE instruction. To use these outputs, connect them to a contact, coil, or a Boolean input to another instruction.

Parameter	Description
ENO	Use this output as the input to another instruction for easily chaining multiple instructions. This output follows the state of EN unless an error occurs.
Q	This bit true when all of the following conditions occur: • the instruction is enabled • In1 is not equal to In2 • if In3 is used, none of the values are equal Otherwise, Q is false.

4.8.3 Example of a Not Equal To Instruction

This logic shows that when *switch1* is true the variable *bottle_count* will be compared to determine if it is not equal to 24,000. If so, Q will be set true.



4.9 Errors Caused by Compare Instructions

This section describes the possible errors for all compare instructions and those additional errors specific to the LIMIT instruction.

4.9.1 Errors Caused by All Compare Instructions

These errors can occur when you are using the compare instructions in a program. They are logged in the error log.

If this error occurs:	Then:	Do the following:
The array index is negative.	ENO is set according to ERROR_ENO, and element zero of the array is used for the instruction's operation.	Specify a valid array element.
The array index is too large.	ENO is set according to ERROR_ENO, and the last element of the array is used for the instruction's operation.	Specify a valid array element.

4.9.2 Errors Caused by the Limit Instruction

These errors can occur when you are using the LIMIT instruction in a program. They are logged in the error log.

If this error occurs:	Then:	Do the following:
The result is larger than what Out's data type supports.	ENO is set according to ERROR_ENO, and Out contains the largest signed value allowed for the data type being used.	Specify a larger data type for Out. For example, if you are using integers, specify the data type as a double integer.
Minimum is greater than Maximum.	ENO is set according to ERROR_ENO, and the values for Mn and Mx are swapped before the limit operation is performed.	Make sure the value for Mx is larger than that for Mn.

5.0 Compute Instructions

Use compute instructions to perform math operations.

Choose from these compute instructions:

- Absolute Value (ABS)
- Add (ADD)
- Divide (DIV)
- Modulo (MOD)
- Multiply (MUL)
- Multiply Divide (MDV)
- Negate (NEG)
- Square Root (SQRT)
- Subtract (SUB)

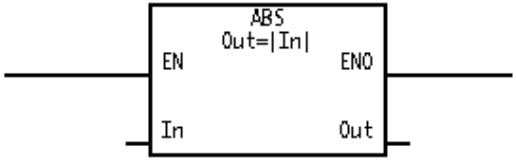
The supported parameters are:

- simple integers and double integers
- integer and double integer constants
- elements of integer and double integer arrays
- Timer variables (*name*.TPreset and *name*.Elapsed)
- Counter variables (*name*.CPreset and *name*.Current)

See each input and output parameter description for each instruction for specific information.

When integer and double integer variable types are mixed within a compute instruction, integer values are converted to signed double integer values before the compute operation is performed.

5.1 Absolute Value (ABS)



Use this instruction to calculate the absolute value of a variable or constant.

While EN is true, the instruction calculates the absolute value of In. The result is stored in Out.

5.1.1 Input Parameters for the Absolute Value Instruction

This table lists the inputs for the ABS instruction and the variable and type and data type/range that each input supports.

Parameter	Description	Variable Type	Data Type/Range
EN	When this input is true, the instruction executes. When this input is false, the instruction is not executed, and ENO is false.	Connect a Boolean input or output.	
In	Enter the variable or constant of which you want the absolute value.	• simple • constant • element of an array	• integer • double integer • timer (*name*.TPreset and *name.*Elapsed) • counter (*name*.CPreset and *name.*Current)

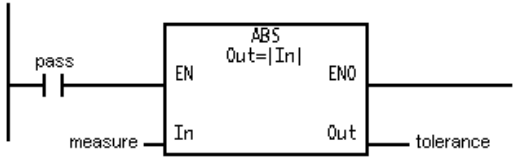
5.1.2 Output Parameters for the Absolute Value Instruction

This table lists the outputs for the ABS instruction and the variable type and data type/range that each output supports.

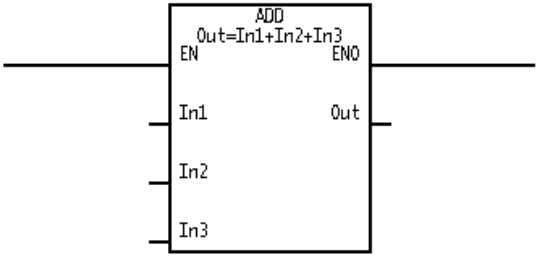
Parameter	Description	Variable Type	Data Type/Range
ENO	Use this output as the input to another instruction for easily chaining multiple instructions. This output follows the state of the EN input unless an error occurs.	Connect a contact, coil, or a Boolean input to another instruction.	
Out	This output contains the absolute value of In.	• simple • element of an array	• integer (0 to 32767) • double integer (0 to 2147483647) • timer (*name*.TPreset and *name.*Elapsed) • counter (*name*.CPreset and *name.*Current)

5.1.3 Example of an Absolute Value Instruction

This logic shows that when the variable *pass* is true the instruction calculates the absolute value of the variable *measure* and stores it in the variable *tolerance*.



5.2 Add (ADD)



Use this instruction to add two or three variables or constants together. While EN is true, the instruction adds the values of In1, In2, and In3. The result is stored in Out.

5.2.1 Input Parameters for the Add Instruction

This table lists the inputs for the Add instruction and the variable type and data type/range that each input supports.

Parameter	Description	Variable Type	Data Type/Range
EN	When this input is true, the instruction executes. When this input is false, the instruction is not executed, and ENO is false.	Connect a Boolean input or output.	
In1 In2 In3	Enter the variables or constants you want to add. You must specify values for at least In1 and In2. In3 is optional.	• simple • constant • element of an array	• integer • double integer • timer (*name*.TPreset and *name.*Elapsed) • counter (*name*.CPreset and *name.*Current)

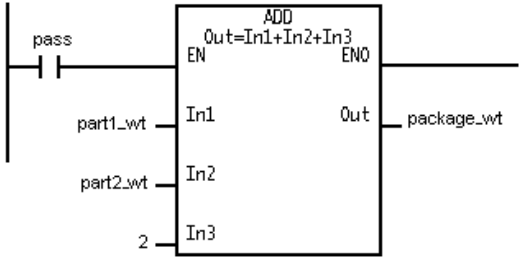
5.2.2 Output Parameters for the Add Instruction

This table lists the outputs for the Add instruction and the variable type and data type/range that each output supports.

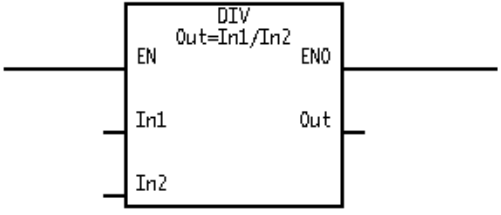
Parameter	Description	Variable Type	Data Type/Range
ENO	Use this output as the input to another instruction for easily chaining multiple instructions. This output follows the state of the EN input unless an error occurs.	Connect a contact, coil, or a Boolean input to another instruction.	
Out	This output contains the result of adding the values in In1 and In2 or In1, In2, and In3.	• simple • element of an array	• integer • double integer • timer (*name*.TPreset and *name.*Elapsed) • counter (*name*.CPreset and *name.*Current)

5.2.3 Example of the Add Instruction

This logic shows that when the variable *pass* is true the instruction adds the variables *part1_wt*, *part2_wt*, and the constant 2. The result is stored in the variable *package_wt*.



5.3 Divide (DIV)



Use this instruction to divide two variables or constants.

While EN is true, the instruction divides In1 by In2. The whole number of the quotient (truncated quotient) is stored in Out. For example, if the result (quotient) of the division is 1.6, Out contains a value of 1.

5.3.1 Input Parameters for the Divide Instruction

This table lists the inputs for the DIV instruction and the variable type and data type/range that each input supports.

Parameter	Description	Variable Type	Data Type/Range
EN	While this input is true, the instruction executes. When this input is false, the instruction does not execute and ENO is false.	Connect a Boolean input or output.	
In1 In2	Enter the variables or constants you want to divide. In1 is divided by In2. In2 cannot be 0.	• simple • constant • element of an array	• integer • double integer • timer (*name*.TPreset and *name.*Elapsed) • counter (*name*.CPreset and *name.*Current)

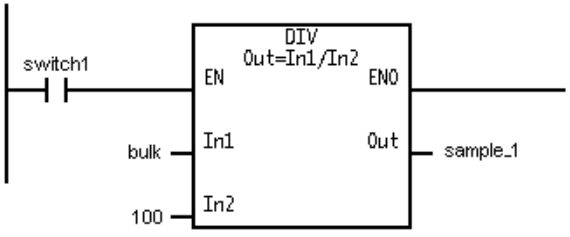
5.3.2 Output Parameters for the Divide Instruction

This table lists the outputs for the DIV instruction and the variable type and data type/range that each output supports.

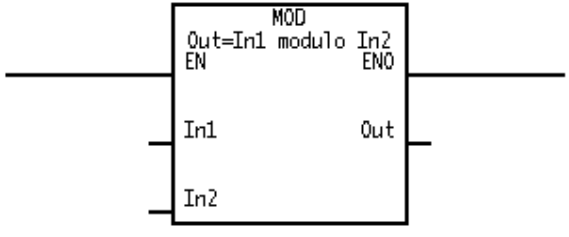
Parameter	Description	Variable Type	Data Type/Range
ENO	Use this output as the input to another instruction for easily chaining multiple instructions. This output follows the state of the EN input unless an error occurs.	Connect a contact, coil, or a Boolean input to another instruction.	
Out	This output contains the truncated quotient.	• simple • constant • element of an array	• integer • double integer • timer (*name*.TPreset and *name.*Elapsed) • counter (*name*.CPreset and *name.*Current)

5.3.3 Example of a Divide Instruction

This logic shows that when the variable *switch1* is true the instruction divides the variable *bulk* by 100 and stores the result in the variable *sample_1*.



5.4 Modulo (MOD)



Use this instruction to calculate the remainder resulting from dividing two variables or constants.

While EN is true, the instruction calculates the remainder of In1 divided by In2. The result is stored in Out.

5.4.1 Input Parameters for the Modulo Instruction

This table lists the inputs for the MOD instruction and the variable type and data type/range that each input supports.

Parameter	Description	Variable Type	Data Type/Range
EN	While this input is true, the instruction executes. When this input is false, the instruction does not execute and ENO is false.	Connect a Boolean input or output.	
In1 In2	Enter the variables or constants you want to divide and have the remainder calculated. In1 is divided by In2. In2 cannot be 0.	• simple • constant • element of an array	• integer • double integer • timer (*name*.TPreset and *name.*Elapsed) • counter (*name*.CPreset and *name.*Current)

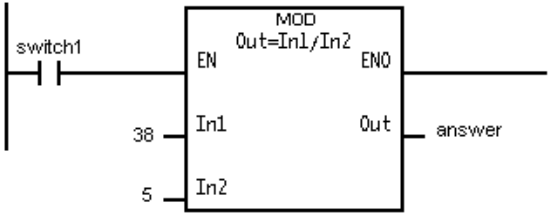
5.4.2 Output Parameters for the Modulo Instruction

This table lists the outputs for the MOD instruction and the variable type and data type/range that each output supports.

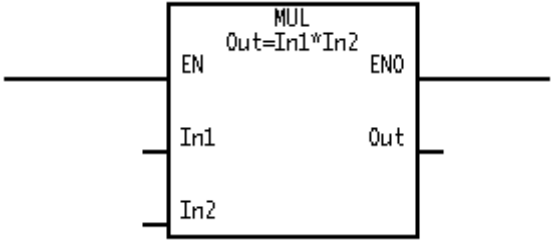
Parameter	Description	Variable Type	Data Type/Range
ENO	Use this output as the input to another instruction for easily chaining multiple instructions. This output follows the state of the EN input unless an error occurs.	Connect a contact, coil, or a Boolean input to another instruction.	
Out	This output contains the remainder of the division operation.	• simple • element of an array	• integer • double integer • timer (*name*.TPreset and *name.*Elapsed) • counter (*name*.CPreset and *name.*Current)

5.4.3 Example of the Modulo Instruction

This logic shows that when the variable *switch1* is true the instruction divides the constant 38 by 5 and stores the remainder of the division (3) in the variable *answer*.



5.5 Multiply (MUL)



Use this instruction to multiply two variables or constants together. While EN is true, the instruction multiplies the values of In1 and In2. The result is stored in Out.

5.5.1 Input Parameters for the Multiply Instruction

This table lists the inputs for the MUL instruction and the variable type and data type/range that each input supports.

Parameter	Description	Variable Type	Data Type/Range
EN	While this input is true, the instruction executes. When this input is false, the instruction does not execute, and ENO is false.	Connect a Boolean input or output.	
In1 In2	Enter the variables or constants you want to multiply.	• simple • constant • element of an array	• integer • double integer • timer (*name*.TPreset and *name.*Elapsed) • counter (*name*.CPreset and *name.*Current)

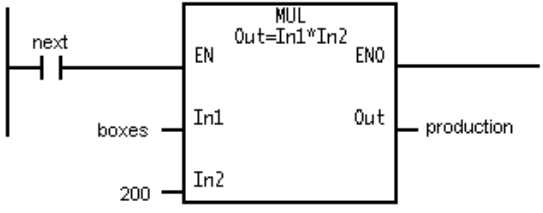
5.5.2 Output Parameters for the Multiply Instruction

This table lists the outputs for the MUL instruction and the variable type and data type/range that each output supports.

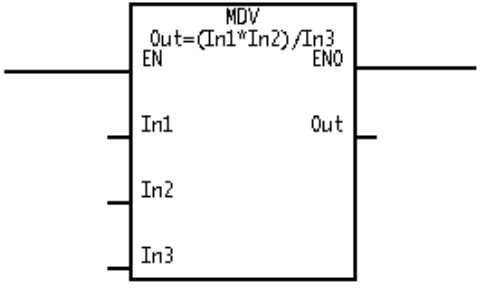
Parameter	Description	Variable Type	Data Type/Range
ENO	Use this output as the input to another instruction for easily chaining multiple instructions. This output follows the state of the EN input unless an error occurs.	Connect a contact, coil, or a Boolean input to another instruction.	
Out	This output contains the result of multiplying the values in In1 and In2.	• simple • element of an array	• integer • double integer • timer (*name*.TPreset and *name.*Elapsed) • counter (*name*.CPreset and *name.*Current)

5.5.3 Example of a Multiply Instruction

This logic shows that when the variable *next* is true the instruction multiplies the variable *boxes* by 200 and stores the product in the variable *production*.



5.6 Multiply Divide (MDV)



Use this instruction to multiply two variables or constants together and divide the result by a third variable or constant. This operation can give you a greater precision than if you use a MUL instruction and then a DIV instruction, because the product of In1 and In2 is kept in double precision and then divided by In3 to reduce it to single precision.

While EN is true, the instruction multiplies the values of In1 and In2, then divides the product by In3. The truncated quotient is stored in Out. For example, if the result was 1.6, Out would contain a value of 1.

5.6.1 Input Parameters for the Multiply Divide Instruction

This table lists the inputs for the MDV instruction and the variable type and data type/range that each input supports.

Parameter	Description	Variable Type	Data Type/Range
EN	While this input is true, the instruction executes. When this input is false, the instruction does not execute and ENO is false.	Connect a Boolean input or output.	
In1 In2	Enter the variables or constants you want to multiply.	• simple • constant • element of an array	• integer • double integer • timer (*name*.TPreset and *name*.Elapsed) • counter (*name*.CPreset and *name*.Current)
In3	Enter the variable or constant by which you want to divide the product of In1 and In2. In3 cannot be 0.		

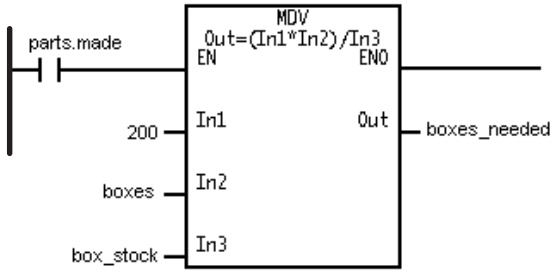
5.6.2 Output Parameters for the Multiply Divide Instruction

This table lists the outputs for the MDV instruction and the variable type and data type/range that each output supports.

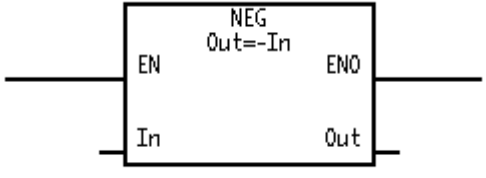
Parameter	Description	Variable Type	Data Type/Range
ENO	Use this output as the input to another instruction for easily chaining multiple instructions. This output follows the state of the EN input unless an error occurs.	Connect a contact, coil, or a Boolean input to another instruction.	
Out	This output contains the result of the calculation.	• simple • element of an array	• integer • double integer • timer (*name*.TPreset and *name.*Elapsed) • counter (*name*.CPreset and *name.*Current)

5.6.3 Example of a Multiply Divide Instruction

This logic shows that when the variable *parts.made* is true the instruction multiplies the constant 200 by the variable *boxes*. The resulting double precision product is then divided by the variable *box_stock*. The single precision result of the calculation is stored in the variable *boxes_needed*.



5.7 Negate (NEG)



Use this instruction to change the sign of a variable or constant. While EN is true, the instruction changes the sign of the value of In. The result is stored in Out.

5.7.1 Input Parameters for the Negate Instruction

This table lists the inputs for the NEG instruction and the variable type and data type/range that each input supports.

Parameter	Description	Variable Type	Data Type/Range
EN	While this input is true, the instruction executes. When this input is false, the instruction does not execute, and ENO is false.	Connect a Boolean input or output.	
In	Enter the variable you want to negate.	• simple • constant • element of an array	• integer • double integer • timer (*name*.TPreset and *name.*Elapsed) • counter (*name*.CPreset and *name.*Current)

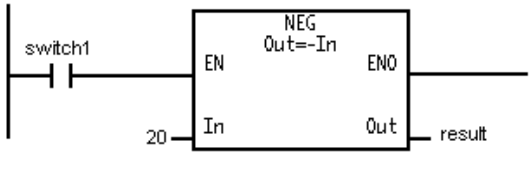
5.7.2 Output Parameters for the Negate Instruction

This table lists the outputs for the NEG instruction and the variable type and data type/range that each output supports.

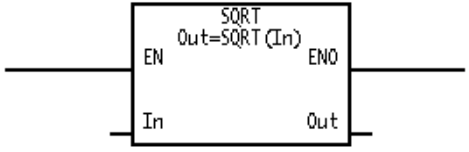
Parameter	Description	Variable Type	Data Type/Range
ENO	Use this output as the input to another instruction for easily chaining multiple instructions. This output follows the state of the EN input unless an error occurs.	Connect a contact, coil, or a Boolean input to another instruction.	
Out	This output contains the negative value of In.	• simple • element of an array	• integer • double integer • timer (*name*.TPreset and *name.*Elapsed) • counter (*name*.CPreset and *name.*Current)

5.7.3 Example of a Negate Instruction

When *switch1* is true, the instruction negates the value of 20 and stores a value of −20 in the variable *result*.



5.8 Square Root (SQRT)



Use this instruction to calculate the square root of a variable or constant. While EN is true, the instruction calculates the square root of In. The truncated result is stored in Out.

5.8.1 Input Parameters for the Square Root Instruction

This table lists the inputs for the SQRT instruction and the variable type and data type/range that each input supports.

Parameter	Description	Variable Type	Data Type/Range
EN	While this input is true, the instruction executes. When this input is false, the instruction does not execute and ENO is false.	Connect a Boolean input or output.	
In	Enter the positive variable or constant for which you want to calculate the square root.	• simple • constant • element of an array	• integer (0 to 32767) • double integer (0 to 2147483647) • timer (*name*.TPreset and *name.*Elapsed) • counter (*name*.CPreset and *name.*Current)

5.8.2 Output Parameters for the Square Root Instruction

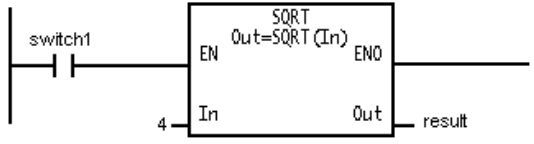
This table lists the outputs for the SQRT instruction and the variable type and data type/range that each output supports.

Parameter	Description	Variable Type	Data Type/Range
ENO	Use this output as the input to another instruction for easily chaining multiple instructions. This output follows the state of the EN input unless an error occurs.	Connect a contact, coil, or a Boolean input to another instruction.	
Out	This output contains the truncated result of the square root of In.	• simple • element of an array	• integer (0 to 32767) • double integer (0 to 46340) • timer (*name*.TPreset and *name.*Elapsed) • counter (*name*.CPreset and *name.*Current)

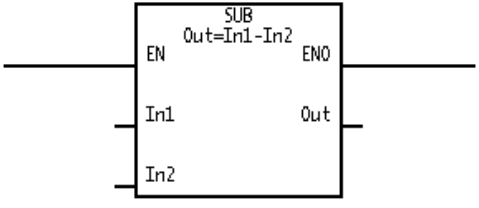
5.8.3 Example of the Square Root Instruction

When *switch1* is true, the instruction calculates the square root of 4 and stores a value of 2 in the variable *result*.

Had In contained a value of 8, the result of the square root calculation still would have been 2, because the result is truncated to a whole number.



5.9 Subtract (SUB)



Use this instruction to subtract two variables or constants. While EN is true, the instruction subtracts In2 from In1. The result is stored in Out.

5.9.1 Input Parameters for the Subtract Instruction

This table lists the inputs for the SUB instruction and the variable type and data type/range that each input supports.

Parameter	Description	Variable Type	Data Type/Range
EN	While this input is true, the instruction executes. When this input is false, the instruction does not execute and ENO is false.	Connect a Boolean input or output.	
In1 In2	Enter the variables or constants you want to subtract (In1 minus In2).	• simple • constant • element of an array	• integer • double integer • timer (*name*.TPreset and *name.*Elapsed) • counter (*name*.CPreset and *name.*Current)

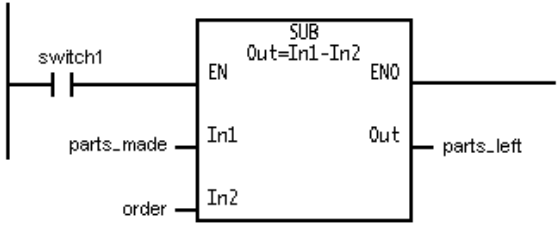
5.9.2 Output Parameters for the Subtract Instruction

This table lists the outputs for the SUB instruction and the variable type and data type/range that each output supports.

Parameter	Description	Variable Type	Data Type/Range
ENO	Use this output as the input to another instruction for easily chaining multiple instructions. This output follows the state of the EN input unless an error occurs.	Connect a contact, coil, or a Boolean input to another instruction.	
Out	This output contains the result of In1 minus In2.	• simple • element of an array	• integer • double integer • timer (*name*.TPreset and *name.*Elapsed) • counter (*name*.CPreset and *name.*Current)

5.9.3 Example of a Subtract Instruction

When the variable *switch1* is true, the instruction subtracts the value of the variable *order* from that of the variable *part_made* and stores the result in the variable *parts_left*.



5.10 Errors Caused by the Compute Instructions

This section describes the possible errors for all compute instructions and those additional errors specific to each compute instruction.

5.10.1 Errors Caused by All Compute Instructions

These errors can occur when you are using compute instructions. They are logged in the error log.

If this error occurs:	Then:	Do the following:
The array index is negative.	ENO is set according to ERROR_ENO, and element zero of the array is used for the instruction's operation.	Specify a valid array element.
The array index is too large.	ENO is set according to ERROR_ENO, and the last element of the array is used for the instruction's operation.	Specify a valid array element.

5.10.2 Errors Caused by the Absolute Value Instruction

This error can occur when you are using the ABS instruction in a program. It is logged in the error log.

If this error occurs:	Then:	Do the following:
The result of the arithmetic calculation is too large for Out.	ENO is set according to ERROR_ENO, and Out contains the largest positive value allowed for its data type.	Specify a larger data type for Out. For example, if you are using integers, specify the data type as a double integer.

5.10.3 Errors Caused by the Addition Instruction

This error can occur when you are using the ADD instruction in a program. It is logged in the error log.

If this error occurs:	Then:	Do the following:
The result of the arithmetic calculation is too large for Out.	ENO is set according to ERROR_ENO, and Out contains the largest signed value allowed for its data type.	Specify a larger data type for Out. For example, if you are using integers, specify the data type as a double integer.

5.10.4 Errors Caused by the Divide, Modulo, and Multiply Divide Instruction

These errors can occur when you are using the DIV, MOD, or MDV instruction in a program. They are logged in the error log.

If this error occurs:	Then:	Do the following:
The result of the arithmetic calculation is too large for Out.	ENO is set according to ERROR_ENO, and Out contains the largest signed value allowed for its data type.	Specify a larger data type for Out. For example, if you are using integers, specify the data type as a double integer.
Cannot divide by zero.	ENO is set according to ERROR_ENO, and Out contains the largest signed value allowed.	Define In2 (In3 for MDV instructions) to be a value other than 0.

5.10.5 Errors Caused by the Multiply Instruction

This error can occur when you are using the MUL instruction in a program. It is logged in the error log:

If this error occurs:	Then:	Do the following:
The result of the arithmetic calculation is too large for Out.	ENO is set according to ERROR_ENO, and Out contains the largest signed value allowed for its data type.	Specify a larger data type for Out. For example, if you are using integers, specify the data type as a double integer.

5.10.6 Errors Caused by the Negate Instruction

This error can occur when you are using the NEG instruction in a program. It is logged in the error log.

If this error occurs:	Then:	Do the following:
The result of the arithmetic calculation is too large for Out.	ENO is set according to ERROR_ENO, and Out contains the largest signed value allowed for its data type.	Make sure that the negated value of In falls within the allowable range for Out.

5.10.7 Errors Caused by the Square Root Instruction

These errors can occur when you are using the SQRT instruction in a program. They are logged in the error log.

If this error occurs:	Then:	Do the following:
The result of the arithmetic calculation is too large for Out.	ENO is set according to ERROR_ENO, and Out contains the largest positive value allowed for its data type.	Specify a larger data type for Out. For example, if you are using integers, specify the data type as a double integer.
Cannot take the square root of a negative number	ENO is set according to ERROR_ENO, and Out contains the square root of the absolute value of In.	Make sure the value of In is positive.

5.10.8 Errors Caused by the Subtract Instruction

This error can occur when you are using the SUB instruction in a program. It is logged in the error log.

If this error occurs:	Then:	Do the following:
The result of the arithmetic calculation is too large for Out.	ENO is set according to ERROR_ENO, and Out contains the largest signed value allowed for its data type.	Make sure that the result falls within the allowable range for Out.

6.0 Logical Instructions

Use logical instructions to perform logic operations on input parameters.

Choose from these logical instructions:

- Logical AND (AND)
- Logical NOT (NOT)
- Logical OR (OR)
- Logical Exclusive OR (XOR)

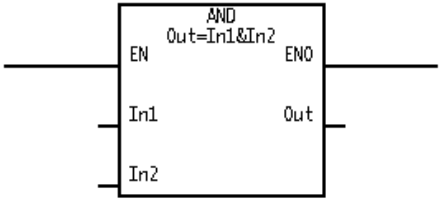
The supported parameters are:

- simple integers and double integers
- integer and double integer constants
- elements of integer and double integer arrays
- Timer variables (*name*.TPreset and *name*.Elapsed)
- Counter variables (*name*.CPreset and *name*.Current)

See each input and output parameter description for each instruction for specific information.

With the logical instructions, you can mix the data types for the inputs and outputs. When mixing integers and double integers within an instruction, the values are first converted to unsigned double integers. The result is then converted to the unsigned data type specified by the output.

6.1 Logical And (AND)



Use the Logical And instruction to perform a bit-wise logical AND between two variables. While EN is true, the instruction performs a logical AND operation on In1 and In2. The result is stored in Out.

The truth table for a bit-wise logical AND operation is as follows:

In1	In2	Out
0	0	0
1	0	0
0	1	0
1	1	1

6.1.1 Input Parameters for the Logical AND Instruction

This table lists the inputs for the AND instruction and the variable type and data type/range that each input supports.

Parameter	Description	Variable Type	Data Type/Range
EN	When this input is true, the instruction executes. When this input is false, the instruction is not executed and ENO is false.	Connect a Boolean input or output.	
In1 In2	Enter two variables or constants on which you want to perform the logical AND. Note: *Any constants entered are displayed in hexadecimal.*	• simple • constant • element of an array	integer (0-FFFF) double integer (0-FFFFFFFF)

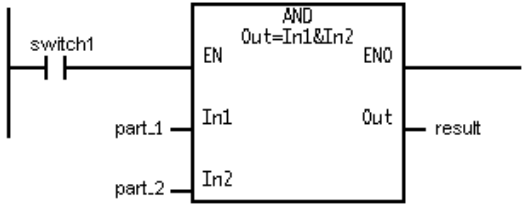
6.1.2 Output Parameters for the Logical AND Instruction

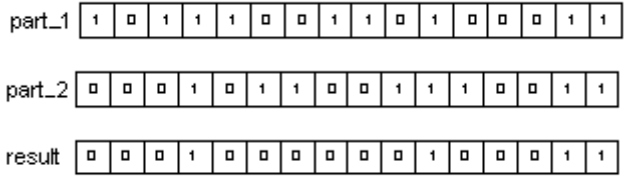
This table lists the outputs for the AND instruction and the variable type and data type/range that each output supports.

Parameter	Description	Variable Type	Data/Type Range
ENO	Use this output as the input to another instruction for easily chaining multiple instructions. This output follows the state of EN unless and error occurs.	Connect a contact, coil, or Boolean input to another instruction.	
Out	This output contains the result of the AND operation on In1 and In2.	• simple • element of an array	integer (0-FFFF) double integer (0-FFFFFFFF)

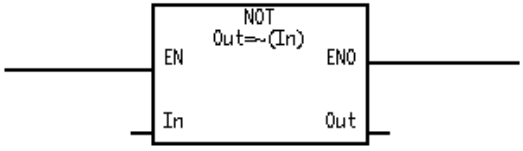
6.1.3 Example of a Logical AND Instruction

When *switch1* is true, the instruction performs a logical AND operation on the variables *part_1* and *part_2*, and Out contains the result in the variable *result*.





6.2 Logical Not (NOT)



Use the Logical Not instruction to change each bit of a single variable to the opposite value. While EN is true, the instruction performs a bit-wise logical NOT operation on In. The result is stored in Out.

The truth table for a bit-wise logical NOT operation is as follows:

In	Out
1	0
0	1

6.2.1 Input Parameters for the Logical NOT Instruction

This table lists the inputs for the NOT instruction and the variable type and data type/range that each input supports.

Parameter	Description	Variable Type	Data Type/Range
EN	When this input is true, the instruction executes. When this input is false, the instruction is not executed and ENO is false.	Connect a Boolean input or output.	
In	Enter the variable or constant on which you want to perform the logical NOT. Note: *Any constants entered are displayed in hexadecimal.*	• simple • constant • element of an array	integer (0-FFFF) double integer (0-FFFFFFFF)

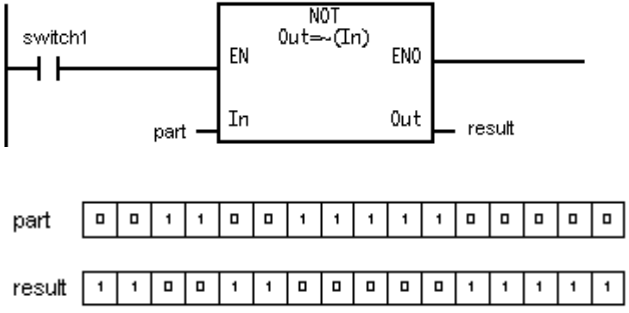
6.2.2 Output Parameters for the Logical NOT Instruction

This table lists the outputs for the NOT instruction and the variable type and data type/range that each output supports.

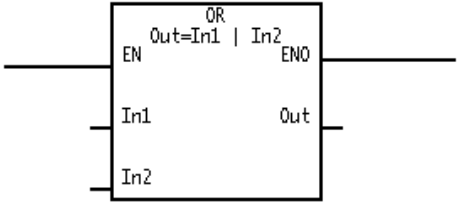
Parameter	Description	Variable Type	Data Type/Range
ENO	Use this output as the input to another instruction for easily chaining multiple instructions. This output follows the state of EN unless an error occurs.	Connect a contact, coil, or Boolean input to another instruction.	
Out	This output contains the result of the NOT operation on In.	• simple • element of an array	integer (0-FFFF) double integer (0-FFFFFFFF)

6.2.3 Example of a Logical NOT Instruction

While *switch1* is true, the instruction performs a logical NOT on the variable *part*, and Out contains the result in the variable *result*.



6.3 Logical Or (OR)



Use the Logical Or instruction to perform a bit-wise logical OR operation between two variables. While EN is true, the instruction performs a logical OR operation between In1 and In2. The result is stored in Out.

The truth table for a bit-wise logical OR operation is as follows:

In1	In2	Out
0	0	0
1	0	1
0	1	1
1	1	1

6.3.1 Input Parameters for the Logical OR Instruction

This table lists the inputs for the OR instruction and the variable type and data type/range that each input supports.

Parameter	Description	Variable Type	Data Type/Range
EN	When this input is true, the instruction executes. When this input is false, the instruction is not executed and ENO is false.	Connect a Boolean input or output.	
In1 In2	Enter two variables or constants on which you want to perform the logical OR. Note: *Any constants entered are displayed in hexadecimal.*	• simple • constant • element of an array	integer (0-FFFF) double integer (0-FFFFFFFF)

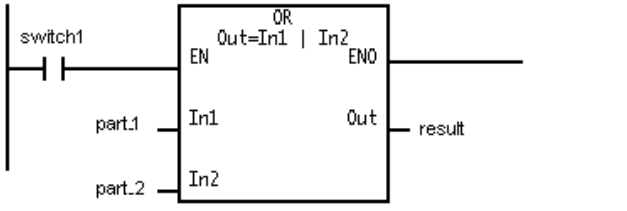
6.3.2 Output Parameters for the Logical OR Instruction

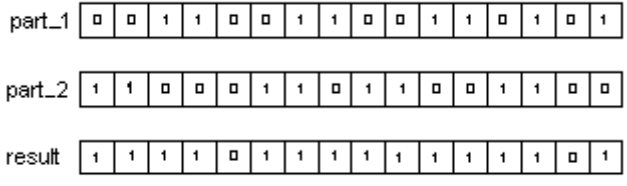
This table lists the outputs for the OR instruction and the variable type and data type/range that each output supports.

Parameter	Description	Variable Type	Data Type/Range
ENO	Use this output as the input to another instruction for easily chaining multiple instructions. This output follows the state of EN unless an error occurs.	Connect a contact, coil, or Boolean input to another instruction.	
Out	This output contains the result of the OR operation on In1 and In2.	• simple • element of an array	integer (0-FFFF) double integer (0-FFFFFFFF)

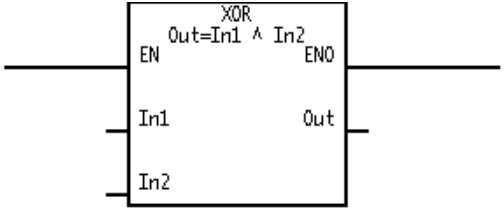
6.3.3 Example of a Logical OR Instruction

While the variable *switch1* is true, the instruction performs a logical OR on the variables *part_1* and *part_2,* and Out contains the result in the variable *result*.





6.4 Logical Exclusive Or (XOR)



Use the Logical Exclusive Or instruction to perform a bit-wise logical exclusive OR operation between two variables. While EN is true, the instruction performs a logical exclusive OR operation between In1 and In2. The result is stored in Out.

The truth table for a bit-wise logical XOR operation is as follows:

In1	In2	Out
0	0	0
1	0	1
0	1	1
1	1	0

6.4.1 Input Parameters for the Logical Exclusive OR Instruction

This table lists the inputs for the XOR instruction and the variable type and data type/range that each input supports.

Parameter	Description	Variable Type	Data Type/Range
EN	When this input is true, the instruction executes. When this input is false, the instruction is not executed and ENO is false.	Connect a Boolean input or output.	
In1 In2	Enter two variables or constants on which you want to perform the logical XOR. Note: *Any constants entered are displayed in hexadecimal.*	• simple • constant • element of an array	integer (0-FFFF) double integer (0-FFFFFFFF)

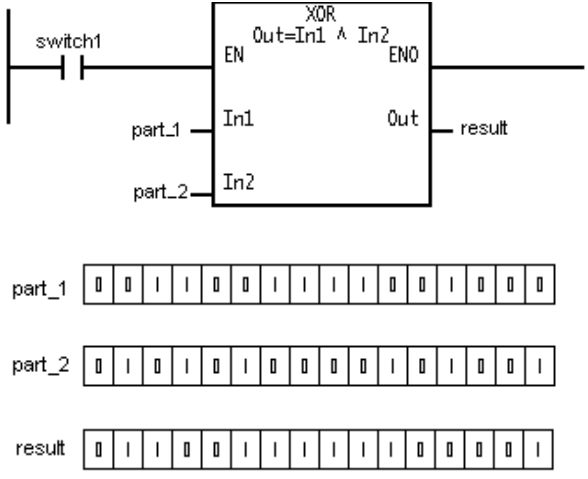
6.4.2 Output Parameters for the Logical Exclusive OR Instruction

This table lists the outputs for the XOR instruction and the variable type and data type/range that each output supports.

Parameter	Description	Variable Type	Data Type/Range
ENO	Use this output as the input to another instruction for easily chaining multiple instructions. This output follows the state of EN unless an error occurs.	Connect a contact, coil, or Boolean input to another instruction.	
Out	This output contains the result of the XOR operation on In1 and In2.	• simple • element of an array	integer (0-FFFF) double integer (0-FFFFFFFF)

6.4.3 Example of a Logical Exclusive OR Instruction

When the variable *switch1* is true, the instruction performs a logical exclusive OR operation on the variables *part_1* and *part_2*, and Out contains the result in the variable *result*.



6.5 Errors Caused by Logical Instructions

These errors can occur when you are using the logical instructions in a program. They are logged in the error log.

If this error occurs:	Then:	Do the following:
The array index is negative.	ENO is set to ERROR_ENO, and element zero of the array is used for the instruction's operation.	Specify a valid array element.
The array index is too large.	ENO is set to ERROR_ENO, and the last element of the array is used for the instruction's operation.	Specify a valid array element.

7.0 Data Conversion Instructions

Use data conversion instructions to convert data from integer to binary coded decimal and vice versa.

Choose from these instructions:

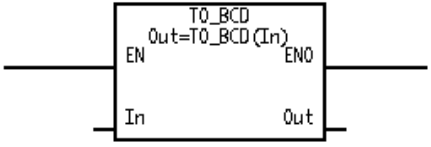
- TO_BCD
- BCD_TO

The supported parameters are:

- simple integer and double integer
- integer and double integer constants
- elements of integer and double integer arrays
- Timer variables (*name*.TPreset and *name*.Elapsed)
- Counter variables (*name*.CPreset and *name*.Current)

See each input and output parameter description for each instruction for specific information.

7.1 Convert Integer Data to BCD (TO_BCD)



Use this instruction to convert integer or double integer data to binary coded decimal (BCD) data.

While EN is true, the instruction converts the value of In to BCD. The result is stored in Out.

7.1.1 Input Parameters for the Convert Integer Data to BCD Instruction

This table lists the inputs for the TO_BCD instruction and the variable and type and data type/range that each input supports.

Parameter	Description	Variable Type	Data Type/Range
EN	While this input is true, the instruction executes. When this input is false, the instruction is not executed and ENO is false.	Connect a Boolean input or output.	
In	Enter the variable that you want converted to binary coded data.	• simple • constant • element of an array	• integer (0 to 9999) • double integer (0 to 99999999) • timer (*name*.TPreset and *name*.Elapsed) • counter (*name*.CPreset and *name*.Current)

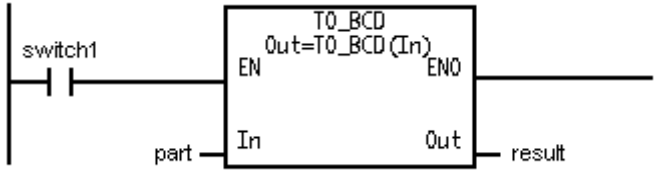
7.1.2 Output Parameters for the Convert Integer Data to BCD Instruction

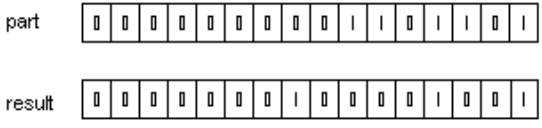
This table lists the outputs for the TO_BCD instruction and the variable type and data type/range that each output supports.

Parameter	Description	Variable Type	Data Type/Range
ENO	Use this output as the input to another instruction for easily chaining multiple instructions. This output follows the state of EN unless an error occurs.	Connect a contact, coil, or Boolean input of another instruction.	
Out	This output contains the binary coded decimal equivalent of the integer value of In.	• simple • element of an array	BCD value of 0-9999 (integer) or 0-99999999 (double integer)

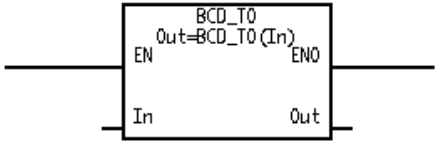
7.1.3 Example of a Convert Integer Data To BCD Instruction

When *switch1* is true, the instruction converts the integer value 109 in the variable *part* to binary coded decimal (BCD) and stores it in the variable *result*.





7.2 Convert From BCD to Integer Data (BCD_TO)



Use this instruction to convert binary coded decimal (BCD) data to integer or double integer data.

When EN is true, the instruction converts the BCD data of In to integer or double integer data. The result is stored in Out.

7.2.1 Input Parameters for the Convert From BCD to Integer Data Instruction

This table lists the inputs for the BCD_TO instruction and the data and type and data type/range that each input supports.

Parameter	Description	Variable Type	Data Type/Range
EN	While this input is true, the instruction executes. When this input is false, the instruction is not executed and ENO is false.	Connect a Boolean input or output.	
In	Enter the variable that you want converted to integer or double integer data.	• simple • constant • element of an array	BCD value of 0-9999 (integer) or 0-99999999 (double integer)

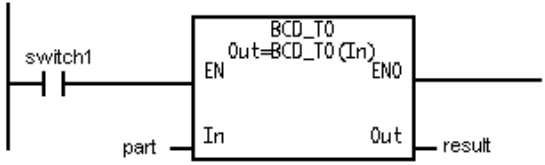
7.2.2 Output Parameters for the Convert From BCD to Integer Data Instruction

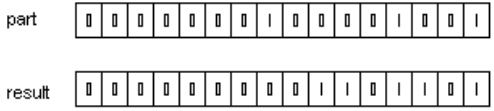
This table lists the outputs for the BCD_TO instruction and the variable type and data type/range that each output supports.

Parameter	Description	Variable Type	Data Type/Range
ENO	Use this output as the input to another instruction for easily chaining multiple instructions. This output follows the state of EN unless an error occurs.	Connect a contact, coil, or Boolean input of another instruction.	
Out	This output stores the integer or double integer equivalent of the BCD value of In.	• simple • element of an array	• converted integer or double integer value • timer (*name*.TPreset and *name*.Elapsed) • count (*name*.CPreset and *name*.Current)

7.2.3 Example of a Convert From Binary Data to Integer Data Instruction

When *switch1* becomes true, the instruction converts the BCD value 109 in the variable *part* to integer and stores the value in the variable *result*.





7.3 Errors Caused by the Data Conversion Instructions

This section describes the possible errors for all data conversion instructions and those additional errors specific to the TO_BCD and BCD_TO instructions.

7.3.1 Errors Caused by All Data Conversion Instructions

These errors can occur when you are using the data conversion instructions. They are logged in the error log.

If this error occurs:	Then:	Do the following:
The array index is negative.	ENO is set according to ERROR_ENO, and element zero of the array is used for the instruction's operation.	Specify a valid array element.
The array index is too large.	ENO is set according to ERROR_ENO, and the last element of the array is used for the instruction's operation.	Specify a valid array element.

7.3.2 Errors Caused by the Convert Integer Data To BCD Instruction

These errors can occur when you are using the TO_BCD instruction in a program. They are logged in the error log.

If this error occurs:	Then:	Do the following:
The result is larger than what Out's data type supports.	• Out contains the maximum value allowed for the data type (9999 for integers and 99999999 for double integers) • ENO is set according to ERROR_ENO	Make sure the value you want to convert to BCD is within the range Out supports.
Tried to convert a negative value to BCD.	• Out contains the value of 0 • ENO is set according to ERROR_ENO	Use a positive value for In.

7.3.3 Errors Caused by the Convert From Binary Data to Integer Data Instruction

These errors can occur when you are using the BCD_TO instruction in a program. They are logged in the error log.

If this error occurs:	Then:	Do the following:
The result is larger than what Out's data type supports.	ENO is set according to ERROR_ENO, and Out contains the largest value allowed for the data type being used.	Specify a larger data type for Out. For example, use a double integer data type instead of a integer.
An illegal BCD digit was found.	• Out contains the value of 0 • ENO is set according to ERROR_ENO.	Use a valid BCD value for In.

8.0 Move Instructions

Use the move instructions to copy data between variables.

Choose from these instructions:

Use this instruction:	To:
Move Source Data (MOVE)	move data from one variable to another
Move Bits Between Integers/Double Integers (MVB)	move a specified number of bits within the same variable or between variables
Masked Move (MVM)	move data from a variable, through a mask, and into another variable

The supported parameters are:

- simple integers and double integers
- integer and double integer constants
- elements of integer and double integer arrays
- Timer variables (*name*.TPreset and *name*.Elapsed)
- Counter variables (*name*.CPreset and *name*.Current)

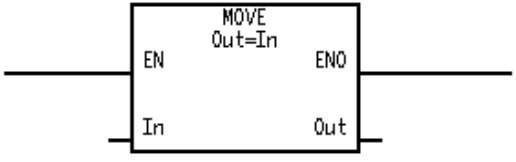
See the input and output parameter description for each instruction for specific information.

Defining a Mask

Choose to move the bits from In to Out by setting the bits in the mask that correspond to the location of the bits in the source (In).

To:	Define the corresponding Mask bit as:
move bits from In into Out	1
not move the bits from In to Out (the bits currently in Out do not change)	0

8.1 Move Source Data to Destination (MOVE)



Use the Move Source Data to Destination instruction to copy the value of the input variable or constant to the output variable. You can move data between variables that use different addressing modes.

When EN is true, the instruction copies the value of In to the variable assigned to Out.

8.1.1 Input Parameters for the Move Source Data to Destination Instruction

This table lists the inputs for the MOVE instruction and the variable type and data type/range that each input supports.

Parameter	Description	Variable Type	Data/Type Range
EN	While this input is true, the instruction executes. When this input is false, the instruction is not executed and ENO is false.	Connect a Boolean input or output.	
In	Enter a constant or whose value you want to copy to Out. In is the source of the move operation.	• simple • constant • element of an array	• integer • double integer • timer (*name*.TPreset and *name*.Elapsed) • counter (*name*.CPreset and *name*.Current)

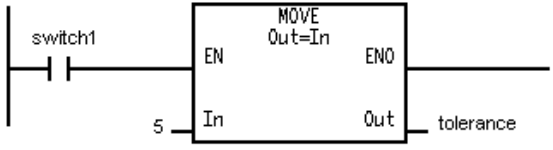
8.1.2 Output Parameters for the Move Source Data to Destination Instruction

This table lists the outputs for the MOVE instruction and the variable type and data type/range that each output supports.

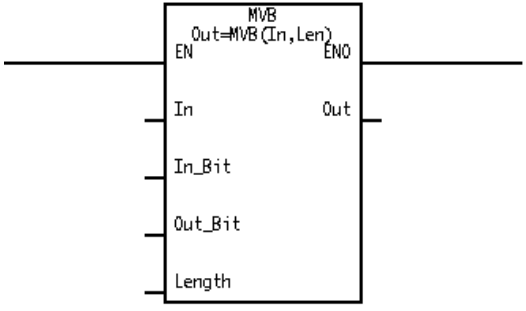
Parameter	Description	Variable Type	Data/Type Range
ENO	Use this output as the input to another instruction for easily chaining multiple instructions. This output follows the state of EN unless an error occurs.	Connect a contact, coil, or Boolean input to another instruction.	
Out	Enter the variable within which you want to store the value copied from In. Out is the destination of the move operation.	• simple • element of an array	• integer • double integer • timer (*name*.TPreeset and *name*.Elapsed) • counter (*name*.CPreset and *name*.Current)

8.1.3 Example of a Move Source Data to Destination Instruction

When *switch1* is true, the instruction moves the constant 5 into the variable *tolerance*.



8.2 Move Bits Between Integers/Double Integers (MVB)



Use the Move Bits Between Integers/Double Integers instruction to copy up to 16 or 32 bits of data within a variable or between two variables.

While EN is true, the instruction moves from In the number of bits specified in Length starting at the bit location specified in In_Bit. The bits are moved to Out, starting at the bit location specified in Out_Bit. The bits of the destination location are overwritten by those from In.

8.2.1 Input Parameters for the Move Bits Between Integers/Double Integers Instruction

This table lists the inputs for the MVB instruction and the variable type and date type/range that each input supports.

Parameter	Description	Variable Type	Data/Type Range
EN	While this input is true, the instruction executes. When this input is false, the instruction is not executed and ENO is false.	Connect a Boolean input or output.	
In	Enter a constant or variable from which you want to move bits, the source of the bit move operation.	• simple • constant • element of an array	• integer • double integer • timer (*name*.TPreset and *name*.Elapsed) • counter (*name*.CPreset and *name*.Current)
In_Bit	Enter the number of the bit within In from which the copying should start.		integer (0-31)
Out_Bit	Enter the number of the bit within Out that the instruction copies the bits to.		
Length	Enter the quantity of bits to be moved from In to Out. When defining a length, keep in mind the following: • Bits written to Out that extend beyond Out's data type boundary are lost. • Bits you copy from In that extend beyond bit 15 for integers and bit 31 for double integers are set to 0.		integer (0-32)

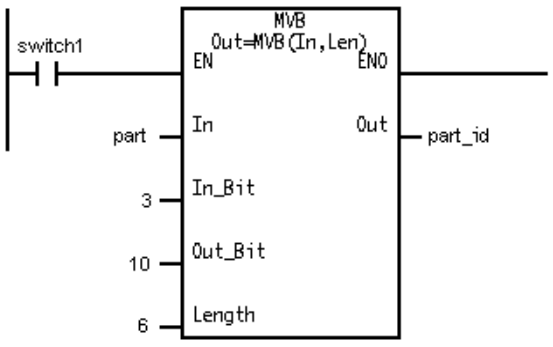
8.2.2 Output Parameters for the Move Bits Between Integers/Double Integers Instruction

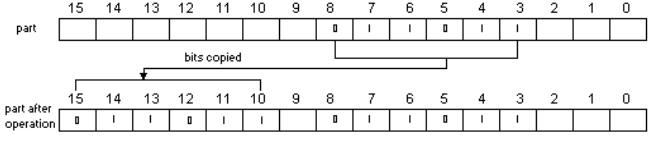
This table lists the outputs for the MVB instruction and the variable type and data type/range that each output supports.

Parameter	Description	Variable Type	Data/Type Range
ENO	Use this output as the input to another instruction for easily chaining multiple instructions. This output follows the state of EN unless an error occurs.	Connect a contact, coil, or Boolean input to another instruction.	
Out	Enter the variable within which to store the moved bits, the destination of the bit move operation. This can be the same variable as In. The bits within Out are overwritten by the bits from In.	• simple • element of an array	• integer • double integer • timer (*name*.TPreset and *name*.Elapsed) • counter (*name*.CPreset and *name*.Current)

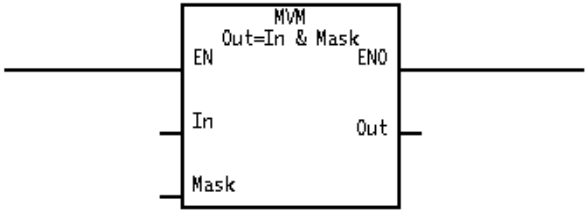
8.2.3 Examples of a Move Bits Between Integers/Double Integers Instruction

When *switch1* is true, the instruction copies six bits from the variable *part* starting at bit 3 to the variable *part_id* starting at bit 10.





8.3 Masked Move (MVM)



Use the Masked Move instruction to copy portions of a variable through a mask and into an output variable. You can use this instruction to extract data from a variable.

When EN becomes true, the instruction copies In through a defined mask and into the variable assigned to Out.

8.3.1 Input Parameters for the Masked Move Instruction

This table lists the inputs for the MVM instruction and the variable type and data type/range that each input supports.

Parameter	Description	Variable Type	Data/Type Range
EN	While this input is true, the instruction executes. When this input is false, the instruction is not executed and ENO is false.	Connect a Boolean input or output.	
In	Enter a constant or variable that you want to copy to Out. In is the source of the masked move operation.	• simple • constant • element of an array	• integer • double integer • timer (*name*.TPreset and *name*.Elapsed) • counter (*name*.CPreset and *name*.Current)
Mask	Enter a variable or hexadecimal constant that specifies which bits to pass or block. A bit set as 1 in the mask passes the source bit into the destination. Whereas, a bit set as 0 blocks the source bit from being copied to the destination. See “Defining a Mask” (section 8.0). Note: *Any constants entered are displayed in hexadecimal.*	• simple • element of an array • hexadecimal constant	• integer • double integer • timer (*name*.TPreset and *name*.Elapsed) • counter (*name*.CPreset and *name*.Current)

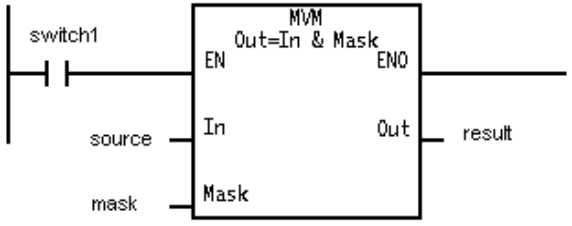
8.3.2 Output Parameters for the Masked Move Instruction

This table lists the outputs for the MVM instruction and the variable type and data type/range that each output supports.

Parameter	Description	Variable Type	Data/Type Range
ENO	Use this output as the input to another instruction for easily chaining multiple instructions. This output follows the state of EN unless an error occurs.	Connect a contact, coil, or Boolean input to another instruction.	
Out	Enter the variable within which you want to store the value copied from In. Out is the destination of the masked move operation.	• simple • element of an array	• integer • double integer • timer (*name*.TPreset and *name*.Elapsed) • counter (*name*.CPreset and *name*.Current)

8.3.3 Example of a Masked Move Instruction

When *switch1* is true, the instruction moves the data from the variable *source* through the mask as defined by the variable *mask* and stores the data in the variable *result*.



8.4 Errors Caused by Move Instructions

This section describes the possible errors for all Move instructions and those additional errors specific to the MOVE and MVB instructions.

8.4.1 Errors Caused by All Move Instructions

These errors can occur whenyou use any move instruction. They are loged in the error log.

If this error occurs:	Then:	Do the following:
The array index is negative.	ENO is set according to ERROR_ENO, and element zero of the array is used for the instruction's operation.	Specify a valid array element.
The array index is too large.	ENO is set according to ERROR_ENO, and the last element of the array is used for the instruction's operation.	Specify a valid array element.

8.4.2 Errors Caused by the Move Source Data to Destination Instruction

This error can occur when you are using the MOVE instruction in a program. It is logged in the error log.

If this error occurs:	Then:	Do the following:
The result is larger that what Out's data type supports.	ENO is set according to ERROR_ENO, and Out contains the largest value allowed for its data type.	Define the variable in Out to be a double integer.

8.4.3 Errors Caused by the Move Bits Between Integers/Double Integers Instruction

This error can occur when you are using the MVB instruction in a program. It is logged in the error log.

If this error occurs:	Then:	Do the following:
• The bit number is negative • The bit number is too large. • The Length input is negative • The Length input is greater than 32.	ENO is set according to ERROR_ENO, and Out will not be written to.	Specify a value within the appropriate range for the input in error.

9.0 Shift Register Instructions

Use the Shift Register instructions to store, extract, and manipulate binary data.

The Shift Register instructions can help you:

- track parts or product
- record bar-code information
- control machinery or processes
- track and record system diagnostic information

Choose from these instructions:

Use this instruction:	**To:**
Shift Left (SL)	Shift out to the left the most significant bit and load either a 1 or 0 into the least significant bit position.
Circular Rotate Bits Left (ROL)	Rotate a specified number of the most significant bits into the least significant bit positions while the instruction is enabled.
Circular Rotate Bits Left on Transition (RL)	Rotate a specified number of the most significant bits into the least significant bit positions for each false-true transition of the enable bit. This instruction performs the same operation as ROL, but it is edge-triggered.
Shift Right (SR)	Shift out to the right the least significant bit and load either a 1 or 0 into the most significant bit position.

Use this instruction:	To:
Circular Rotate Bits Right (ROR)	Rotate a specified number of the least significant bits into the most significant bit positions while the instruction is enabled.
Circular Rotate Bits Right on Transition (RR)	Rotate a specified number of the least significant bits into the most significant bit positions for each false-true transition of the enable bit. This instruction performs the same operation as ROR, but it is edge-triggered.

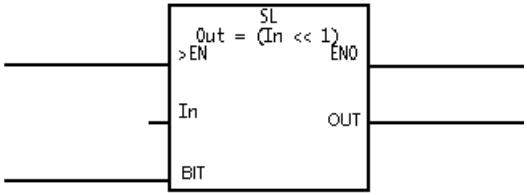
The supported parameters are:

- Boolean arrays
- simple integer and double integer
- element-indexed integer and double integer arrays

Rotating Bits Within Boolean Arrays

In a Boolean array, you cannot rotate more bits than the array contains. For example, if a Boolean array has 5 elements, you cannot rotate 6 elements. The maximum number of bits that can be rotated in one instruction is 8 bits.

9.1 Shift Left (SL)



Use this instruction to shift all the bits in the variable to the left and load either a 1 or 0 into the least significant bit position for each false-true transition of the enable input.

When EN transitions from false to true, the bits of In are shifted one position to the left. The most significant bit is moved to OUT, and the value of BIT is shifted into the least significant bit position of In.

9.1.1 Input Parameters for the Shift Left Instruction

This table lists the inputs for the SL instruction and the variable type and data type/range that each input supports.

Parameter	Description	Variable Type	Data/Type Range
EN	While this input is true, the instruction executes. When this input is false, the instruction is not executed and ENO is false.	Connect a Boolean input or output.	
In	Enter the name of the variable containing the bits you want to shift.	• Boolean array • simple integer or double integer • element of an integer or double integer array	• 1 or 0 • integer (0-FFFF) • double integer (0-FFFFFFFF)
BIT	This input provides the state of the least significant bit after the shift occurs. To place a 1 in the least significant bit location, the Boolean parameter connected to BIT must be true. To place a 0 in the least significant bit location, the Boolean parameter connected to BIT must be false. If no Boolean parameter is programmed, a 0 is loaded into the least significant bit location.	Connect a Boolean input or output.	

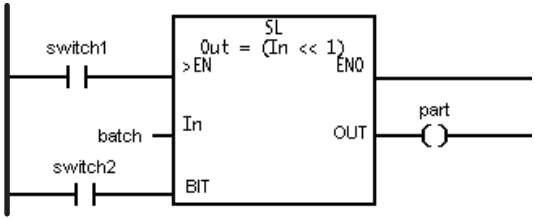
9.1.2 Output Parameters for the Shift Left Instruction

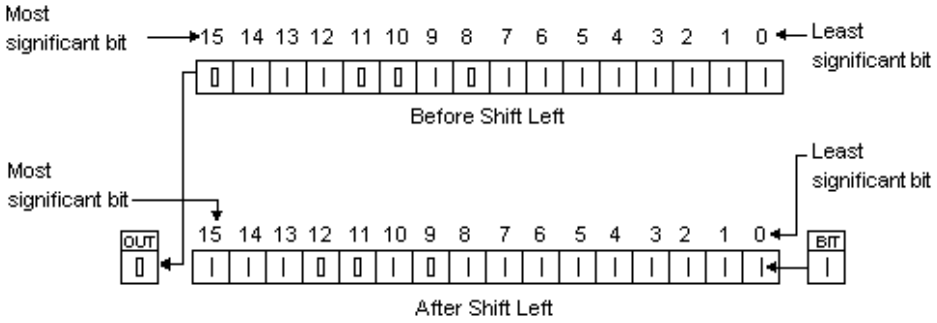
This table lists the outputs for the SL instruction and the variable type and data type/range that each output supports. To use them, connect them to a contact, coil, or Boolean input of another instruction.

Parameter	Description
ENO	Use this output as the input to another instruction for easily chaining multiple instructions. This output follows the state of EN unless an error occurs.
OUT	This output contains the value of most significant bit that is shifted out of the variable specified for In. This value is retained when EN is set false.

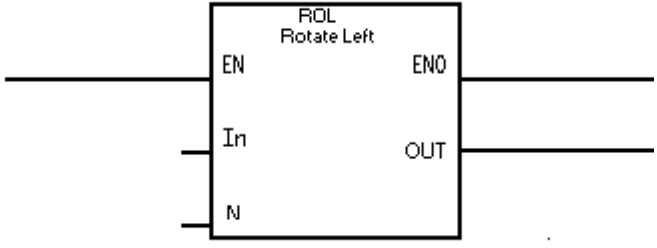
9.1.3 Example of a Shift Left Instruction

When *switch1* transitions from false to true, the bits shift left, moving the original value of bit 15 into OUT. Then, the state of *switch2* is read, and that value is shifted into bit 0 of the variable *batch*. In this example, *switch2* is true, so a value of 1 is shifted into bit 0.





9.2 Circular Rotate Bits Left (ROL)



Use this instruction to rotate all the bits within a variable a specified number of bit positions to the left. The bits are rotated out of the most significant bit positions and are rotated back into the least significant bit positions.

While EN is true, the following occurs:

- the number of the most significant bits of In specified in N are moved from In
- the remaining bits shift left
- the bits rotated from the most significant bits of In are filled into the least significant bit positions of In.

The specified number of bits are rotated left every program scan while the instruction is true.

9.2.1 Input Parameters for the Circular Rotate Bits Left Instruction

This table lists the inputs for the ROL instruction and the variable type and data type/range that each input supports.

Parameter	Description	Variable Type	Data/Type Range
EN	While this input is true, the instruction executes. When this input is false, the instruction is not executed and ENO is false.	Connect a Boolean input or output.	
In	Enter the name of the variable containing the bits you want to shift.	• Boolean array • simple integer or double integer • element of an integer or double integer array	• 1 or 0 • integer (0-FFFF) • double integer (0-FFFFFFFF)
N	Enter the number of bit positions you want to rotate left. If you do not want to rotate any bits, enter 0.	• integer or double integer constant • simple integer or double integer • element of an integer or double integer array	• 0 to 8 (if In is a Boolean array) • 0 to 15 (if In is an integer) • 0 to 31 (if In is a double integer, including timer or counter elements)

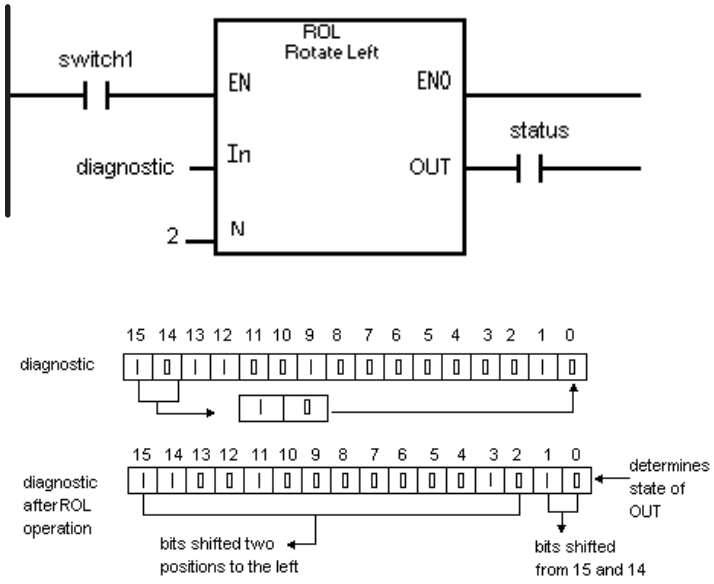
9.2.2 Output Parameters for the Circular Rotate Bits Left Instruction

This table lists the outputs for the ROL instruction and the variable type and data type/range that each output supports. To use them, connect them to a contact, coil, or Boolean input of another instruction.

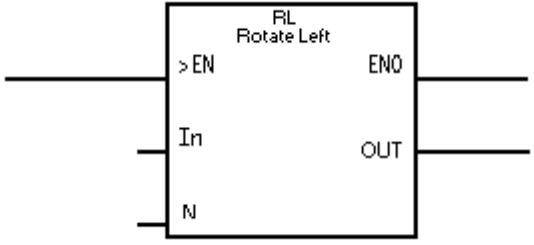
Parameter	Description
ENO	Use this output as the input to another instruction for easily chaining multiple instructions. This output follows the state of EN unless an error occurs.
OUT	This output reflects the state of the least significant bit of In following the rotate operation.

9.2.3 Example of a Circular Rotate Bits Left Instruction

While *switch1* is true, the two most significant bits (bits 15 and 14) of the integer variable *diagnostic* are shifted out, the remaining bits are shifted left, and the former most significant bits (bits 15 and 14) are placed into bits 1 and 0. The state of the variable *status* reflects the value in bit 0; in this case, the variable *status* is false.



9.3 Circular Rotate Bits Left on Transition (RL)



Use this instruction to rotate all the bits within a variable a specified number of bit positions to the left on a false-true transition. The bits are rotated left from the most significant bit positions and into the least significant bit positions.

When EN transitions from false to true, the following occurs:

- the number of the most significant bits of In specified in N are moved from In
- the remaining bits shift left
- the bits rotated from the most significant bits of In are filled into the least significant bit positions of In

The specified number of bits are rotated left only when EN transitions from false to true.

9.3.1 Input Parameters for the Circular Rotate Bits Left on Transition Instruction

This table lists the inputs for the RL instruction and the variable type and data type/range that each input supports.

Parameter	Description	Variable Type	Data/Type Range
EN	While this input is true, the instruction executes. When this input is false, the instruction is not executed and ENO is false.	Connect a Boolean input or output.	
In	Enter the name of the variable containing the bits you want to shift.	• Boolean array • simple integer or double integer • element of an integer or double integer array	• 1 or 0 • integer (0-FFFF) • double integer (0-FFFFFFFF)
N	Enter the number of bit positions you want to rotate left. If you do not want to rotate any bits, enter 0.	• integer or double integer constant • simple integer or double integer • element of an integer or double integer array	• 0 to 8 (if In is a Boolean array) • 0 to 15 (if In is an integer) • 0 to 31 (if In is a double integer, including timer or counter elements)

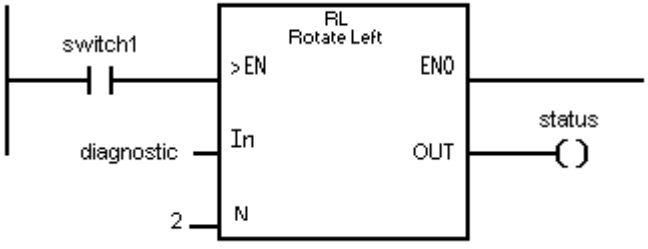
9.3.2 Output Parameters for the Circular Rotate Bits Left On Transition Instruction

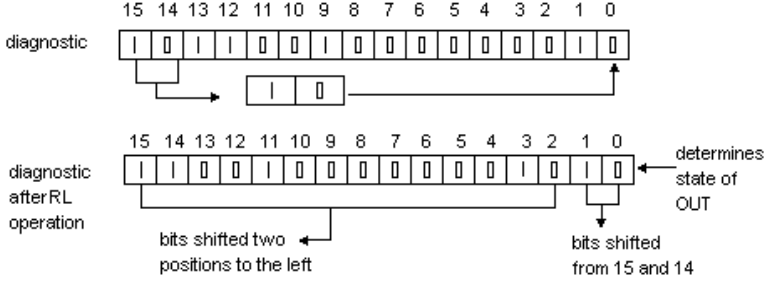
This table lists the outputs for the RL instruction and the variable type and data type/range that each output supports. To use them, connect them to a contact, coil, or Boolean input of another instruction.

Parameter	Description
ENO	Use this output as the input to another instruction for easily chaining multiple instructions. This output follows the state of EN unless an error occurs.
OUT	This output reflects the state of the least significant bit of In following the rotate operation. This value is retained when EN is set false.

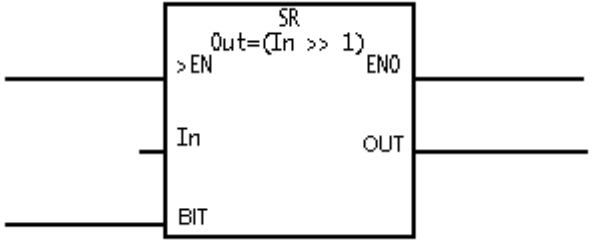
9.3.3 Example of a Circular Rotate Bits Left On Transition Instruction

When *switch1* transitions from false to true, the two most significant bits (bits 15 and 14) of the integer variable *diagnostic* are shifted out, the remaining bits are shifted left, and the former most significant bits (bits 15 and 14) are placed into bits 1 and 0. The state of the variable *status* reflects the value in bit 0. In this case, the variable *status* is false.





9.4 Shift Right (SR)



Use this instruction to shift all the bits in a variable to the right and load either a 1 or 0 into the most significant bit position for each false-true transition of the enable input.

When EN transitions from false to true, the bits of In are shifted one position to the right. The value of the least significant bit is moved to OUT, and the value of BIT is shifted into the most significant bit position of In.

9.4.1 Input Parameters for the Shift Right Instruction

This table lists the inputs for the SR instruction and the variable type and data type/range that each input supports.

Parameter	Description	Variable Type	Data/Type Range
EN	While this input is true, the instruction executes. When this input is false, the instruction is not executed and ENO is false.	Connect a Boolean input or output.	
In	Enter the name of the variable containing the bits you want to shift.	• Boolean array • simple integer or double integer • element of an integer or double integer array	• 1 or 0 • integer (0-FFFF) • double integer (0-FFFFFFFF)
BIT	This input provides the state of the most significant bit after the shift occurs. To place a 1 in the most significant bit location, the Boolean parameter connected to BIT must be true. To place a 0 in the most significant bit location, the Boolean parameter connected to BIT must be false. If no Boolean parameter is programmed, a 0 is loaded into the most significant bit location.	Connect a Boolean input or output.	

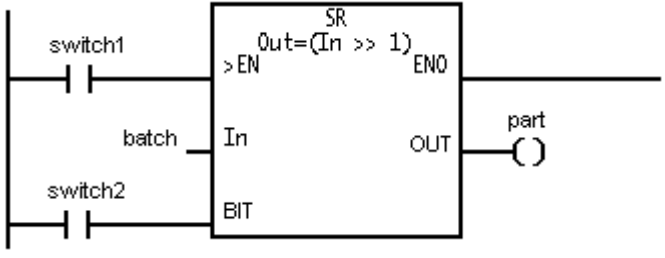
9.4.2 Output Parameters for the Shift Right Instruction

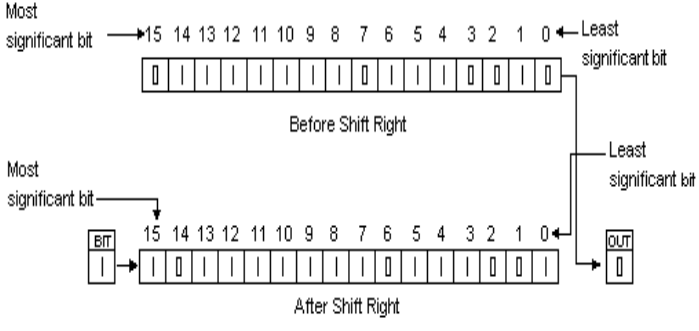
This table lists the outputs for the SR instruction and the variable type and data type/range that each output supports. To use them, connect them to a contact, coil, or Boolean input of another instruction.

Parameter	Description
ENO	Use this output as the input to another instruction for easily chaining multiple instructions. This output follows the state of EN unless an error occurs.
OUT	This output contains the value of the least significant bit that is shifted out of the variable specified for In. This value is retained when EN is set false.

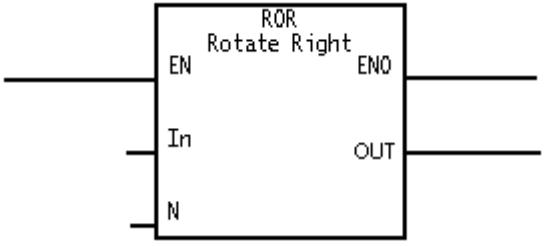
9.4.3 Example of a Shift Right Instruction

When *switch1* transitions from false to true, the bits shift right, moving the original value of bit 0 into OUT. Then, the state of *switch2* is read, and that value is shifted into bit 15 of the variable *batch*. In this example, *switch2* is true, so 1 is shifted into bit 15.





9.5 Circular Rotate Bits Right (ROR)



Use this instruction to rotate all the bits within a variable a specified number of bit positions to the right. The bits that are rotated out of the least significant bit positions are rotated back into the most significant bit positions.

While EN is true, the following occurs:

- the number of least significant bits of In specified in N are moved from In
- the remaining bits shift right
- the bits rotated from the least significant bits of In are filled into the most significant bit positions of In

The specified number of bits are rotated right every program scan while the instruction is true.

9.5.1 Input Parameters for the Circular Rotate Bits Right Instruction

This table lists the inputs for the ROR instruction and the variable type and data type/range that each input supports.

Parameter	Description	Variable Type	Data/Type Range
EN	While this input is true, the instruction executes. When this input is false, the instruction is not executed and ENO is false.	Connect a Boolean input or output.	
In	Enter the name of the variable containing the bits you want to shift.	• Boolean array • simple integer or double integer • element of an integer or double integer array	• 1 or 0 • integer (0-FFFF) • double integer (0-FFFFFFFF)
N	Enter the number of bit positions you want to rotate right. If you do not want to rotate any bits, enter 0.	• integer or double integer constant • simple integer or double integer • element of an integer or double integer array	• 0 to 8 (if In is a Boolean array) • 0 to 15 (if In is an integer) • 0 to 31 (if In is a double integer, including timer or counter elements)

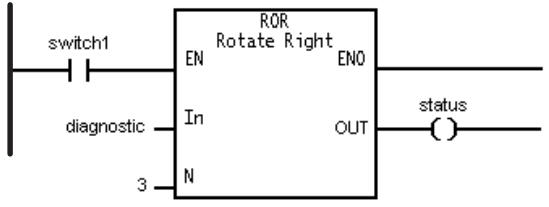
9.5.2 Output Parameters for the Circular Rotate Bits Right Instruction

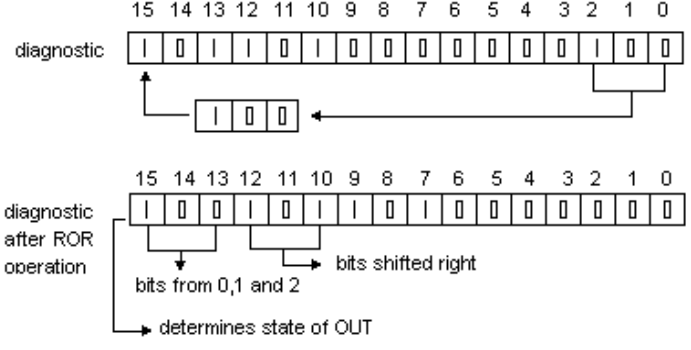
This table lists the outputs for the ROR instruction and the variable type and data type/range that each output supports. To use them, connect them to a contact, coil, or Boolean input of another instruction.

Parameter	Description
ENO	Use this output as the input to another instruction for easily chaining multiple instructions. This output follows the state of EN unless an error occurs.
OUT	This output reflects the state of the most significant bit of In following the rotate operation.

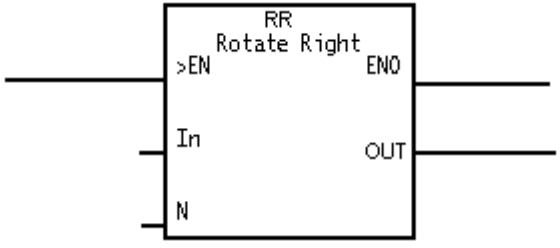
9.5.3 Example of a Circular Rotate Bits Right Instruction

While *switch1* is true, the three least-significant bits (bits 0, 1, and 2) of the variable *diagnostic* are shifted out. The remaining bits are shifted right, and the former least significant bits (bits 0, 1, and 2) of the integer variable *diagnostic* are placed into the three most-significant bits (bits 15, 14, and 13, respectively). The state of the variable *status* is true.





9.6 Circular Rotate Bits Right on Transition (RR)



Use this instruction to rotate all the bits within a variable a specified number of bit positions to the right. The bits that are rotated out of the least significant bit positions are rotated back into the most significant bit positions.

When EN transitions from false to true, the following occurs:

- the number of least significant bits of In specified in N are moved from In
- the remaining bits shift right
- the rotated bits from the least significant bits of In are filled into the most significant bit positions of In

The specified number of bits are rotated right only when EN transitions from false to true.

9.6.1 Input Parameters for the Circular Rotate Bits Right on Transition Instruction

This table lists the inputs for the RR instruction and the variable type and data type/range that each input supports.

Parameter	Description	Variable Type	Data/Type Range
EN	While this input is true, the instruction executes. When this input is false, the instruction is not executed and ENO is false.	Connect a Boolean input or output.	
In	Enter the name of the variable containing the bits you want to shift.	• Boolean array • simple integer or double integer • element of an integer or double integer array	• 1 or 0 • integer (0-FFFF) • double integer (0-FFFFFFFF)
N	Enter the number of bit positions you want to rotate right. If you do not want to rotate any bits, enter 0.	• integer or double integer constant • simple integer or double integer • element of an integer or double integer array	• 0 to 8 (if In is a Boolean array) • 0 to 15 (if In is an integer) • 0 to 31 (if In is a double integer, including timer or counter elements)

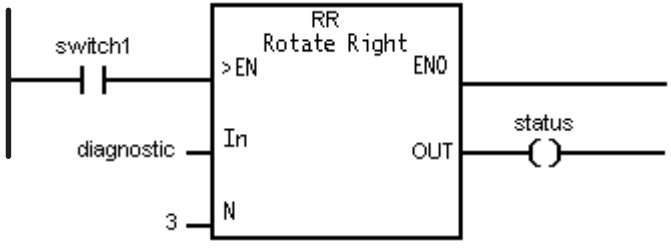
9.6.2 Output Parameters for the Circular Rotate Bits Right On Transition Instruction

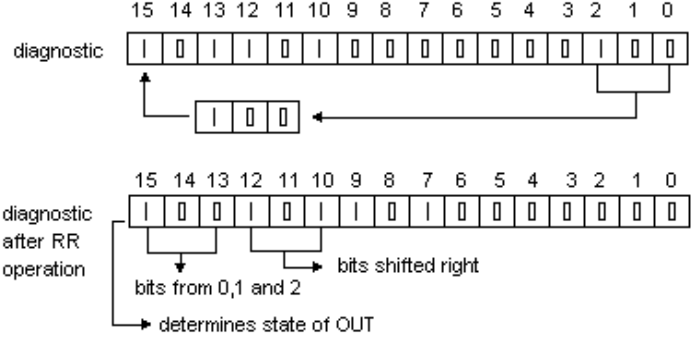
This table lists the outputs for the RR instruction and the variable type and data type/range that each output supports. To use them, connect them to a contact, coil, or Boolean input of another instruction.

Parameter	Description
ENO	Use this output as the input to another instruction for easily chaining multiple instructions. This output follows the state of EN unless an error occurs.
OUT	This output reflects the state of the most significant bit of In following the rotate operation.

9.6.3 Example of a Circular Rotate Bits Right On Transition Instruction

While *switch1* is true, the three least-significant bits (bits 0, 1, and 2) of the variable *diagnostic* are shifted out, the remaining bits are shifted right, and the former least significant bits (bits 0, 1, and 2) of the integer variable *diagnostic* are placed into the three most-significant bits (bits 15, 14, and 13, respectively). The state of the variable *status* is true.





9.7 Errors Caused by Shift Register Instructions

This section describes the possible errors for all Shift Register instructions and those additional errors specific to the ROL, RL, ROR, and RR instructions.

9.7.1 Errors Caused by All Shift Register Instructions

These errors can occur when you are using the Shift Register instructions in a program. They are logged in the error log.

If this error occurs:	Then:	Do the following:
Array index is negative.	ENO is set according to ERROR_ENO, and element zero of the array is used for the instruction's operation.	Specify a valid array element.
Array index is too large.	ENO is set according to ERROR_ENO, and the last element in the array is used for the instruction's operation.	Specify a valid array element.

9.7.2 Errors Caused by the Circular Rotate Bits Left, Circular Rotate Bits Left on Transition, Circular Rotate Bits Right, and Circular Rotate Bits Right on Transition Instructions

This error can occur when you are using the ROL, RL, ROR, and RR instructions. It is logged in the error log.

If this error occurs:	Then:	Do the following:
• The number of bits to rotate is negative. • The number of bits to rotate is too large.	• No bits are rotated • ENO is set to ERROR_ENO • Out reflects least significant bit of In	Make sure that N is within the allowable range for the data type used for In.

10.0 Array Instructions

The array instructions perform functions on array variables that are similar to the functions of compute, compare, logical, shift, and move instructions.

Using array instructions, you can mix simple variable inputs and outputs with array inputs and outputs for flexible data manipulation.

Choose from these instructions:

To perform these operations:	Choose this instruction:
Logical, arithmetic, and move operations on one operand: • absolute value • logical NOT • square root • negate • move	Unary Array (AR1)
Logical and arithmetic operations on two operands: • logical AND • logical OR • logical exclusive OR • addition • subtraction • multiplication • division	Multi-Array (AR2)

To perform these operations:	Choose this instruction:
Shift array elements up	Array Shift Up (ASU)
Shift array elements down	Array Shift Down (ASD)
Compare operations on two operands: • equal to • not equal to • greater than • greater than or equal to • less than • less than or equal to	Array Compare (ARC)

The supported parameters are:

- integer and double integer
- elements of integer and double integer arrays
- integer and double integer constants
- simple integer and double integer variables

Both the input variable and the output variable must be the same data type.

Overview of How the AR1, AR2, and ARC Instructions Operate

A single false-true transition of the EN input starts the array instruction. You can program the number of elements on which to operate per program scan. Choosing all the elements or a portion of them affect the program scan in different ways.

Determining the Number of Elements on Which To Operate

When you are using an array instruction you must specify how many elements to operate on per program scan. For example, if you want to move 10 elements of an array to another array, you can move all 10 elements in one program scan or you can move them 2 elements at a time over 5 program scans.

Having an instruction operate on a large number of elements per program scan can significantly increase the time required for the program scan to complete. You can reduce the impact of executing large array operations on a program scan by choosing to perform the array operation in more than one program scan. This breaks up the large execution time and spreads the operation over multiple scans.

To break up an array operation over multiple program scans, choose a portion of the total number of elements to operate on per program scan. Then, during every program scan in which the instruction is still true, the instruction operates on a portion of the elements, advancing through the total number (Length) of elements.

For example, if you must operate on the 20th through the 29th element within an array (10 elements), you can choose to only operate on 5 elements per program scan. The sequence of events would be:

- Program Scan 1: The instruction prepares this array operation.
- Program Scan 2: Elements 20 through 24 are operated on.
- Program Scan 3: Elements 25 through 29 are operated on.

However, some disadvantages of breaking up an array operation exist. They are:

- The array instruction's operation is not completed (DN is true) for several program scans. This could impact the timeliness of other logic execution that requires the result of an array operation.
- The actual array operation begins on the **second** program scan. The instruction uses the first scan to set up the instruction. Whereas, if all the array elements are to be operated on in one scan, the instruction sets up and performs the operation in one scan.

How Array Instructions Execute

The array instructions execute following this sequence of events:

- The array instructions execute when the enable bit for the instruction first transitions from false to true. If the enable bit is set false before the array operation is completed, the operation is canceled.
- The instruction's input and output control parameters are latched.

  Array variables are **not** latched. Therefore, be aware that the values may be changed if another program is writing into some or all of the array elements. This also can occur when the array operation is to complete in a single scan and a higher priority program or a program from another Processor is writing data to the array.

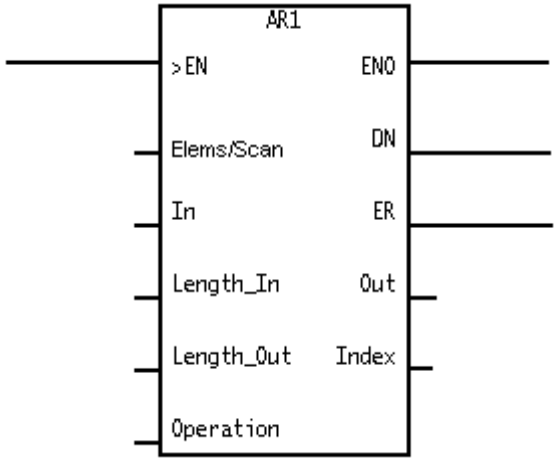
  However, if a single array element or variable (length=1) is used as a parameter, it is latched to ensure that the value remains fixed even though the operation may occur over multiple scans.
- Once the control variables are latched and limit checking on the data is complete, the instruction performs the programmed array operation.

  The instruction immediately performs the operation when you specify a 0 for the Elems/Scan input. If this input is not 0, the actual array operation begins on the **second** scan.
- When the instruction finishes operating on the last element within the array, the done (DN) bit becomes true.

Tip

If the instruction's operation is distributed among two or more program scans, you can determine how far into the array operation that the instruction is by monitoring Index. The Index output parameter indicates the last element that was operated on during the last scan.

10.1 Unary Array Instruction (AR1)



Use this instruction to perform one of these operations on a single operand:

- calculate the absolute value (ABS)
- perform a logical NOT (NOT)
- calculate the square root (SQRT)
- perform a negate operation (NEG)
- move from a source location to a destination location (MOVE)

Use this instruction to perform arithmetic or move operations on:

- simple variables
- a single element of an array
- multiple elements of an array

The In input can be an array variable or a simple variable/constant. You can place the result in a simple or array variable.

When EN transitions from false to true, the instruction prepares to carry out the function specified in Operation by latching (buffering) all the parameters. If you have programmed the array instruction to operate on the array elements over multiple program scans, the first scan sets up the operation. Then, the operation is performed over a series of scans until done.

As long as EN remains true, the instruction completes the operation by operating on the number of elements specified in Elems/Scan with each program scan.

A specified number of elements (Length_Out) is stored in Out. Index acts as a marker indicating the last element of the array that was operated on during the last scan.

10.1.1 Input Parameters for the Unary Array Instruction

This table lists the inputs for the AR1 instruction and the variable type and data type/range that each input supports.

Parameter	Description	Variable Type	Data/Type Range
EN	While this input is true, the instruction executes. When this input is false, the instruction does not execute and is reset.	Connect a Boolean input or output.	
Elems/ Scan	Enter the number of elements to operate on each time the instruction is scanned. To operate on all elements during a single scan, enter 0. To distribute the array operation over multiple program scans, enter the number of elements you want to operate on each time the instruction is scanned (after the first scan).	• simple • constant • element of an array	• integer or double integer • timer (*name*.TPreset and *name*.Elapsed) • counter (*name*.CPreset and *name*.Current) The range is 0 through 100,000 and must not exceed the maximum array index plus 1.

Parameter	Description	Variable Type	Data/Type Range
In	Enter the name of the variable or enter a constant value that is to be operated on.	• constant • array • simple • element of an array	• integer • double integer • timer (*name*.TPreset and *name*.Elapsed) • counter (*name*.CPreset and *name.*Current)
Length_In	Enter the number of elements of In that this instruction is to operate on. If In is a simple variable or a constant, enter 1. If In is an array variable, enter a number 1 or larger.	• simple • constant • element of an array	• integer or double integer • timer (*name*.TPreset and *name*.Elapsed) • counter (*name*.CPreset and *name*.Current) The range is 0 through 100,000 and must not exceed the maximum array index plus 1.

Parameter	Description	Variable Type	Data/Type Range
Length_ Out	Enter the number of elements to store in Out. If Out is a simple variable, enter 1. If Out is an array variable, enter a number 1 or larger. When the values in Length_In and Length_Out are greater than 1, they must be equal.	• simple • constant • element of an array	• integer or double integer • timer (*name*.TPreset and *name*.Elapsed) • counter (*name*.CPreset and *name*.Current) The range is 0 through 100,000 and must not exceed the maximum array index plus 1.
Operation	Enter the mnemonic of the operation you want to perform on In. Choose from these operations: • ABS (absolute value) • MOVE (move) • NEG (negate) • NOT (logical NOT) • SQRT (square root)	Character string	A-Z

10.1.2 Output Parameters for the Unary Array Instruction

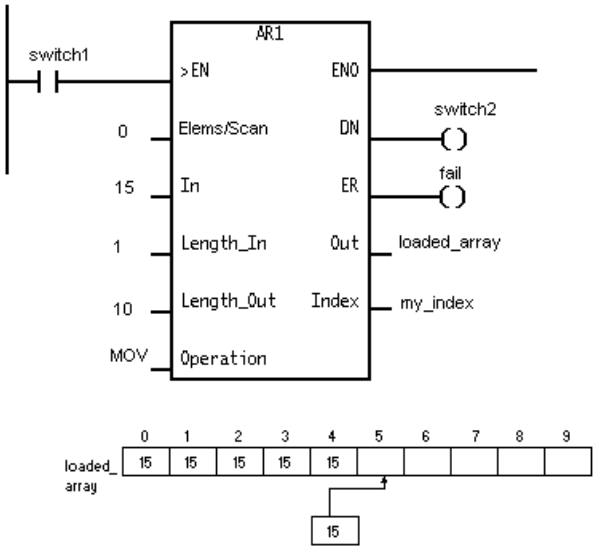
This table lists the outputs for the AR1 instruction and the variable type and data type/range that each output supports.

Parameter	Description	Variable Type	Data/Type Range
ENO	Use this output as the input to another instruction for easily chaining multiple instructions. This output follows the state of EN unless an error occurs.	Connect a contact, coil or Boolean parameter of another instruction.	
DN	This output is true when the instruction has operated on the last array element or an error was found during the first scan. DN is false when EN is false.		
ER	This output is set true after the instruction is completed when an error was encountered during its operation. This output is set false when EN is false.		
Out	Enter the name of a variable in which you want to store the result of the Operation. The data type for this variable must be the same data type as that for In.	• array • simple • element of an array	• integer • double integer • timer (*name*.TPreset and *name*.Elapsed) • counter (*name*.CPreset and *name*.Current)

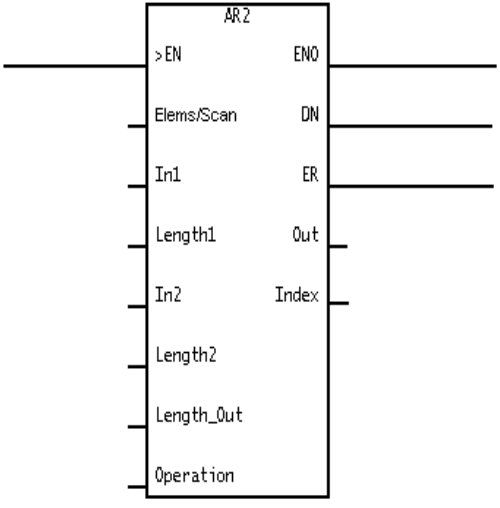
Parameter	Description	Variable Type	Data/Type Range
Index	Enter the name of a simple variable in which you want to store the index that was operated on during the last scan. When ER is true, this contains the index of the first error detected. Note: This number is relative to where the array operation started.	• simple • element of an array	• integer • double integer The range is 0 to 99999

10.1.3 Example of an Unary Array Instruction

After *switch1* becomes true and remains true, a value of 15 is moved into the first ten elements of the array variable *loaded_array*. The variable *switch2* is set true when the array is filled. The variable *fail* is set true if an error occurs.



10.2 Multi-Array Instruction (AR2)



Use this instruction to perform one of these operations:

- logical AND (AND) on bit data
- logical OR (OR) on bit data
- logical exclusive OR (XOR) on bit data
- addition (ADD)
- subtraction (SUB)
- multiplication (MUL)
- division (DIV)

Use this instruction to perform logical or arithmetic operations on:

- simple variables
- a single element of an array
- multiple elements of an array

In1 and In2 each can be an array variable or a simple variable/constant. You can place the result in a simple or array variable.

When EN transitions from false to true, the instruction prepares to carry out the function specified in Operation by latching (buffering) all parameters. If you have programmed the array instruction to operate on the array elements over multiple program scans, the first scan sets up the operation. Then, the operation is performed as programmed over a series of scans until the operation is done.

As long as EN remains true, the instruction completes the operation on the specific lengths (Length1 and Lenght2) of the variables in In1 and In2, operating on the number of elements specified in Elems/Scan with each program scan.

A specified number of elements (Length_Out) is stored in Out. Index acts as a marker indicating the last element of the array that was operated on during the last scan.

10.1.4 Input Parameters for the Multi-Array Instruction

This table lists the inputs for the AR2 instruction and the variable type and data type/range that each input supports.

Parameter	Description	Variable Type	Data/Type Range
EN	While this input is true, the instruction executes. When this input is false, the instruction does not execute and is reset.	Connect a Boolean input or output.	
Elems/ Scan	Enter the number of elements to operate on each time the instruction is scanned. To operate on all the elements, enter 0. To distribute the array operation over a number of program scans, enter the number of elements you want to operate on each time the instruction is scanned (after the first scan).	• simple • constant • element of an array	• integer or double integer • timer (*name*.TPreset and *name*.Elapsed) • counter (*name*.CPreset and *name*.Current) The range is 0 through 100,000 and must not exceed the maximum array index plus 1.

Parameter	Description	Variable Type	Data/Type Range
In1	Enter the name of the first variable or constant that is to be operated on.	• constant • array • simple • element of an array	• integer • double integer • timer (*name*.TPreset and *name*.Elapsed) • counter (*name*.CPreset and *name*.Current)
Length1	Enter the number of elements within In1 that this instruction is to operate on. If In1 is a simple variable or constant, enter 1. If In1 is an array variable, enter a number 1 or larger.	• simple • constant • element of an array	• integer or double integer • timer (*name*.TPreset and *name*.Elapsed) • counter (*name*.CPreset and *name*.Current) The range is 0 through 100,000 and must not exceed the maximum array index plus 1.

Parameter	Description	Variable Type	Data/Type Range
In2	Enter the name of the second variable or constant that is to be operated on.	• constant • array • simple • element of an array	• integer • double integer • timer (*name*.TPreset and *name*.Elapsed) • counter (*name*.CPreset and *name*.Current)
Length2	Enter the number of elements within In2 that this instruction is to operate on. If In2 is a simple variable or constant, enter 1. If In2 is an array variable, enter a number 1 or larger. When two of either Length 1, Length2, or Length_Out are greater than 1, they must be equal, and the third parameter must also be equal or be a value of 1.	• simple • constant • element of an array	• integer or double integer • timer (*name*.TPreset and *name*.Elapsed) • counter (*name*.CPreset and *name*.Current) The range is 0 through 100,000 and must not exceed the maximum array index plus 1.
Length_ Out	Enter the number of elements to store in Out. If Out is a simple variable, enter 1. If Out is an array variable, enter a number 1 or larger. When two of either Length 1, Length2, or Length_Out are greater than 1, they must be equal, and the third parameter must also be equal or be a value of 1.		

Parameter	Description	Variable Type	Data/Type Range
Operation	Enter the mnemonic of the operation you want to perform on In1 and In2. Choose from these operations: ● ADD (addition) ● AND (bit-wise logical AND) ● DIV (division) ● MUL (multiplication) ● OR (bit-wise logical OR) ● SUB (subtraction) ● XOR (bit-wise logical exclusive OR)	Character string	A−Z

10.2.2 Output Parameters for the Multi-Array Instruction

This table lists the outputs for the AR2 instruction and the variable type and data type/range that each output supports.

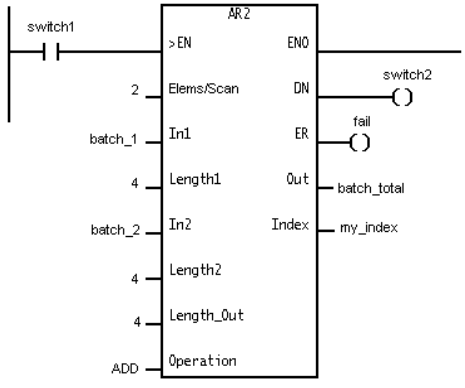
Parameter	Description	Variable Type	Data/Type Range
ENO	Use this output as the input to another instruction for easily chaining multiple instructions. This output follows the state of EN unless an error occurs.	Connect a contact, coil or Boolean parameter of another instruction.	
DN	This output is true when the instruction has operated on the last array element or an error was found during the first scan. DN is false when EN is false.		
ER	This output is set true after the instruction is completed when an error was encountered during its operation. This output is set false when EN is false.		
Out	Enter the name of a variable in which you want to store the result of the Operation. The data type for this variable must be the same data type as that for In1 and In2.	• array • simple • element of an array	A-Z
Index	Enter the name of a simple variable in which you want to store the index that was operated on during the last scan. When ER is true, this contains the index of the first error detected. Note that this number is relative to where the array operation started.	• simple • element of an array	• integer • double integer The range is 0 to 99999

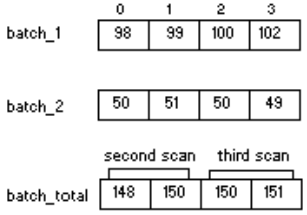
10.2.3 Example of a Multi-Array Instruction

When *switch_1* becomes true and remains true, the parameters are latched and checked. On subsequent scans, two elements of the array variables *batch_1* and *batch_2* are added together.

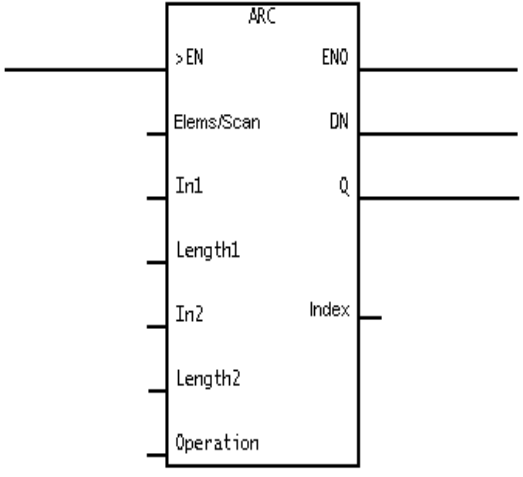
The totals are stored in the elements of the array variable *batch_total*.

The variable *switch2* is set true when all the elements of *batch_1* and *batch_2* have been operated on. The variable *fail* is set true if an error occurs.





10.3 Array Compare (ARC)



Use this instruction to perform one of these operations:

- equal to (EQ)
- not equal to (NE)
- greater than (GT)
- greater than or equal (GE)
- less than (LT)
- less than or equal (LE)

Use this instruction to perform comparison operations on:

- simple variables
- a single elements of an array
- multiple elements of an array

When EN transitions from false to true, the instruction prepares to carry out the function specified in Operation by latching (buffering) all parameters. If you have programmed the array instruction to operate on the array elements over multiple program scans, the first scan sets up the operation. Then, the operation is performed as programmed over a series of scans until the operation is done.

As long as EN remains true, it completes the operation on the specific lengths (Length1 and Length2) of the variables in In1 and In2, operating on the number of elements specified in Elems/Scan with each program scan.

Index acts as a marker indicating the last element of the array that was operated on during the last scan.

10.3.1 Input Parameters for the Array Compare Instruction

This table lists the inputs for the ARC instruction and the variable type and data type/range that each input supports.

Parameter	Description	Variable Type	Data/Type Range
EN	While this input is true, the instruction executes. When this input is false, the instruction does not execute and is reset.	Connect a Boolean input or output.	
Elems/ Scan	Enter the number of elements to operate on each time the instruction is scanned. To operate on all the elements, enter 0. To distribute the array operation over a number of program scans, enter the number of elements you want to operate on each time the instruction is scanned (after the first scan).	• simple • constant • element of an array	• integer or double integer • timer (*name*.TPreset and *name*.Elapsed) • counter (*name*.CPreset and *name*.Current) The range is 0 through 100,000 and must not exceed the maximum array index plus 1.

Parameter	Description	Variable Type	Data/Type Range
In1	Enter the name of the first variable or constant that is to be operated on.	• constant • array • simple • element of an array	• integer • double integer • timer (*name*.TPreset and *name*.Elapsed) • counter (*name*.CPreset and *name*.Current)
Length1	Enter the number of elements within In1 that this instruction is to operate on. If In1 is a simple variable or constant, enter 1. If In1 is an array variable, enter a number 1 or larger.	• simple • constant • element of an array	• integer or double integer • timer (*name*.TPreset and *name*.Elapsed) • counter (*name*.CPreset and *name*.Current) The range is 0 through 100,000 and must not exceed the maximum array index plus 1.

Parameter	Description	Variable Type	Data/Type Range
In2	Enter the name of the second variable or constant that is to be operated on.	• constant • array • simple • element of an array	• integer • double integer • timer (*name*.TPreset and *name*.Elapsed) • counter (*name*.CPreset and *name*.Current)
Length2	Enter the number of elements within In2 that this instruction is to operate on. If In2 is a simple variable or constant, enter 1. If In2 is an array variable, enter a number 1 or larger. This value must be equal to the value for Length1 if both are greater than 1.	• simple • constant • element of an array	• integer or double integer • timer (*name*.TPreset and *name*.Elapsed) • counter (*name*.CPreset and *name*.Current) The range is 0 through 100,000 and must not exceed the maximum array index plus 1.

Parameter	Description	Variable Type	Data/Type Range
Operation	Enter the mnemonic of the operation you want to perform on In1 and In2. Choose from these operations: • EQ (equal to) • GT (greater than) • GE (greater than or equal) • LT (less than) • LE (less than or equal) • NE (not equal to)	Character string	A-Z

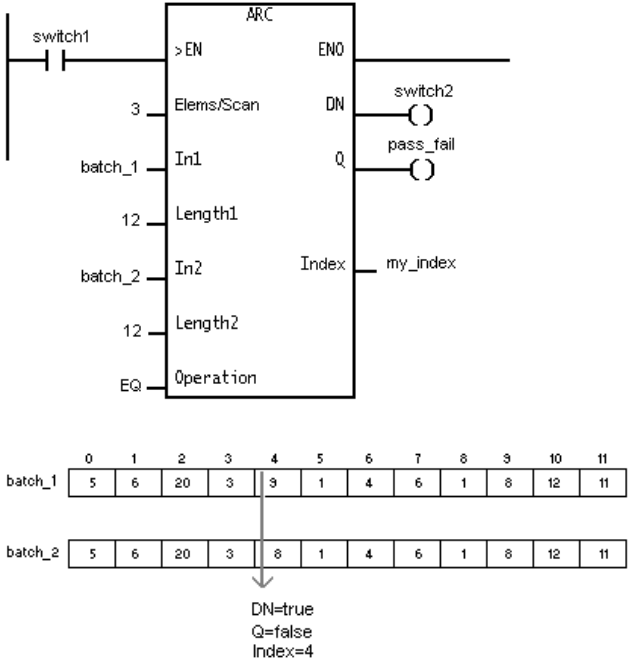
10.3.2 Output Parameters for the Array Compare Instruction

This table lists the outputs for the ARC instruction and the variable type and data type/range that each output supports.

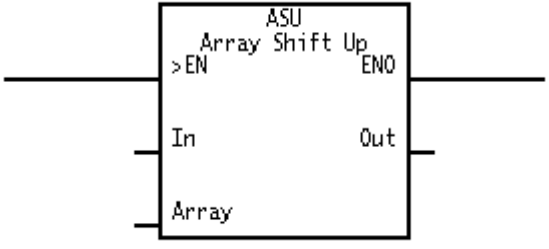
Parameter	Description	Variable Type	Data/Type Range
ENO	Use this output as the input to another instruction for easily chaining multiple instructions. This output follows the state of EN unless an error occurs.	Connect a contact, coil or Boolean parameter of another instruction.	
DN	This output becomes true when: • The instruction has operated on the last array element. • A comparison operation results in a false condition, even though all elements within each array have not been evaluated. • An error is encountered during the first scan. DN is false when EN is false.		
Q	This output becomes true when the operation is complete (DN is true) and the result of the comparison is true. Otherwise, the Q output is false.		
Index	Enter the name of a simple variable in which you want to store the index that was operated on during the last scan.	• simple • element of an array	• integer • double integer The range is 0 to 99999

10.3.3 Example of an Array Compare Instruction

When *switch1* is set true and remains true, the parameters are latched and checked. On subsequent scans, the next three elements of batch_1 are compared to the corresponding three elements of batch_2. The first comparison that evaluates false causes the instruction to be completed, with Q set false. If all twelve corresponding array elements had been equal, the instruction would have finished on the fifth scan with Q set true.



10.4 Array Shift Up (ASU)



Use this instruction to shift elements in an array from bottom to top. This instruction can be useful for tracking parts and/or data.

When EN transitions from false to true, the instruction:

- moves the first element of Array to Out
- shifts the remaining elements up to fill in the empty location
- moves In into the last element of the array

10.4.1 Input Parameters for the Array Shift Up Instruction

This table lists the inputs for the ASU instruction and the variable type and data type/range that each input supports.

Parameter	Description	Variable Type	Data/Type Range
EN	While this input is true, the instruction executes. When this input is false, the instruction does not execute.	Connect a Boolean input or output.	
In	Enter the name of a variable or a constant that you want to move into the last element position of the array. If no value is specified, 0 is used.	• simple • constant • element of an array	• integer • double integer • timer (*name*.TPreset and *name*.Elapsed) • counter (*name*.CPreset and *name*.Current)
Array	Enter the name of the array variable that you want to use.	array	• integer • double integer

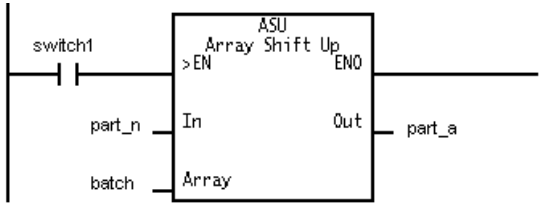
10.4.2 Output Parameters for the Array Shift Up Instruction

This table lists the outputs for the ASU instruction and the variable type and data type/range that each output supports.

Parameter	Description	Variable Type	Data/Type Range
ENO	Use this output as the input to another instruction for easily chaining multiple instructions. This output follows the state of EN unless an error occurs.	Connect a contact, coil or Boolean parameter of another instruction.	
Out	Enter the name of variable in which you want to store the top-most element that was shifted out of the array. This variable must be of the same data type as In.	• simple • element of an array	• integer • double integer • timer (*name*.TPreset and *name*.Elapsed) • counter (*name*.CPreset and *name.*Current)

10.4.3 Example of an Array Shift Up Instruction

When *switch1* transitions from false to true, the topmost element shifted from *batch* will be placed in the variable *part_a*, all the elements in the array are moved up one element, and the data in variable *part_n* will be moved into the last element of the array.



Before the ASU instruction executes:

This is batch before executing the ASU instruction.

[0] FFFF
[1] AAAA
[2] 3333
[3] CCCC
[4] F0F0
[5] 00FF

part_n prior to ASU instruction 0000

After the ASU instruction executes:

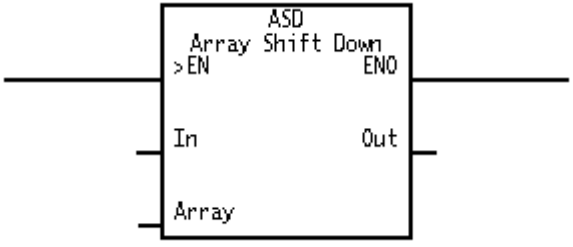
part_a now has contents of Array [0] FFFF

All elements are shifted upwards in array batch.

[0] AAAA
[1] 3333
[2] CCCC
[3] F0F0
[4] 00FF
[5] 0000

part_n is placed in the last element 0000

10.5 Array Shift Down (ASD)



Use this instruction to shift elements in an array from top to bottom. This instruction can be useful for tracking parts and/or data.

When EN transitions from false to true, the instruction:

- moves the last element of Array to Out
- shifts the remaining elements down to fill in the empty location
- moves In into the first element of the array

10.5.1 Input Parameters for the Array Shift Down Instruction

This table lists the inputs for the ASD instruction and the variable type and data type/range that each input supports.

Parameter	Description	Variable Type	Data/Type Range
EN	While this input is true, the instruction executes. When this input is false, the instruction does not execute.	Connect a Boolean input or output.	
In	Enter the name of a variable or a constant that you want to move into the first element position of Array. If no value is specified, a 0 is used.	• simple • constant • element of an array	• integer • double integer • timer (*name*.TPreset and *name*.Elapsed) • counter (*name*.CPreset and *name*.Current)
Array	Enter the name of the array variable that you want to use.	array	• integer • double integer

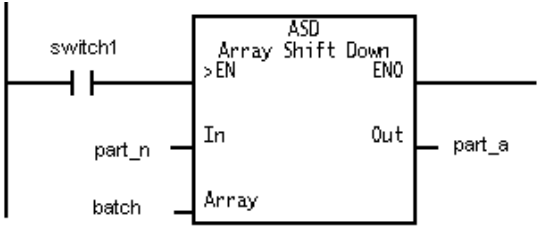
10.5.2 Output Parameters for the Array Shift Down Instruction

This table lists the outputs for the ASD instruction and the variable type and data type/range that each output supports.

Parameter	Description	Variable Type	Data/Type Range
ENO	Use this output as the input to another instruction for easily chaining multiple instructions. This output follows the state of EN unless an error occurs.	Connect a contact, coil or Boolean parameter of another instruction.	
Out	Enter the name of variable in which you want to store the bottom-most element that was shifted out of the array. This variable must be of the same data type as In.	• simple • element of an array	• integer • double integer • timer (*name*.TPreset and *name*.Elapsed) • counter (*name*.CPreset and *name.*Current)

10.5.3 Example of an Array Shift Down Instruction

When *switch1* transitions from false to true, the bottom-most element shifted from the array *batch* will be placed in the variable *part_a*, all the elements in the array are moved down one element, and the value in the variable *part_n* will be moved into the array element *batch[0]*.



Before the ASD instruction executes:

part_n prior to ASD instructions 0000

This is batch before executing the ASD instruction.

[0] FFFF
[1] AAAA
[2] 3333
[3] CCCC
[4] F0F0
[5] FF00

After the ASD instruction executes:

part_n is shifted into batch [0] 0000

All elements are shifted downward in batch.

[0] 0000
[1] FFFF
[2] AAAA
[3] 3333
[4] CCCC
[5] F0F0

part_a now has the previous contents of batch [5] FF00

10.6 About the State of the Unary Array, Multi-Array, and Array Compare Instruction Outputs under Various Input Conditions

The following shows the state of each array instruction output under various input conditions.

Input Conditions		Output Conditions			
EN	**instruction's state**	**DN**	**ER**	**Q**	**ENO**
true	initial scan	false	false	false	true
false	disabled	false	false	false	false
true	rising edge: error latching the inputs and outputs	true	true	false	false
true	rising edge: inputs are latched	false	false	false	true
true	instruction is executing	false	false	false	true
true	the operation is complete with an error	true	true	false	false
true	the operation is complete and error-free	true	false	true	true

The AR1 and AR2 instructions do not stop on an arithmetic error. They correct the error, and the operation continues until it is completed. If multiple errors occur, the first error is logged and the value pointing to where the error occurred is placed into the Index output when the instruction has completed.

The ARC instruction does not have an ER output. Errors occur during the initial scan, setting the DN on.

10.7 Errors Caused by Array Instructions

This section describes the possible errors for the Array instructions.

10.7.1 Errors Caused by the Unary Array Instruction

Errors can occur when parameters are latched and checked. If an error occurs at this point, no operation occurs.

If an arithmetic error occurs during the instruction's operation, a default value is generated and the operation continues.

When an error occurs:

- ENO is set to ERROR_ENO.
- When AR1 is complete,
  - the ER output is set true and retains its value until EN goes false
  - the Index output points to the first entry that caused the error

These errors can occur when you are using the AR1 instruction in a program.

If this error occurs:	Do the following:
The result of the arithmetic calculation is too large for Out.	Use smaller values or re-arrange the calculation so that errors do not occur.
Cannot take the square root of a negative number.	Do not perform a square root operation on a negative number.
Array's length inputs are greater than 1 but are not equal.	Make sure the values are equal.
The Elems/Scan input is less than 0.	Specify a value of 0 or greater.
• Length_In is larger than the In input. (This input can exceed the array size if starting at the first element or if starting somewhere within the array.) • Length_In is less than or equal to 0. • The value in Length_Out is less than or equal to 0. • The value in Length_Out is too large for the array block's Out.	Make sure Length_In and Length_Out are within the range of the array.
• The array index is negative. • The array index is too large.	Specify a valid array element.

10.7.2 Errors Caused by the Multi-Array Instruction

Errors can occur when parameters are latched and checked. If an error occurs at this point, no operation occurs.

If an arithmetic error occurs during the instruction's operation, a default value is generated and the operation continues.

When an error occurs:

- ENO is set to ERROR_ENO.
- When AR2 is complete,
  - the ER output is set true and retains its value until EN goes false
  - the Index output points to the first entry that caused the error

These errors can occur when you are using the AR2 instruction in a program:

If this error occurs:	**Do the following:**
The result of the arithmetic calculation is too large for Out.	Use smaller values or re-arrange the calculation so that errors do not occur.
Cannot divide by zero. (As a result, Out contains the largest signed value allowed for Out's data type.)	Make sure that the In2 contains values greater than 0.
Array's length inputs are greater than 1 but are not equal.	Make sure the values are equal.
The Elems/Scan input is less than 0.	Specify a value of 0 or greater.

If this error occurs:	Do the following:
• The value in Length1 is too large for array block's In1. • The value in Length2 is too large for array block's In2. (These inputs can exceed the array size if starting at the first element or if starting somewhere within the array.) • The value in Length1 is less than or equal to 0. • The value in Length2 is less than or equal to 0. • The value in Length_Out is too large for array block's Out. • The value in Length_Out is less than or equal to 0.	Make sure the lengths are within the range of the array.
• The array index is negative. • The array index is too large.	Specify a valid array element.

10.7.3 Errors Caused by the Array Compare Instruction

Errors only occur when parameters are latched and checked. If an error occurs at this point, the DN output is set true.

When an error occurs, ENO is set to ERROR_ENO.

These errors can occur when you are using the ARC instruction in a program.

If this error occurs:	Do the following:
Array's length inputs are greater than 1 but are not equal.	Make sure the values are equal.
The Elems/Scan input is less than 0.	Specify a value of 0 or greater.
• The value in Length1 is too large for array block's In1. • The value in Length2 is too large for array block's In2. (These inputs can exceed the array size if starting at the first element or if starting somewhere within the array.) • The value in Length1 is less than or equal to 0. • The value in Length2 is less than or equal to 0.	Make sure the lengths are within the range of the array.
• The array index is negative. • The array index is too large.	Specify a valid array element.

10.7.4 Errors Caused by the Array Shift Up and Array Shift Down Instructions

These errors can occur when you are using the ASU and ASD instructions in a program. They are logged in the error log.

If this error occurs:	Then:	Do the following:
The array index is negative.	ENO is set according to ERROR_ENO, and element zero of the array is used for the instruction's operation.	Specify a valid array element.
The array index is too large.	ENO is set according to ERROR_ENO, and the last element of the array is used for the instruction's operation.	Specify a valid array element.

11.0 Program Control Instructions

Use the Program control instructions to change the sequence of ladder logic execution.

Choose from these instructions:

- SET
- JMP
- LBL

See the input and output parameter for each instruction for specific information.

Using Jump and Label Instructions to Skip or Repeat Portions of Ladder Logic

Use JMP and LBL instructions together to jump forward or backward within a ladder program. Jumping forward helps you save program scan time by only executing portions of the program when they are needed. By jumping backwards in a program, you can repeat iterations of a logic segment.

Follow these rules when using JMP and LBL constructs:

- You can jump to a single LBL instruction from multiple JMP instructions.
- A rung can contain only one JMP instruction.

IMPORTANT

- The JMP instruction must be the last instruction on the rung.
- The LBL instruction must be the first instruction on a rung.
- Any name used in the JMP or LBL instructions must be defined as a Label. The Editor assigns the type when you first enter a new variable name.

Be aware of the following conditions:

CAUTION: Avoid jumping backwards an excessive number of times since this can cause a STOP ALL error code 14, which is the Processor Overlap Limit Exceeded error. Failure to observe this precaution could result in damage to, or destruction of, the equipment.

CAUTION: Avoid skipping timer instructions using a JMP instruction. The timer's Boolean output will not get set if the timer does not execute. Failure to observe this precaution could result in damage to, or destruction of, the equipment.

11.1 Set Event (SET)

Use the Set Event instruction to synchronize a ladder program with another program of any type within the same rack. When EN transitions from off to on (false-to-true), it sets the name of a software event (Event Name).

11.1.1 Input Parameters for the Set Event Instruction

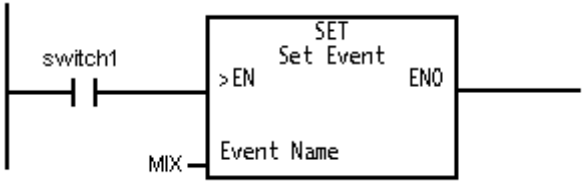
Parameter	Description
EN	While this input is true, the instruction executes. When this input is false, the instruction is not executed and ENO is false. Connect a Boolean input or output.
Event Name	Enter the name of the software event you want to use. IMPORTANT Software Events are used only in the Event Name input of SET instructions. When you specify a new name in the Event Name input of the Set Event instruction, the Editor automatically defines it as a Software Event.

11.1.2 Output Parameters for the Set Event Instruction

Parameter	Description
ENO	Use this output as the input to another instruction for easily chaining multiple instructions. This output follows the state of the EN input. Connect a Boolean input or a coil.

11.1.3 Example of the SET Instruction

When *switch1* transitions from false to true, the event *MIX* will be set and the program waiting for *MIX* to become true will become ready to run. If the waiting program is of a higher priority than the one that executed this instruction, the higher-priority program executes immediately following the instruction.



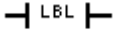
11.2 Jump (JMP)

Use the Jump instruction to skip or repeat rungs by jumping to the rung identified by the Label instruction. Use this output instruction to jump to rungs that fall earlier or later within the program or to repeat rungs.

When the rung containing the JMP instruction goes true, the Processor jumps to the rung identified by a Label instruction that has the same name as is used on the JMP instruction. If the rung containing the JMP instruction is false, the jump is not performed and program execution continues with the next sequential rung.

Enter the name used on the LBL instruction that identifies the logic to which you want to jump. The JMP coil must be the last coil on a rung.

11.3 Label (LBL)

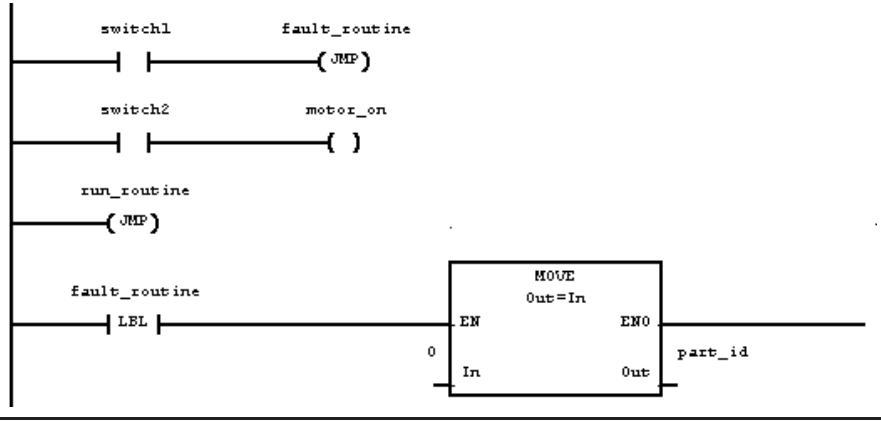


Use the Label instruction to mark a ladder logic rung as a target for a JMP instruction. Place the LBL instruction as the first instruction on a rung and enter a unique label name. This is the same name that you will enter on a JMP instruction.

When a rung containing the JMP instruction becomes true, execution continues at the rung with the corresponding label.

11.4 Example of Using the Jump and Label Instruction

This logic shows that when *switch1* is true, the next rung to execute is the one identified by the label *fault_routine*, which moves a 0 into the variable *part_id*.



11.5 The Error Caused by the Jump Instruction

If a Label does not exist, an error will be logged and control will pass to the next rung.

12.0 I/O Read and Write Instructions

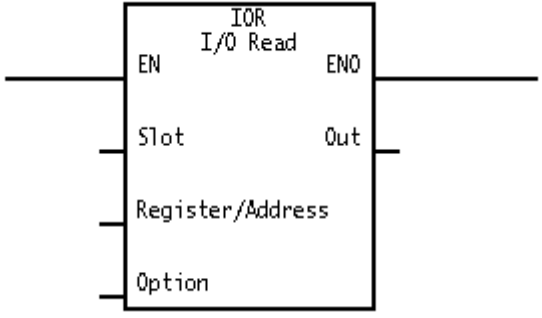
Use the I/O Read and I/O Write instructions to send information to and receive information from I/O modules. These instructions are particularly useful for reading data from and writing data to non-Reliance I/O modules that are only byte-accessible or to any modules that have not been configured.

The supported parameters are:

- simple integers and double integers
- integer and double integer constants
- element of an integer or double integer array

See the input and output parameter descriptions for each instruction for specific information.

12.1 I/O Read (IOR)



Use the I/O Read instruction to get information from I/O modules. This instruction is useful for reading data from:

- non-Reliance I/O modules that are only byte-accessible
- any modules that have not been configured in the rack configurator

Use the I/O Read instruction to read a:

- byte - This is used for modules that support only 8-bit addressing.
- double byte - This should only be used for non-Reliance modules that contain 16-bit data but only support 8-bit Multibus access.
- integer (16-bits)
- double integer (32-bits)

While the enable bit is true, the Processor reads the I/O data you specified from the location you specified. The data is stored in Out.

12.1.1 Input Parameters for the I/O Read Instruction

This table lists the inputs for the I/O Read instruction and the variable type and the data type/range that each input supports.

Parameter	Description	Variable Type	Data/Type Range
EN	While this input is true, the instruction executes. When this input is false, the instruction is not executed and ENO is false.	Connect a Boolean input or output.	
Slot	Enter the slot number of the AutoMax rack in which the I/O module resides. If you are using the Address input, you cannot enter a slot number.	• simple • constant • element of an array	integer (4-15)
Register/	Enter the 16-bit register number to be accessed in the selected slot within the AutoMax rack. The slot and register information define the location of the data.		integer or double integer (0 to 32767 [7FFFH])

Parameter	Description	Variable Type	Data/Type Range
Address	Enter the address of the location that contains the data you want to read. Use this input when you must read a byte from an “odd” address. If you enter an address, the slot input is ignored. You must calculate the address using this information. • For a module that follows AutoMax conventions, choose the base address based on the slot that the module occupies and add an offset equal to the register * 2. See “Listing of Base Addresses for Each Slot in the AutoMax Chassis,” section 12.3. • For a non-Reliance module, choose the address that corresponds to the location of the information on the module.	• simple • constant • element of an array	integer or double integer (0240000 to 2FFFFH)
Option	Define the amount of I/O data you want to read by entering an option number that corresponds to the amount of data you want. See Defining the Amount of I/O Data to Read.		integer (1 to 4)

12.1.2 Output Parameters for the I/O Read Instruction

This table lists the outputs for the I/O Read instruction and the variable type and data type/range that each output supports.

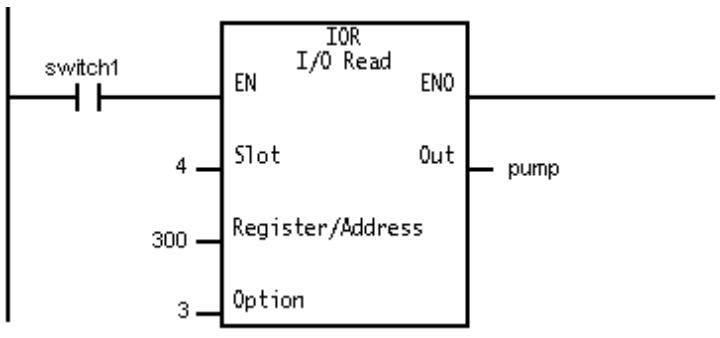
Parameter	Description	Variable Type	Data/Type Range
ENO	Use this output as the input to another instruction for easily chaining multiple instructions. This output follows the state of EN unless an error occurs.	Connect a contact, coil, or Boolean input of another instruction.	
Out	This parameter contains the requested data.	• simple • element of an array	• integer • double integer • timer (*name*.TPreset and *name.*Elapsed) • counter (*name*.CPreset and *name*.Current)

12.1.3 Defining the Amount of I/O Data to Read

To read:	In the Option field, enter:	In the Register/address field, enter:	Result:
byte	1	data's location	a byte is read
double byte Note: *Only use this option to read data from modules that only support an 8-bit Multibus access.*	2	data's location	The low byte is read first and then the high byte.
integer (16-bits) Note: *Use this option to read data from modules that support AutoMax addressing and data conventions.*	3	data's location as an even address	A 16-bit word is read from the designated address.
double integer (32-bits)	4	data's location as an even address	A 32-bit word is read from the designated address in this order: • MS 16-bit word • LS 16-bit word

12.1.4 Example of an I/O Read Instruction

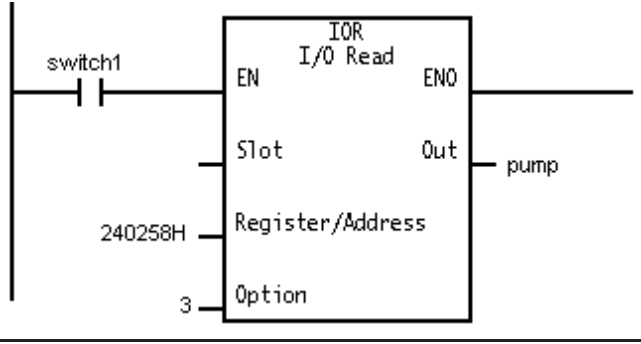
When *switch1* is true, the instruction reads one word of data from register 300 of the I/O module in slot 4 of the AutoMax rack. This data is placed in the variable *pump*.



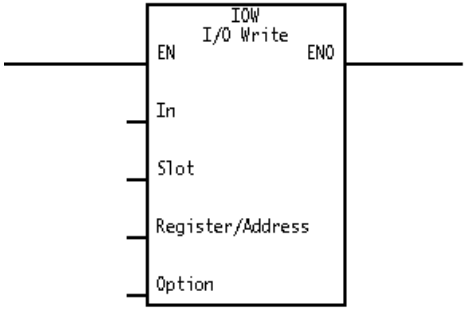
If you were to use an address instead of a slot and register location, the example would look like the instruction below. The Address of 240258H is derived by this formula:

base register address of slot 4 + 2 × the register number

Consequently, 240000H + 2(300)=240258H.



12.2 I/O Write (IOW)



Use the I/O Write instruction to send information to I/O modules. This instruction is particularly useful for writing data to:

- non-Reliance I/O modules that are only byte-accessible
- any modules that have not been configured

Use the I/O Write instruction to write a:

- byte — This is used for modules that support only 8-bit addressing
- double byte — This should only be used for non-Reliance modules that contain 16-bit data but only support 8-bit Multibus accesses
- integer (16-bits)
- double integer (32-bits)

When EN becomes true, the Processor writes the amount of I/O data you specified in the location you specified.

12.2.1 Input Parameters for the I/O Write Instruction

This table lists the inputs for the IOW instruction and the variable type and data type/range that each input supports.

Parameter	Description	Variable Type	Data/Type Range
EN	While this input is true, the instruction executes. When this input is false, the instruction is not executed and ENO is false.	Connect a Boolean input or an output.	
Slot	Enter the slot number of the AutoMax rack in which the I/O module resides. If you are entering an Address input, you cannot enter a slot number.	• simple • constant • element of an array	integer (4-15)
Register/	Enter the 16-bit register number to be accessed in the selected slot within the AutoMax rack. The slot and register information define the location of the data.		integer or double integer (0 to 32757 [7FFFH])

Parameter	Description	Variable Type	Data/Type Range
Address	Enter the address of the location where you want to write data. If you enter an address, the slot input is ignored. You must calculate the address using this information: For a module that follows AutoMax conventions, choose the base address based on the slot that the module occupies and add an offset equal to the register * 2. (See the Listing of Base Addresses for Each Slot in the AutoMax Chassis) For a non-Reliance module, choose the address that corresponds to the location of the information on the module.	• simple • constant • element of an array	integer or double integer (0240000 to 2FFFFFH)
Option	Define the amount of I/O data you want to write by entering an option number that corresponds to the amount of data you want. See Defining the Amount of I/O Data to Write.		integer (1-4)

12.2.2 Output Parameters for the I/O Write Instruction

This table lists the outputs for the IOW instruction and the variable type that each output supports. To use ENO, connect it to a contact, coil, or Boolean input of another instruction.

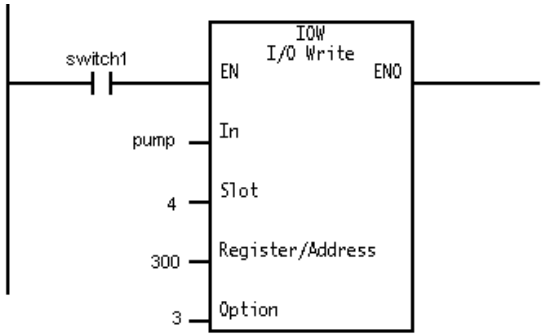
Parameter	Description
EN	Use this output as the input to another instruction for easily chaining multiple instructions. This output follows the state of EN unless an error occurs.

12.2.3 Defining the Amount of I/O Data to Write

To write:	In the Option field, enter:	In the Register/ address field, enter:	Result:
byte	1	data's location	a byte is written
double byte Note: *Use this option to write data to modules that only support an 8-bit Multibus access.*	2	data's location	The low byte is written first and then the high byte.
integer (16-bits) Note: *Use this option to write data from modules that support AutoMax addressing and data conventions.*	3	data's location as an even address	A 16-bit word is written from the designated address.
double integer (32-bits)	4	data's location as an even address	A 32-bit word is written from the designated address in this order: • MS 16-bit word • LS 16-bit word

12.2.4 Example of an I/O Write Instruction

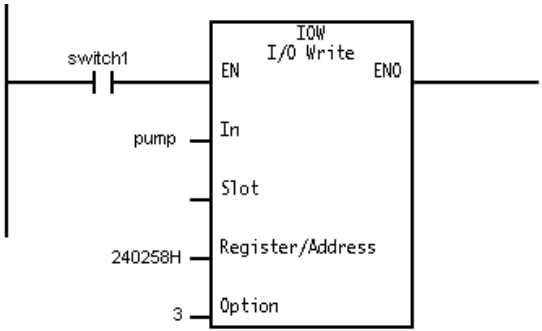
When *switch1* becomes true, the instruction writes one word of data from the variable *pump* to register 300 of the I/O module in slot 4 of the AutoMax rack.



If you were to use an address instead of a slot and register location, the example would look like the instruction below. The Address of 240258H is derived by this formula:

`base register address of slot 4 + 2 × the register number`

Consequently, 240000H + 2(300)=240258H.



12.3 Listing of Base Addresses for Each Supported Slot in the AutoMax Chassis

Slot	Hex Address Range	Slot	Hex Address Range
4	240000 to 24FFFF	10	2A0000 to 2AFFFF
5	250000 to 25FFFF	11	2B0000 to 2BFFFF
6	260000 to 26FFFF	12	2C0000 to 2CFFFF
7	270000 to 27FFFF	13	2D0000 to 2DFFFF
8	280000 to 28FFFF	14	2E0000 to 2EFFFF
9	290000 to 29FFFF	15	2F0000 to 2FFFFF

12.4 Errors Caused by the I/O Read and I/O Write Instructions

This section describes the possible errors for I/O read and I/O write instructions.

If this error occurs:	Then:	Do the following:
• An illegal slot number of an IOR or IOW is selected • An illegal register of an IOR or IOW is selected	ENO is set according to ERROR_ENO, and nothing is written to Out.	Correct the slot and register or address parameters used in the instruction.
An illegal option of an IOR or IOW instruction is selected.	ENO is set according to ERROR_ENO, and nothing is written to Out.	Correct the number in the option field. The valid range is 1-4.
The array index is negative	ENO is set according to ERROR_ENO, and element zero of the variable will be used for the instruction's operation.	Specify a valid array element.
The array index is too large	ENO is set according to ERROR_ENO, and element zero of the variable will be used for the instruction's operation.	Specify a valid array element.
(I/O Read Only) The result is larger than what Out's data type supports	ENO is set according to ERROR_ENO, and Out contains the largest signed value allowed for the data type being used.	Specify the variable in Out to be a double integer.

13.0 Immediate Input and Output Instructions

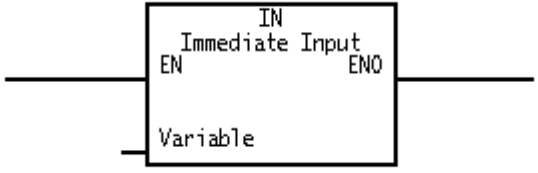
Use the Immediate Input to update the latched value with the most current value of a global input.

Use the Immediate Output instruction to update a global variable's physical location with the current latched value.

These instructions support simple Boolean, integer, or double integer variables.

See the input and output parameter descriptions for each instruction for specific information.

13.1 Immediate Input (IN)



Use the Immediate Input instruction to update the program's latched value corresponding to a global variable with that global variable's current value. Since inputs are latched at the start of a program scan, the IN instruction is useful for gathering data that may have changed since the start of a program scan.

While EN is true, the instruction reads the state of a program's global variable (Variable) at its physical location and updates the latched value with the new value. The newly updated latched value is used in subsequent instructions as needed.

13.1.1 Input Parameters for the Immediate Input Instruction

This table lists the inputs for the IN instruction and the variable type and data type/range that each input supports.

Parameter	Description	Variable Type	Data/Type Range
EN	While this input is true, the instruction executes. When this input is false, the instruction is not executed and ENO is false.	Connect a Boolean input or output.	
Variable	Enter the global variable from which you want to use the latest value.	Simple	• Boolean • integer • double integer

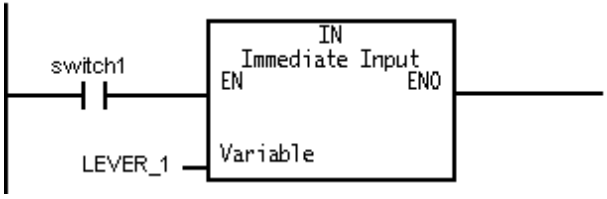
13.1.2 Output Parameters for the Immediate Input Instruction

This table lists the output for the IN instruction and the variable type that it supports. To use ENO, connect it to a contact, coil, or Boolean input of another instruction.

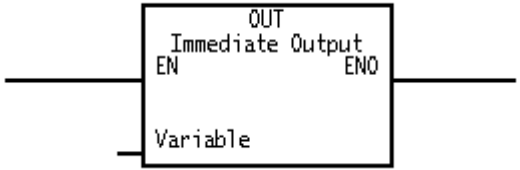
Parameter	Description
EN	While this input is true, the instruction executes. When this input is false, the instruction is not executed and ENO is false.
Variable	Enter the global variable from which you want to use the latest value.
ENO	Use this output as the input to another instruction for easily chaining multiple blocks. This output follows the state of the EN input.

13.1.3 Example of an Immediate Input Instruction

While *switch1* is true, the program's latched value for the global variable *LEVER_1* is updated.



13.2 Immediate Output (OUT)



Use the Immediate Output instruction when you need to update a global variable's physical location prior to the end of the program scan. Output locations are normally updated at the end of a program scan. This instruction lets you immediately use the latest output value for a global variable.

While EN is true, the instruction updates a global variable's (Variable) actual location with the value from the program's latched value.

13.2.1 Input Parameters for the Immediate Output Instruction

This table lists the inputs for the OUT instruction and the variable type and data type/range that each input supports.

Parameter	Description	Variable Type	Data/Type Range
EN	While this input is true, the instruction executes. When this input is false, the instruction is not executed and ENO is false.	Connect a Boolean input or output	
Variable	Enter the global variable that you want to update with the value from the program's latched value.	Simple	• Boolean • integer • double integer

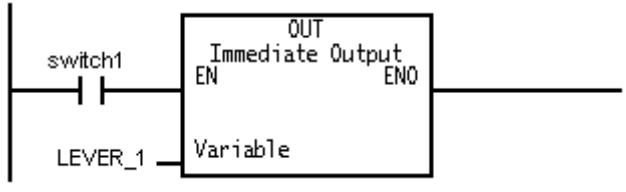
13.2.2 Output Parameters for the Immediate Output Instruction

This table lists the output for the OUT instruction and the variable type and data type/range that it supports. To use ENO, connect it to a contact, coil, or Boolean input of another instruction.

Parameter	Description
ENO	Use this output as the input to another instruction for easily chaining multiple blocks. This output follows the state of the EN input.

13.2.3 Example of an Immediate Output Instruction

While *switch1* is true, the global variable *LEVER_1* is updated with its current latched value.



Appendix A

Using Variables

All operations performed in AutoMax programs use symbolic names (variables) to represent storage locations for inputs, outputs, and other data. You use these variables to reference ladder instruction input and output parameters.

With the Editor you name variables just as you do in the Variable Configurator. Variable names can be a maximum of 16 characters. The valid characters are A–Z, a–z, 0–9, and _ (underscore). Names must begin with a letter.

The type of variable you choose depends upon the type of operation to be performed. Variables are either:

• simple	stores a single piece of data The data can be Boolean, integer, or double integer.
• array	stores a collection of data of the same data type The data can be Boolean, integer, or double integer.
• data structure	stores both Boolean and double integer data Data structures are used for Timer and Counter data types. You can use pre-defined keywords to help you specify the individual timer and counter elements.

Variables also have these properties:

- Variables store data according to the data type they are defined as. The Editor assigns a default data type to variables you enter and provides a way for you to change the type, so you need not enter the variable type indicators @, %, and !. For more information, see section A.1.
- Variables have a scope, either global or local. The case of the first letter of the variable name indicates the scope of the variable. A variable name beginning with an upper-case letter is interpreted by the Editor as a global variable, while a variable name beginning with a lower-case letter is interpreted by the Editor as a local variable. For more information, see section A.3.

  You cannot have two variables of different data types or scopes with the same name.

  To configure global variables, you must define them in the Variable Configurator.
- Variables must have a name and can have a description.

An additional characteristic of simple and array variables is the option for you to access (index into) the data within these variables down to the bit level. For example, within a simple integer variable you can access each bit with a variable or with a constant. For more information, see section A.2.

A.1 Data Types

This section describes Boolean, integer, double integer, timer, and counter variables in more detail.

A.1.1 Boolean Variables

A Boolean variable stores the status (either 1 or 0) of one bit. When a bit has a value of 1, it is said to be on and logically true. When a bit has a value of 0, it is said to be off and logically false.

You can reference an individual bit using any of these methods:

Method	Example
simple Boolean	switch1
bit-indexed integer or double integer	vat.15 vat.31 vat.fill
Boolean array element	panel[20]
bit-indexed integer or double integer array element	panel[rack2].15
timer and counter status bits	timer_name.Q counter_name.QD

See section A.2 for more information.

This table presents data about the number of bits for some configurations:

A rack with:	Contains a maximum of:
one 7010 Processor	16, 384 bits
Common Memory module and 6011 processor(s)	14,496 bits
one 6011 Processor	11,464 bits

These bits are used by ladder programs for simple Boolean variables and Boolean array variables. These bits can be used by a single program or split among multiple programs.

You define the size of an array by entering a value in Maximum Array Index located within the Variable Properties dialog box. For more information about arrays, see section A.4.

A.1.2 Integer and Double Integer Variables

Integer and double integers have the following data and value ranges:

	Data Range	Value Range
integer	16 bits	–32768 to 32767
double integer	32 bits	–2147483648 to 2147483647

You can reference an integer or double integer variable the following ways:

Method	Example
simple	vat
element of an array	panel[rack2] panel[10]
timer or counter	timer_name.Elapsed counter_name.Current

You can also use decimal and hexadecimal constants. See section A.2 for more information.

A.1.3 Timer Variables

A timer variable is a data structure that combines pre-defined elements into a single symbol.

This element:	Is a:
TPreset	double integer element that stores the value at which the timer expires (where 1=10ms)
Elapsed	double integer element that stores the amount of time that has elapsed (where 1 = 10 ms) Elapsed is initialized to 0 when a program first begins to run unless the timer is a global and declared non-volatile in the Variable Configurator.
Q	Boolean output that indicates that the timer timed out
T	Boolean output that indicates that the timer is timing

To reference any element of a timer variable in a ladder logic program

- Assign a name to the timer variable and then add the reserved element name as an index.

For example, to reference a timer's elapsed value, type this: *name*.Elapsed. And to reference the Q status bit, type this: *name*.Q.

Where *name* is the name of the timer data structure.

IMPORTANT:

- Timer elements cannot be forced.
- You must enter global timers into the Variable Configurator as five-element, double integer, non-volatile arrays. Example: TIMER1!(4).

A.1.4 Counter Variables

A counter variable is a data structure that combines pre-defined elements into a single symbol.

This element:	Is a:
Current	double integer element that contains the current count Current is initialized to 0 when a program first begins to run unless the counter is a global and declared non-volatile in the Variable Configurator.
CPreset	double integer element that contains the value that the counter should count to or from
QU	Boolean output that is true when the value in Current is greater than or equal to the preset value (CPreset)
QD	Boolean output that is true when the value in Current is less than or equal to zero

To reference any element within a counter variable in a ladder logic program

- Assign a name to the counter variable and then add the reserved element name as an index

For example, to reference a counter's Current element, type this: *name*.Current.

where *name* is the name of the counter data structure.

IMPORTANT

- Counter elements cannot be forced.
- You must enter global counters into the Variable Configurator as five-element, double integer, non-volatile arrays.
  Example: COUNTER1!(4).

A.1.5 Labels

Labels are used only in JMP and LBL instructions. When you enter a new name for a JMP or LBL instruction, the Editor automatically defines it as a Label.

A.2 Accessing Data Within Variables Via Bit-Indexing and Element-Indexing

For the ladder instruction parameters, you can specify bits within integer and double integer variables and elements and bits within an array variable. You can refer to bits or elements by using constants or variables. For example, you could access bit 17 in the double integer variable *pump* by using either *pump.17*, or any bit within *pump* using *pump.fill*. Notice a period (.) was used to separate *pump* from the bit. The period (.) is the bit delimiter.

To specify an element name or number, enclose it in square brackets ([ ]). For example, if *pump* was an array and you want element number 17 within the array, you can access it by: *pump[17]*. To access any element within the array *pump*, you can use *pump[fill]*. The square brackets ([ ]) are the element delimiter.

This methodology is called indexing. Using bit-indexed and element-indexed variables can help reduce the number of unique names required for your application and help you manage data collection and manipulation. Bit-indexed variables can also help make referring to I/O points easier. Just assign a name for the register corresponding to the I/O module and reference each I/O point by appending the appropriate bit number. For example, if one I/O module is controlling all the switches for a conveyor belt, you can assign the name “conveyor_switch” to the I/O module and reference each I/O point by appending a bit’s name or constant, such as “conveyor_switch.2.”

However, use bit-indexing with caution because the program’s execution time increases with the complexity of the variable.

Only variables assigned to I/O modules can have names assigned to the whole integer and to each bit.

You can use bit-indexed variables on relay instructions. For specific information about the supported variable types for an instruction, see the section pertaining to that instruction.

Element and bit variable names can each be a maximum of 16 characters (A–Z, 0–9, and _), not including the delimiter (. or [ ]). Valid ranges for constants used to reference bits are 0–15 for integers and 0–31 for double integers.

IMPORTANT

You can force only simple variables. You cannot force element-indexed or bit-indexed variables. For example, you cannot force variables like: vat.13, array_var[11], array_var[index_name], array_var[11].12 or array_var[index_name].bit_name.

A.3 Global and Local Variables (Scope)

A property of a variable is its scope. A variable's scope can be either local or global.

A.3.1 Local Variables

Local variables are those that can only be used in the program in which they are defined. No other programs can reference them. If you type in the first letter of a variable name using lower case, the default scope will be local. The names of local variables appear in lower case.

A.3.2 Global Variables

Global variables can be referenced by ladder, Control Block, or BASIC programs in a rack. These variables can refer to memory locations, physical I/O locations, or network locations. Global memory variables can be of any data type supported by the Editor. If you type in the first letter of a variable name using upper case, the default scope will be global.

Global variables used for physical I/O must be simple variables. They can be either Boolean, integer, or double integer.

Global I/O variables representing this:	**Can be:**
inputs	read but not written to
outputs	read **or** written to

Within BASIC or Control Block programs, use the statement COMMON to access global variables.

A.4 Arrays

Arrays in ladder programs can have only one dimension. An item within an array is called an element.

This type of array:	Can have a maximum of:
integer and double integer	65,536 elements
Boolean	16,000 elements

The maximum size of a local array or global array when no Common Memory module is present is limited by the amount of available application memory. If a Common Memory module is present, a global array is limited by the amount of memory available on the module.

The first element within the array is always 0. Therefore, if an array had 65,536 elements, they would be numbered 0–65,535. You define the size of an array by entering a value in Maximum Array Index located within the Variable Properties tab.

The maximum array index is the number that represents the last element within an array variable. When you enter a new array variable in an instruction parameter, the maximum array index is automatically defined as the element number you entered as part of the variable. Should you need a larger array, change the Maximum Array Index field of the variable's property sheet. Remember array elements are numbered starting at 0.

To specify the maximum array index for an array variable using the Variable Properties

Step 1. Select the instruction that uses the array variable.

Step 2. Access the variable properties by doing either of the following:

- From the File menu, choose Properties and then the Variables tab.
- Press the right mouse button, and choose Properties from the pop-up menu and then the Variables tab.

Step 3. Select the array variable name from the variable list.

Step 4. In the Maximum Array Index field, enter the number of the last element. The largest number you can enter is 65,535 (65,535 = space reserved for 65,536 elements).

Step 5. Click OK.

Tip

To increase the maximum array index, you can also enter the array variable with a greater index. For example, the variable *A[100*] has 101 elements. If you need a larger index, you can enter *A[200]*. Variable *A* now has 100 more elements than before.

A.5 Constants

Use constants to specify:

- unchanging values within ladder instruction input parameters
- an element of an array (array[2])
- a bit within an integer or double integer variable (pump.15)
- a bit within an element of an integer or double integer array (array[2].31)

You can include the plus (+) and minus (–) signs for constants entered in the ladder instruction parameters.

IMPORTANT

To enter a hexadecimal value that begins with a letter as a ladder instruction input parameter, you must enter the value using a leading 0; end the value with an " h or H." For example, enter 0A5A5H not A5A5H.

A.6 About Initializing Variables

You can choose the value that a variable contains when a program is downloaded to the Processor or while the program is running. This is useful for loading a pre-defined value into a variable for calculations, modifying parameters, or changing a timer's or counter's preset value. You can change the value of a variable and/or its initial value by:

- setting or forcing the variable
- changing a preset value for a timer or counter via inline editing
- changing the initial value via the Variable Properties dialog box

You can choose from the following initialization methods for a variable from its Variable Properties tab:

- No Initialization (No Init.)
- User Specified Value
- Retained Value

A.6.1 About the No Initialization (No Init.) Method

When a variable is configured to use no initialization, the value it contains is determined by the buffering of inputs and outputs and by program execution. For global and local variables, the default initialization is No Initialization. Local and global variables defined as using no initialization are set to 0 when the program in downloaded to the Processor. Global outputs are not cleared at the start of the first scan.

This table explains the value of variables under specific conditions:

<table>
<tr><th rowspan="2">Ladder Program Condition</th><th colspan="3">Variable Type</th></tr>
<tr><th>Non-volatile global</th><th>Volatile global</th><th>Local</th></tr>
<tr><td>Power fail</td><td rowspan="2">values retained, unless battery backup fails</td><td rowspan="2">values lost and set to zero</td><td rowspan="3">values retained</td></tr>
<tr><td>Stop-All</td></tr>
<tr><td>Program stopped</td><td colspan="2">values unchanged by operation; programs that are still running may cause values to change</td></tr>
</table>

Ladder Program Condition	Variable Type		
	Non-volatile global	Volatile global	Local
Program stopped and individually restarted	values unchanged by operation; programs that are still running may cause values to change		values retained, unless the logic within the program writes to variables
Program stopped and the configuration individually reloaded	values are lost; values are zero when the configuration is installed		values retained
Program stopped and configuration and programs individually reloaded	values are lost; values are zero when the configuration is installed		values are lost; values are zero when the program is installed
Stop-All occurred and configuration and programs reloaded	values are lost; values are zero when the configuration is installed		

A.6.2 About the User Specified Initialization Method

When a variable is configured to use a User Specified value, the value defined as the initial value is written to the variable at the start of the first scan. If the variable's initial value is changed via inline editing, forcing, setting, or a program's execution, this new value is not retained when the program is stopped and rerun. The value written to the variable each time the program is placed into run is always that specified in the Initial Value field of the Variable's property sheet. Choose User Specified initialization when you want a variable to use the initial value you defined every time the program is placed into run.

IMPORTANT
Do not use user-specified initialization for a global I/O variable being used as an input. If you do, a bus error (error 31) occurs when you try to run the task.

A.6.3 About the Retained Value Initialization Method

When a variable is configured to use a Retained Value, the value defined as the initial value is written to the variable at the start of the first scan. If the variable's initial value is changed via inline editing, forcing, setting, or through the Variable Properties dialog box, the value the variable contains when program execution stops is the one that is written to the variable when the program is rerun. Retained Value initialization is the default initialization type for timers and counters. An initial value of a variable configured to use Retained Value initialization can be changed by any ladder program. For example, you can change a timer or counter preset value by using ladder logic. Initial values changed online can be permanently installed only by saving the program from the Processor. Choose Retained Value initialization when you want a variable to be initialized with the value it had at the time the program was stopped. Make sure that you specify global variables using Retained Value initialization as non-volatile variables.

IMPORTANT
Do not use retained-value initialization for global I/O variables. Doing so causes an error when the program is verified.

A.6.4 About Initializing Timer and Counter Variables

You can define the initialization mode for timer and counter preset variables. Choose between a User Specified Value or Retained Value. The Retained Value is the default initialization mode. You cannot define an initialization method for the Elapsed timer element or the Current counter element. These elements are usually reset to 0 at the start of program scan, unless the timer or counter data structure is defined as global and non-volatile. If you reference a timer or counter element in a program whose data structure is not used in a timer or counter instruction within the same program, the variable uses the No Initialization method. You cannot change it to User Specified Value or Retained Value. However, if you later add that timer or counter data structure on a newly added timer or counter instruction, the initialization type of the timer or counter elements becomes Retained Value. You can then choose either a Retained Value or User Specified Value initialization. The value contained in the Initial Value field is the timer's or counter's preset value.

A.6.5 About Initializing Arrays

You can define an initial value for individual elements of an array. The list of available elements and their current values appears in the Array Index list box.

A.6.6 Defining the Type of Initialization To Use for a Variable

You can define how a variable is initialized when a program is downloaded to the Processor and put into run. This is useful for loading a pre-defined value into a variable for calculations, modifying values for parameters, or changing a timer's or counter's preset value. Use the Variable Properties dialog box for the variable whose initialization method you want to change.

To define the type of initialization to use for a variable

Step 1. In an offline program, select the instruction containing the variable whose initialization method you want to change.

Step 2. From the File menu, choose Properties. The Instruction Properties dialog box is displayed.

Step 3. Choose the Variable Properties tab, and select the variable whose initialization method you want to change.

Step 4. Choose the variable initialization method you want to use. Choose from No Init. (No Initialization), User Specified Value, or Retained Value.

Step 5. If you have chosen User Specified Value or Retained Value, define an initial value.

Step 6. Accept the change by clicking OK or Apply.

Step 7. Save the program, and reload it to the Processor.

Tip
You can change the initial value while editing the program offline or online.

A.6.7 Defining the Initial Value of a Variable

You can define the value that a variable contains when a program is downloaded to the Processor or placed into run. You can also define an initial value for individual elements of an array. The list of available elements and their current values appears in the variable's Variable Properties tab.

The initial value applies to User Specified Value or Retained Value initialization.

To define the initial value of a variable

Step 1. In an offline or online program, select the instruction containing the variable whose initial value you want to change.

Step 2. From the File menu, choose Properties. The Instruction Properties dialog box is displayed.

Step 3 Choose the Variable Properties tab, and select the variable whose initial value you want to change.

Step 4. Make sure that the selected initialization method is the one you want to use.

Step 5. Use the following table to determine your next step:

If the variable is:	Do the following:
not an element of an array	• In the Initial Value field, enter a value that you want to use as the initial value. A value of 0 is the default value. In the case of a timer or counter data structure, the value in the Initial Value field is the timer's or counter's preset value. A preset value cannot be a negative number.
an element of an array	Step A. In the Array Index list box, select the element whose initial value you want to change. The element's current value is displayed next to the array element. Step B. In the Initial Value field, enter a value that you want to use as the initial value. Zero is the default value.

Step 6. To accept the change, click OK or Apply.

When changing the initial value of a variable using Retained Value initialization during an online editing session, a dialog box prompts you to decide if the changed value should be sent to the rack. Choosing Yes immediately sends the change to the rack. The variable's value is immediately changed. When you save the program from the Processor, the newly changed initial value replaces the original value. Choosing No cancels the change.

Any change made to the value of a variable using User Specified Value initialization during an online editing session does not take effect until the program is stopped and re-run.

Tip
You can enter initial values in hexadecimal. To enter a hexadecimal value that begins with a letter, you must enter the value using a leading 0; end the value with an " h or H." For example, enter 0A5A5H, not A5A5H.

Appendix B

Using Timer Variables in BASIC Programs

You can use timer elements in BASIC programs. To do this, you must declare the timer as a global, double integer array. For example, COMMON TIMER! (4). The elements of the timer data structure can be accessed as follows:

Element:	Description:
0	Elapsed
1	Reserved for system
2	TPreset
3	bit 15 Q bit 23 T bit 31 TR
4	Reserved for system

WARNING

DO NOT MODIFY TIMER ELEMENTS 0, 1, 3, AND 4 FROM A BASIC PROGRAM. THIS WILL CAUSE THE TIMER TO OPERATE INCORRECTLY, RESULTING IN UNPREDICTABLE MACHINE OPERATION.

Within a BASIC program, you may want to:

- determine if a timer has expired
- change a timer's preset

To change the preset value by using a BASIC statement

- Enter a statement in this format:

  NAME!(2)=new_preset_value

  where NAME! is the name of the global timer variable whose preset you want to change

The value of 2 is the location of the element TPreset within a Timer data type.

Example:

TIMER1!(2)=2000

This statement sets the preset value for the variable TIMER1! to 2000.

To determine if a timer has expired by monitoring element Q in a BASIC program

- Use the BASIC function BIT_SET@ to test the value of Q.

Example:

IF BIT_SET@(TIMER1!(3),15) THEN 250

This statement examined bit 15 of element 3 in the global variable TIMER1.

Appendix C

Using Counter Variables in BASIC Programs

You can use counter elements in BASIC programs. To do this, you must declare the counter as a global, double integer array. For example, COMMON COUNTER!(4). The elements of the counter data structure can be accessed as follows:

Element:	Description:	
0	Current	
1	CPreset	
2	Reserved for system	
3	bit 7	LD
	bit 15	CR
	bit 23	QD
	bit 31	QU
4	Reserved for system	

WARNING

DO NOT MODIFY TIMER ELEMENTS 0, 1, 3, AND 4 FROM A BASIC PROGRAM. THIS WILL CAUSE THE TIMER TO OPERATE INCORRECTLY, RESULTING IN UNPREDICTABLE MACHINE OPERATION.

Within a BASIC program you may want to:

- change a counter's preset
- use the Current element of a counter for data display, logic sequencing, or other purposes
- monitor the status of a counter

To change the preset value by using a BASIC statement

- Enter a statement in this format:
  name!(1)=new_preset_value
  where NAME! is the name of the counter variable

The value of 1 is the location of the element CPreset within a Counter data type.
Example:
COUNTER1!(1)=50
This statement is setting the preset value for the global variable COUNTER1 to 50.

To specify the Current element of a counter in a BASIC statement

- Enter the variable in this format:
  NAME!(0)
  where NAME is the name of the counter variable

The value of 0 is the location of the element Current within a Counter data type.

To determine if a counter has reached its preset by monitoring element QU in a BASIC program

- Use the BASIC function BIT_SET@ to test the value of QU.

  Example:

  IF BIT_SET@(COUNTER1!(3),31) THEN 250

  This statement examined bit 31 of element 3 in the global variable COUNTER1.

Appendix D

Using the Pre-Defined (Reserved) Ladder Language Variables

The Editor contains pre-defined variables that you can use in an individual ladder program to:

- execute logic based on a Processor scan
- specify how to handle error conditions
- check scan time execution

The pre-defined ladder language variables are local variables, which you can use for ladder instruction parameters. Their names are reserved and appear in the choices offered by the Variable Smart Matching option.

D.1 Using the Pre-Defined Program Scan Variables

Use the following Boolean variables to execute logic based on the Processor's scan.
Only use these variables for input parameters (read only):

• first_scan	Use this variable to execute logic during a program's first scan. This variable is true during the initial scan of the ladder rung and false during all other scans.
• second_scan	Use this variable to execute logic during a program's second scan. This variable is true during the second scan of the ladder rung and false during all others.
• last_scan	Use this variable to execute logic during a program's last scan. This variable is true on the final pass of the ladder rung after you have selected TASK STOP. This variable is not set when a STOP-ALL error occurs. To use the last_scan variable, your program scan must be less than 0.5 s. **IMPORTANT:** Do not use the last_scan variable in an event-driven program. Because these programs are based on the occurrence of an event, the last_scan variable may never be executed when used in an event-driven program.

D.2 Using the Pre-Defined Error Handling Variables

Use the following Boolean variables to help you handle error conditions. Use error_eno and no_error_log for output (read and write) parameters. You can use task_error as either an input (read only) or output (read and write) parameter:

• task_error	This variable is set true whenever an error is found. Monitor the bit to see if an error occurs during execution and clear it by using the ladder logic. This bit is set true even if errors are not being logged.
• error_eno	Use this bit to determine the value ENO outputs will have if the instruction has an error. The default value is false, which disables any instructions that are connected to the ENO output, possibly making math expressions incomplete. When you set error_eno true, you can continue the execution of the logic connected to the ENO output even if the instruction block had an error. This variable can be changed during ladder logic program execution.
• no_error_log	Set this variable true to prevent errors from being entered into the program error log or being seen by the rung monitor. Only one error is logged for each instruction per program scan; however, you may want to prevent the errors encountered on certain instructions from being entered into the error log. The default state for the no_error_log is false. You can suppress error messages for a group of rungs by changing this variable during program execution. STOP-ALL errors and parameter limit errors for AR1, AR2, and ARC instructions are still inserted into the error log when the no_error_log variable is true.

D.3 Using the Pre-Defined Ladder Execution Time Variables

Use the following double integer parameters to help you check and monitor the program's execution time. Only use these variables as input parameters.

• task_usec_max	Use this variable to monitor the maximum execution time (in μs) for the current program. When this variable is defined in a ladder program.
• task_usec_now	Use this variable to monitor the latest execution time (in μs) of the current program. When this variable is defined in a ladder program. The execution time is the real "clock" time it took the program to run from start to finish. It includes the execution time for higher priority programs and interrupt service routines if they run while your program is running.

To reset these times, write a value of 0 into the variable.

Appendix E

Ladder Instruction Error Code Cross-Reference

Run-time errors are reported as follows:

- For block instruction errors, the error code is displayed to the right of the ENO parameter, while the error code and accompanying text message appears in the Error Log.
- For relay instructions, the error code and the text message appears in the Error Log.

You can access the Error Log from Program Properties.

E.1 Error Codes 3001-3010

Error Code:	Text Description:	How To Correct the Error:
3001	Cannot divide by zero	Define the divisor of the DIV, MDV, or MOD instruction to be a value other than zero.
3002	The result of the arithmetic calculation is too large for Out	Use smaller values or re-arrange the calculation so that errors do not occur.
3003	The result is larger than what Out's data type supports	Specify a larger data type for Out. For example, if you are using integers, specify the data type as a double integer.
3004	Minimum was greater than Maximum	For the LIMIT instruction, make sure the value for Mx is larger than that for Mn.
3005	An illegal BCD digit was found	In the BCD_TO instruction, use a valid BCD value for In.
3006	Tried to convert a negative value to BCD	In the TO_BCD instruction, use a positive value for In.
3007	Label does not exist	Add a LABEL instruction that uses the same name as referenced on a JMP instruction.

Error Code:	Text Description:	How To Correct the Error:
3008	The number of bits to rotate is negative	For the rotate instructions, make sure that N is within the allowable range for the data type used for In.
3009	The number of bits to rotate is too large	For the rotate instructions, make sure that N is within the allowable range for the data type used for In.
3010	The bit number is negative	Make sure the bit number is positive.

E.2 Error Codes 3011-3020

Error Code:	Text Description:	How To Correct the Error:
3011	The bit number is too large	Make sure the bit is within the appropriate range.
3012	The array index is negative	For parameters that accept array variables, specify a valid array element.
3016	The array index is too large	For parameters that accept array variables, specify a valid array element.
3020	The value in Length_Out is too large for array block's Out	Make sure Length_In and Length_Out are within the range of the array.

E.3 Error Codes 3021-3030

Error Code:	Text Description:	How to Correct the Error:
3021	The value in Length1 is too large for array block's In1	Make sure the lengths are within the range of the array.
3022	The value in Length2 is too large for array block's In2	Make sure the lengths are within the range of the array.
3023	The value in Length_Out is less than or equal to 0	Make sure Length_In and Length_Out are within the range of the array.
3024	The value in Length1 is less than or equal to 0	Make sure the lengths are within the range of the array.
3025	The value in Length2 is less than or equal to 0	Make sure the lengths are within the range of the array.
3026	Array's length inputs are greater than 1 but are not equal	For the AR1 instruction, make sure Length_In and Length_Out are within the range of the array.
3027	The Elems/Scan input is less than 0	Make sure the value is at least 0.
3028	Illegal slot number of an IOR or IOW instruction is selected	Correct the slot and register or address parameters used in the instruction.

Error Code:	Text Description:	How to Correct the Error:
3029	An illegal register of an IOR or IOW instruction is selected	Correct the slot and register or address parameters used in the instruction.
3030	An illegal option of an IOR or IOW instruction is selected	Correct the number in the option field. The valid range is 1 to 4.

E.4 Error Codes 3031-3035

Error Code:	Text Description:	How To Correct the Error:
3031	Cannot take the square root of a negative number	Make sure the input for the SQRT operation is not a negative number.
3032	The Length input is negative	For the MVB instruction, specify a value within the appropriate range for the input in error.
3033	The Length input is greater than 32	For the MVB instruction, specify a value within the appropriate range for the input in error.
3034	Length_In is larger than the In input	Make sure Length_In and Length_Out are within the range of the array.
3035	Length_In is less than or equal to 0	Make sure Length_In and Length_Out are within the range of the array.

Appendix F

AutoMax Enhanced Ladder Language Execution Times and Memory Usage for AutoMax 7010 & 6011

Unless otherwise noted in the table or implied by the instruction itself, all block instruction execution times are based on non-indexed, 16-bit variables. Relay instruction execution times are based on simple Boolean variables. Using arrays increases the execution time of the instruction.

Category	Code	Title	7010 Execution Time (μs)		6011 Execution Time (μs)		Bytes of Memory
			True	False	True	False	
Relay	NOI	Normally Open Contact	0.4	0.4	1.5	1.5	6
	NCI	Normally Closed Contact	0.4	0.4	1.5	1.5	6
	PTI	Positive Transition Contact	0.8	0.8	3.0	3.0	6
	NTI	Negative Transition Contact	0.8	0.8	3.0	3.0	6
	ATI	Always True Contact	0.2	0.2	0.5	0.5	5
	AFI	Always False Contact	0.2	0.2	0.5	0.5	5
	CO	Coil	4.5	4.5	18.0	18.0	8
	SCO	Set (Latch) Coil	4.5	1.6	18.0	6.8	10
	RCO	Reset (Unlatch) Coil	4.5	1.6	18.0	6.8	10

Appendix F
(continued)

Category	Code	Title	7010 Execution Time (µs)		6011 Execution Time (µs)		Bytes of Memory
			True	False	True	False	
Branch	LNET	Start of Ladder Network	1.3	1.3	5.3	5.3	16
	BST	Branch Start	0.3	0.4	0.8	1.3	2
		(with ABND)	0.6	0.6	2.0	2.0	4
	NXB	Next Branch	0.3	0.4	0.8	1.3	2
		(with ABND)	0.8	0.8	2.5	2.5	4
	BND	Branch End	0.0	0.0	0.0	0.0	0
	ABND	Always Branch End	0.6	0.6	2.0	2.0	5
Counter	CTUD	Count Up	8.2	3.1	36.0	12.5	12
		Count Down	8.2	--	36.0	--	--
		Reset	6.7	--	29.3	--	--
		Load	7.2	--	32.0	--	--
Timers	TON	Timer On Delay	7.6	3.6	36.8	17.0	10
	TOF	Timer Off Delay	3.8	7.6	18.0	36.8	--
	TP	Timer Pulse	7.0	4.2	35.5	19.3	--
	RTO	Retentive Timer On	7.5	3.9	36.8	17.5	--
		Reset	4.1	--	19.3	--	--

Appendix F
(continued)

Category	Code	Title	7010 Execution Time (μs)		6011 Execution Time (μs)		Bytes of Memory
			True	False	True	False	
Compare	EQ2	2-Input Equal	5.4	1.9	21.0	7.3	17
	GE2	2-Input Greater or Equal	5.4	1.9	21.0	7.3	17
	GT2	2-Input Greater	5.4	1.9	21.0	7.3	17
	LE2	2-Input Less or Equal	5.4	1.9	21.0	7.3	17
	LT2	2-Input Less	5.4	1.9	21.0	7.3	17
	NE2	2-Input Not Equal	5.4	1.9	21.0	7.3	17
	EQ3	3-Input Equal	8.2	4.4	31.3	16.5	24
	GE3	3-Input Greater or Equal	8.2	4.4	31.3	16.5	24
	GT3	3-Input Greater	8.2	4.4	31.3	16.5	24
	LE3	3-Input Less or Equal	8.2	4.4	31.3	16.5	24
	LT3	3-Input Less	8.2	4.4	31.3	16.5	24
	NE3	3-Input Not Equal	8.5	4.4	32.5	16.5	24
	LIMIT	Clamp at Limits	9.7	4.4	36.5	16.5	28
	MSK	Mask Compare	8.0	4.4	31.0	16.5	24

Appendix F
(continued)

Category	Code	Title	7010 Execution Time (µs)		6011 Execution Time (µs)		Bytes of Memory
			True	False	True	False	
Compute	ABS	Absolute Value	5.7	1.9	22.0	7.3	17
	ADD2	2-Input Add	5.7	1.9	21.8	7.3	19
	ADD3	3-Input Add	10.0	4.4	38.3	16.5	26
		(32-bit)	14.3	--	59.5	--	--
	DIV	Divide	9.5	1.9	43.5	7.3	21
		(32-bit / 32-bit)	10.8	--	230.0	--	--
	MOD	Modulo	9.8	1.9	45.0	7.3	21
	MUL	Multiply	6.8	1.9	27.0	7.3	21
		(32-bit * 32-bit)	8.3	--	101.8	--	--
	MDV	Multiply Divide	13.0	4.4	57.3	16.5	26
		((32 * 32 = 64-bit) / 32-bit)	15.4	--	674.5	--	--
	NEG	Negate	5.5	1.9	21.3	7.3	17
	SQRT	Square Root	31.8	1.9	113.5	7.3	17
		(32-bit)	53.2	--	233.8	--	--
	SUB	Subtract	5.8	1.9	22.3	7.3	19

Appendix F
(continued)

Category	Code	Title	7010 Execution Time (µs)		6011 Execution Time (µs)		Bytes of Memory
			True	False	True	False	
Logical	AND	And	5.6	1.9	21.0	7.3	19
	NOT	Not	5.2	1.9	20.0	7.3	17
	OR	Or	5.6	1.9	21.0	7.3	19
	XOR	Exclusive Or	5.6	1.9	21.0	7.3	19
Convert	BCD_TO	Binary to BCD	13.5	1.9	64.5	7.3	17
		(32-bit to 8-digit)	20.9	--	115.0	--	--
	TO_BCD	BCD to Binary	13.6	1.9	76.3	7.3	17
		(8-digit to 32-bit)	23.4	--	147.0	--	--
Bit Move	MVB	Move Bits	37.9	29.2	157.8	114.8	40
	MOVE	Move Source to Destination	5.1	1.9	19.5	7.3	17
	MVM	Masked Move	6.8	1.9	26.3	7.3	21
Array	AR1	Unary Array Operation	--	5.5	--	21.8	183
		Single Scan (Length < 32768)	81.2 + (AR1 operation)	--	350.0 + (AR1 operation)	--	--
		Single Scan (Length > 32767)	81.2 + (AR1 operation)	--	555.0 + (AR1 operation)	--	--

Appendix F
(continued)

Category	Code	Title	7010 Execution Time (µs)		6011 Execution Time (µs)		Bytes of Memory
			True	False	True	False	
Array (Continued)	AR1 (Continued)	1st Scan Initialization (Length < 32768)	67.6	--	295.0	--	--
		1st Scan Initialization (Length > 32767)	67.6	--	500.0	--	--
		2nd through Nth Scan	20.0 + (AR1 operation)	--	81.5 + (AR1 operation)	--	--
		(AR1 Operation)					
		NOT Scan	1.7 * words	--	8.5 * words	--	--
		ABS Scan	2.0 * words	--	10.0 * words	--	--
		NEG Scan	1.9 * words	--	9.3 * words	--	--
		SQRT Scan	29.3 * words	--	107.3 * words	--	--
		MOV Scan	1.6 * words	--	8.0 * words	--	--
	AR2	Binary Array Operation	--	5.5	--	21.8	223
		Single Scan (Length < 32768)	100.9 + (AR2 operation)	--	435.0 + (AR2 operation)	--	--

Appendix F
(continued)

Category	Code	Title	7010 Execution Time (µs)		6011 Execution Time (µs)		Bytes of Memory
			True	False	True	False	
Array (Continued)	AR2 (Continued)	Single Scan (Length > 32767)	100.9 + (AR2 operation)	--	635.0 + (AR2 operation)	--	--
		1st Scan Initialization (Length < 32768)	85.0	--	366.5	--	--
		1st Scan Initialization (Length > 32767)	85.0	--	570.3	--	--
		2nd through Nth Scan	22.3 + (AR2 operation)	--	89.8 + (AR2 operation)	--	
		(AR2 operation)					
		AND,OR,XOR Scan	2.0 * words	--	9.5 * words	--	--
		ADD,SUB Scan	2.2 * words	--	10.3 * words	--	--
		MUL Scan	3.4 * words	--	16.8 * words	--	--
		DIV Scan	4.6 * words	--	26.8 * words	--	--
	ARC	Array Compare	--	5.5	--	21.8	186
		Single Scan (Length < 32768)	79.1 + (Compare Scan)	--	340.0 + (Compare Scan)	--	--

Appendix F
(continued)

Category	Code	Title	7010 Execution Time (µs)		6011 Execution Time (µs)		Bytes of Memory
			True	False	True	False	
Array (Continued)	AR2 (Continued)	Single Scan (Length > 32767)	79.1 + (Compare Scan)	--	545.0 + (Compare Scan)	--	--
		1st Scan Initialization (Length < 32768)	67.8	--	290.3	--	--
		1st Scan Initialization (Length > 32767)	67.8	--	494.0	--	--
		2nd through Nth Scan	17.7+	--	72.8+	--	--
		Compare Scan	0.8 * words	--	3.3 * words	--	--
	ASU	Array Shift Up	17.3 + 0.7 * words	12.7	68.8 + 3.3 * words	49.3	33
	ASD	Array Shift Down	17.3 + 0.7 * words	12.7	68.8 + 3.5 * words	49.3	33
Shift	SL	Shift Left	7.5	2.6	27.3	10.0	19
		(Boolean Array)	6.2 + 1.3 * bytes	3.0	24.3 + 4.3 * bytes	11.8	23

Appendix F
(continued)

Category	Code	Title	7010 Execution Time (µs)		6011 Execution Time (µs)		Bytes of Memory
			True	False	True	False	
Shift (Continued)	SR	Shift Right	7.5	2.6	27.3	10.0	19
		(Boolean Array)	7.7 + 1.3 * bytes	3.0	31.3 + 4.5 * bytes	11.8	23
	ROL	Circular Rotate Bits Left	8.3	2.6	32.5	10.0	21
		(Boolean Array)	11.3 + 1.6 * bytes	3.0	47.3 + 9.8 * bytes	11.8	29
	RL	Circular Rotate Bits Left	9.0	2.6	35.3	10.0	21
		(Boolean Array)	10.5 + 1.6 * bytes	3.0	44.5 + 9.8 * bytes	11.8	29
	ROR	Circular Rotate Bits Right	8.3	2.6	32.5	10.0	21
		(Boolean Array)	10.6 + 1.5 * bytes	3.0	46.0 + 9.3 * bytes	11.8	29
	RR	Circular Rotate Bits Right	9.0	2.6	35.3	10.0	21
		(Boolean Array)	11.4 + 1.5 * bytes	3.0	48.8 + 9.3 * bytes	11.8	29
Control	SET	Set Event	76.0	2.8	311.0	11.5	11
	JMP	Jump	2.3	1.4	9.0	5.5	9
	LBL	Label	0.0	0.0	0.0	0.0	8+name

Appendix F
(continued)

Category	Code	Title	7010 Execution Time (µs)		6011 Execution Time (µs)		Bytes of Memory
			True	False	True	False	
I/O	IOR	Input Read	18.1	4.4	71.5	16.5	28
	IOW	Output Write	31.4	24.3	124.5	95.8	34
	IN	Immediate Input	7.0	3.0	28.0	12.8	12
	OUT	Immediate Output	4.3	2.9	16.8	11.5	12
Program	Fixed	System Overhead	--	117.0	--	479.0	8,650
	Variable	Local Boolean	0.0	--	0.0	--	6+name
		Local Boolean Array	0.0	--	0.0	--	46+name
		Local Integer	0.0	--	0.0	--	12+name
		Local Integer Array	0.0	--	0.0	--	46+name + 2*size
		Local Double Integer	0.0	--	0.0	--	16+name
		Local Double Integer Array	0.0	--	0.0	--	46+name + 4*size
		Local Counter or Timer	0.0	--	0.0	--	80+name
		Global Boolean	--	--		--	10+name
		(Within Another Variable)	0.0	--	0.0	--	6+name

Appendix F
(continued)

Category	Code	Title	7010 Execution Time (µs)		6011 Execution Time (µs)		Bytes of Memory
			True	False	True	False	
Program (Continued)	Variable (Continued)	Global Boolean Array	2.3 + 2.7 * words	--	12.3 + 11.8 * words	--	50+name
		Global Integer	2.6	--	12.3	--	16+name
		(Containing a Boolean)	3.5	--	16.0	--	14+name
		Global Integer Array	0.0	--	0.0	--	54+name
		Global Double Integer	2.6	--	15.5	--	18+name
		(Containing a Boolean)	3.5	--	20.5	--	14+name
		Global Double Integer Array	0.0	--	0.0	--	54+name
		Global Counter or Timer	9.0	--	50.8	--	74+name

Appendix G

AutoMax Enhanced Ladder Language Execution Times and Memory Usage for AutoMax PC3000

Unless otherwise noted in the table or implied by the instruction itself, all block instruction execution times are based on non-indexed, 16-bit variables. Relay instruction execution times are based on simple Boolean variables. Using arrays increases the execution time of the instruction.

Category	Code	Title	PC3000 Execution Time (μs)		Bytes of Memory
			True	False	
Relay	NOI	Normally Open Contact	0.3	0.3	6
	NCI	Normally Closed Contact	0.3	0.3	6
	PTI	Positive Transition Contact	0.6	0.6	6
	NTI	Negative Transition Contact	0.6	0.6	6
	ATI	Always True Contact	0.1	0.1	5
	AFI	Always False Contact	0.1	0.1	5
	CO	Coil	3.6	3.6	8
	SCO	Set (Latch) Coil	3.6	1.3	10
	RCO	Reset (Unlatch) Coil	3.6	1.3	10

Appendix G
(Continued)

Category	Code	Title	PC3000 Execution Time (μs)		Bytes of Memory
			True	False	
Branch	LNET	Start of Ladder Network	1.1	1.1	16
	BST	Branch Start	0.3	0.2	2
		(with ABND)	0.5	0.5	4
	NXB	Next Branch	0.3	0.2	2
		(with ABND)	0.6	0.6	4
	BND	Branch End	0.0	0.0	0
	ABND	Always Branch End	0.5	0.5	5
Counter	CTUD	Count Up	6.8	2.5	12
		Count Down	6.8	--	--
		Reset	5.2	--	--
		Load	5.6	--	--
Timers	TON	Timer On Delay	6.3	3.0	10
	TOF	Timer Off Delay	3.1	6.3	--
	TP	Timer Pulse	5.7	3.3	--
	RTO	Retentive Timer On	6.1	3.1	--
		Reset	3.3	--	--

Appendix G
(Continued)

Category	Code	Title	PC3000 Execution Time (μs)		Bytes of Memory
			True	False	
Compare	EQ2	2-Input Equal	4.4	1.6	17
	GE2	2-Input Greater or Equal	4.4	1.6	17
	GT2	2-Input Greater	4.4	1.6	17
	LE2	2-Input Less or Equal	4.4	1.6	17
	LT2	2-Input Less	4.4	1.6	17
	NE2	2-Input Not Equal	4.4	1.6	17
	EQ3	3-Input Equal	6.7	3.6	24
	GE3	3-Input Greater or Equal	6.7	3.6	24
	GT3	3-Input Greater	6.7	3.6	24
	LE3	3-Input Less or Equal	6.7	3.6	24
	LT3	3-Input Less	6.7	3.6	24
	NE3	3-Input Not Equal	6.9	3.6	24
	LIMIT	Clamp at Limits	7.8	3.6	28
	MSK	Mask Compare	6.6	3.6	24

Appendix G
(Continued)

Category	Code	Title	PC3000 Execution Time (μs)		Bytes of Memory
			True	False	
Compute	ABS	Absolute Value	4.7	1.6	17
	ADD2	2-Input Add	4.7	1.6	19
	ADD3	3-Input Add	8.0	3.6	26
		(32-bit)	11.6	--	--
	DIV	Divide	7.4	1.6	21
		(32-bit / 32-bit)	8.0	--	--
	MOD	Modulo	7.6	1.6	21
	MUL	Multiply	5.4	1.6	21
		(32-bit * 32-bit)	6.2	--	--
	MDV	Multiply Divide	9.9	3.6	26
		((32 * 32 = 64-bit) / 32-bit)	11.2	--	--
	NEG	Negate	4.5	1.6	17
	SQRT	Square Root	21.7	1.6	17
		(32-bit)	34.4	--	--
	SUB	Subtract	4.8	1.6	19

Appendix G
(Continued)

Category	Code	Title	PC3000 Execution Time (μs)		Bytes of Memory
			True	False	
Logical	AND	And	4.6	1.6	19
	NOT	Not	4.3	1.6	17
	OR	Or	4.6	1.6	19
	XOR	Exclusive Or	4.6	1.6	19
Convert	BCD_TO	Binary to BCD	10.2	1.6	17
		(32-bit to 8-digit)	15.3	--	--
	TO_BCD	BCD to Binary	10.7	1.6	17
		(8-digit to 32-bit)	18.0	--	--
Bit Move	MVB	Move Bits	31.2	24.1	40
	MOVE	Move Source to Destination	4.2	1.6	17
	MVM	Masked Move	5.6	1.6	21
Array	AR1	Unary Array Operation	--	4.5	183
		Single Scan (Length < 32768)	66.4+ (AR1 operation)	--	--
		Single Scan (Length > 32767)	66.4+ (AR1 operation)	--	--
		1st Scan Initialization (Length < 32768)	54.6	--	--
		1st Scan Initialization (Length > 32767)	54.6	--	--

Appendix G
(Continued)

Category	Code	Title	PC3000 Execution Time (μs)		Bytes of Memory
			True	False	
Logical (Continued)	AR1 (Continued)	2nd through Nth Scan	17.1+ (AR1 operation)	--	--
		(AR1 Operation)			
		NOT Scan	1.4 * words		--
		ABS Scan	1.7 * words		--
		NEG Scan	1.6 * words		--
		SQRT Scan	19.7 * words		--
		MOV Scan	1.4 * words		--
	AR2	Binary Array Operation	--	4.5	223
		Single Scan (Length < 32768)	81.8+ (AR2 operation)	--	--
		Single Scan (Length > 32767)	81.8+ (AR2 operation)	--	--
		1st Scan Initialization (Length < 32768)	68.5	--	--
		1st Scan Initialization (Length > 32767)	68.5	--	--
		2nd through Nth Scan	18.6+ (AR2 operation)	--	

Appendix G
(Continued)

Category	Code	Title	PC3000 Execution Time (μs)		Bytes of Memory
			True	False	
Logical (Continued)	AR2	(AR2 operation)		--	
		AND,OR,XOR Scan	1.7 * words	--	--
		ADD,SUB Scan	1.8 * words	--	--
		MUL Scan	2.6 * words	--	--
		DIV Scan	3.4 * words	--	--
	ARC	Array Compare	--	4.5	186
		Single Scan (Length < 32768)	64.1+	--	--
		Single Scan (Length > 32767)	64.1+	--	--
		1st Scan Initialization (Length < 32768)	54.8	--	--
		1st Scan Initialization (Length > 32767)	54.8	--	--
		2nd through Nth Scan	14.6+	--	--
		Compare Scan	0.8 * words		--
	ASU	Array Shift Up	14.0 + 0.7 * words	10.3	33
	ASD	Array Shift Down	14.0 + 0.7 * words	10.3	33

Appendix G
(Continued)

Category	Code	Title	PC3000 Execution Time (μs)		Bytes of Memory
			True	False	
Shift	SL	Shift Left	5.8	2.2	19
		(Boolean Array)	5.0 + 1.2 * bytes	2.5	23
	SR	Shift Right	5.8	2.2	19
		(Boolean Array)	6.2 + 1.2 * bytes	2.5	23
	ROL	Circular Rotate Bits Left	6.6	2.1	21
		(Boolean Array)	8.8 + 1.5 * bytes	2.4	29
	RL	Circular Rotate Bits Left	7.1	2.1	21
		(Boolean Array)	8.1 + 1.5 * bytes	2.4	29
	ROR	Circular Rotate Bits Right	6.6	2.1	21
		(Boolean Array)	8.2 + 1.4 * bytes	2.4	29
	RR	Circular Rotate Bits Right	7.1	2.1	21
		(Boolean Array)	8.8 + 1.4 * bytes	2.4	29

Appendix G
(Continued)

Category	Code	Title	PC3000 Execution Time (μs)		Bytes of Memory
			True	False	
Control	SET	Set Event	--	2.4	11
	JMP	Jump	1.9	1.2	9
	LBL	Label	0.0	0.0	8+name
I/O	IOR	Input Read	14.8	3.6	28
	IOW	Output Write	25.6	20.1	34
	IN	Immediate Input	5.6	2.5	12
	OUT	Immediate Output	3.5	2.4	12
Program	Fixed	System Overhead	--	80	8,650
	Variable	Local Boolean	0.0	--	6+name
		Local Boolean Array	0.0	--	46+name
		Local Integer	0.0	--	12+name
		Local Integer Array	0.0	--	46+name + 2*size
		Local Double Integer	0.0	--	16+name
		Local Double Integer Array	0.0	--	46+name + 4*size
		Local Counter or Timer	0.0	--	80+name
		Global Boolean	--	--	10+name

Appendix G
(Continued)

Category	Code	Title	PC3000 Execution Time (μs)		Bytes of Memory
			True	False	
Program (Continued)	Variable (Continued)	(Within Another Variable)	0.0	--	6+name
		Global Boolean Array	1.8 + 2.2 * words	--	50+name
		Global Integer	2.1	--	16+name
		(Containing a Boolean)	2.8	--	14+name
		Global Integer Array	0.0	--	54+name
		Global Double Integer	2.1	--	18+name
		(Containing a Boolean)	2.8	--	14+name
		Global Double Integer Array	0.0	--	54+name
		Global Counter or Timer	7.4	--	74+name

Appendix H

Glossary

Accept: To approve the edit made to an online program. Rungs that have been added, deleted, or modified must be accepted and verified before they can be downloaded to a Processor.

Bit-indexed variable: A variable referencing a bit within an integer or double integer variable. For example, pump.15 references bit 15 within the integer variable ”pump.”

Commit: To allow the Editor to verify and download changes made to an online program. You must accept online changes before you can commit them. You can commit online changes immediately after you accept them or while the program is in Test Mode.

Data Structure: Data structures contain a collection of Boolean and double integer data and are used for the timer and counter data types.

Element-indexed variable: A variable referencing an element within an array variable. For example, panel[11] references an element 11 within the array variable ”panel.” An element can be a Boolean, integer, or double integer.

Global Variables: Global variables can be referenced by ladder, Control Block, or BASIC programs in a rack. Global variables can refer to memory locations, physical I/O locations, or network locations. Global memory variables can be of any data type supported by the Editor. If you type in the first letter of a variable using upper case, the default scope will be global. The names of global variables appear in upper case.

Local Variables: Local variables are those that can only be used in the program in which they are defined. No other programs can reference them. If you type in the first letter of a variable name using lower case, the default scope will be local. The names of local variables appear in lower case.

Match: As applied to the Resolve Variable Descriptions command, a global variable in a ladder program that uses the same name as one in the Variable Configurator.

For example, the Editor would determine that a global variable called PUMP_STATUS used in a ladder program is a match to a global variable called PUMP_STATUS present in the Variable Configurator.

The data type of a variable is not a factor when determining whether the global variables match.

Path: The directory structure used by the AutoMax Executive is:

drive:\library\system\rack

where:

drive	is the personal computer hard drive where the Executive is stored
library	is the base directory under which all the AutoMax systems are stored
system	is the subdirectory where the system database files are stored
rack	is the subdirectory where all the rack database files and all programs for the rack are stored

The default drive and library name are specified as part of the Setup procedure for the AutoMax Programming Executive software. If you want to create a new library or change the default (selected) library or drive, you must use the Setup procedure.

Pause: Places the Editor in the Paused state so that you can use the Online Task Manager. The programs continue to run in the Processor, but their display in a program window is not updated.

Program: Task. In the Editor, the terms “program” and “task” are synonymous.

Rung status area: The gray-shaded area left of the power rail. When rung numbers, revision marks, or set triggers are displayed, they are located here.

Test Mode: Lets you actively execute rungs, but the changes made to the online program are not permanently installed in the Processor.

Trigger: A trigger is a way to capture or freeze a rung's status while monitoring the program. Once a trigger is set, the rung's status stays frozen on the programming terminal, but the rung continues to run on the CPU.

A

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

For additional information

1 Allen-Bradley Drive
Mayfield Heights, Ohio 44124 USA
Tel: (800) 241-2886 or (440) 646-3599
http://www.reliance.com/automax

www.rockwellautomation.com

Corporate Headquarters
Rockwell Automation, 777 East Wisconsin Avenue, Suite 1400, Milwaukee, WI, 53202-5302 USA, Tel: (1) 414.212.5200, Fax: (1) 414.212.5201

Headquarters for Allen-Bradley Products, Rockwell Software Products and Global Manufacturing Solutions
Americas: Rockwell Automation, 1201 South Second Street, Milwaukee, WI 53204-2496 USA, Tel: (1) 414.382.2000, Fax: (1) 414.382.4444
Europe/Middle East/Africa: Rockwell Automation SA/NV, Vorstlaan/Boulevard du Souverain 36, 1170 Brussels, Belgium, Tel: (32) 2 663 0600, Fax: (32) 2 663 0640
Asia Pacific: Rockwell Automation, 27/F Citicorp Centre, 18 Whitfield Road, Causeway Bay, Hong Kong, Tel: (852) 2887 4788, Fax: (852) 2508 1846

Headquarters for Dodge and Reliance Electric Products
Americas: Rockwell Automation, 6040 Ponders Court, Greenville, SC 29615-4617 USA, Tel: (1) 864.297.4800, Fax: (1) 864.281.2433
Europe/Middle East/Africa: Rockwell Automation, Brühlstraße 22, D-74834 Elztal-Dallau, Germany, Tel: (49) 6261 9410, Fax: (49) 6261 17741
Asia Pacific: Rockwell Automation, 55 Newton Road, #11-01/02 Revenue House, Singapore 307987, Tel: (65) 6356-9077, Fax: (65) 6356-9011

Publication J2-3094-4 - April 1998