Logic Developer - PC

Ladder Logic Instructions

Ladder logic instructions are the building blocks a ladder program is composed of. You insert instructions onto rungs to create simple executable units of logic. Each instruction performs an operation on variables defined for the target the ladder program is associated with.

Tip: All available instructions are contained in the **Ladder** drawer of the Toolchest. Drag these instructions to a rung in your ladder program.

Ladder logic instructions are grouped functionally (that is, according to the type of operation performed). The instruction groups are:

- Advanced Math
- Allen-Bradley RIO
- ASCII Communications
- Basic Math
- Bit Shift/Rotate
- Bitwise Logic
- Coils
- Comparisons
- Contacts
- Conversions
- Copy
- Counters
- DeviceNet
- Increment/Decrement
- Interbus-S
- Process Control

- Program Flow
- Sequencer
- Timers

Advanced Math Instructions

These instructions perform trigonometric, logarithmic and other advanced mathematical operations.

Absolute Value (ABS)

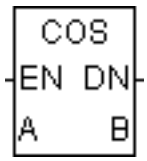
Arc Cosine (ACOS)

Arc Sine (ASIN)

Arc Tangent (ATAN)

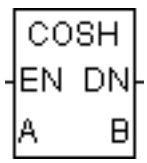
Arc Tangent of Y/X (ATAN2)

Base 10 Logarithm (LOG)



Cosine (COS)

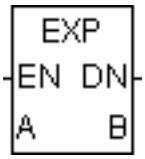
Degrees to Radians (D2R)



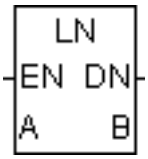
Hyperbolic Cosine (COSH)

Hyperbolic Sine (SINH)

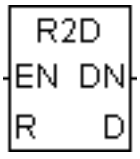
Hyperbolic Tangent (TANH)



Natural Exponent (EXP)



Natural Logarithm (LN)



Radians to Degrees (R2D)

Sine (SIN)

Square Root (SQRT)

Tangent (TAN)

X to the power of Y (EXPT)

Allen Bradley - RIO Instructions

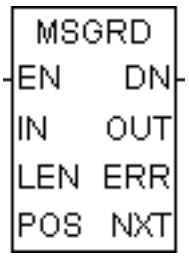
These instructions are used to perform block transfers to and from I/O when an Allen-Bradley RIO I/O driver is configured in your project.

Block Transfer Read (BTR)

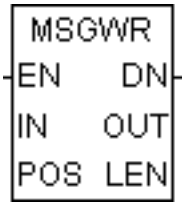
Block Transfer Write (BTW)

ASCII Communications Instructions

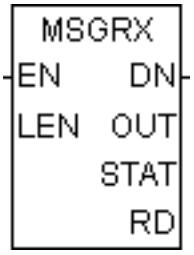
These instructions are used when an ASCII Communications I/O driver is configured for your project. They provide a way to send and receive process data via a serial communications port on the target computer.



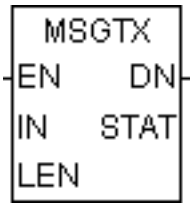
Parse Message Buffer (MSGRD)



Format Message Buffer (MSGWR)



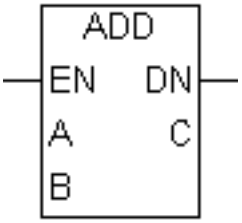
Receive Message Buffer (MSGRX)



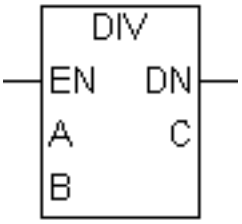
Transmit Message Buffer (MSGTX)

Basic Math Instructions

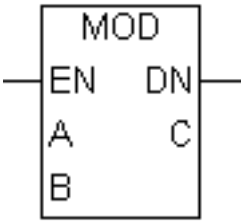
These instructions perform simple arithmetic operations with DINT and/or LREAL parameters.



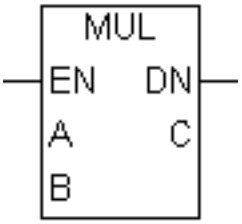
Add (ADD)



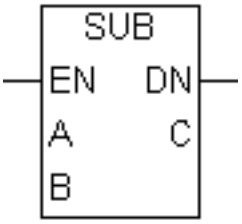
Divide (DIV)



Modulus (MOD)



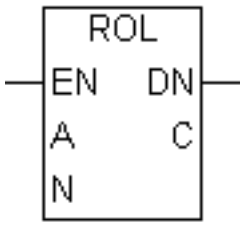
Multiply (MUL)



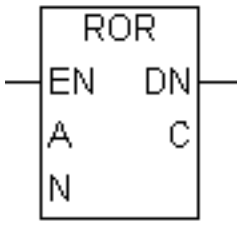
Subtract (SUB)

Bit Shift/Rotate Instructions

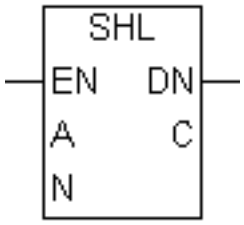
These instructions are used to reposition the bits within the assigned DINT parameters. Operations can also be done on DINT arrays to simulate very large binary numbers.



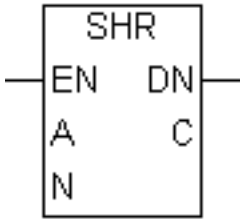
Rotate Left (ROL)



Rotate Right (ROR)



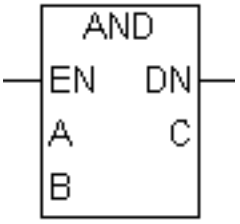
Shift Left (SHL)



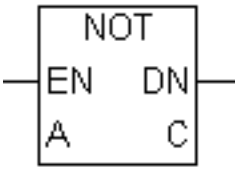
Shift Right (SHR)

Bitwise Logic Instructions

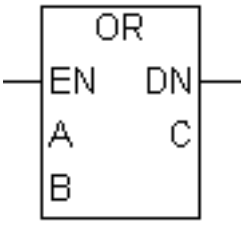
These instructions perform logical (BOOL) operations on DINT parameters. Operations can also be carried out on DINT arrays to simulate very large binary numbers.



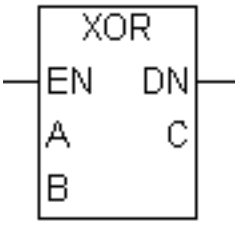
Bitwise AND



Bitwise NOT



Bitwise OR



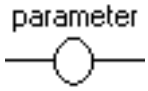
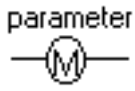
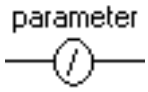
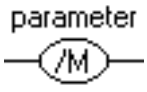
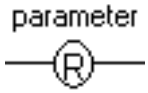
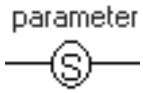
Bitwise XOR

Coil Instructions

These instructions are used to turn ON(1) or OFF(0) the BOOL parameters assigned to them. Typically the assigned parameters are BOOL variables mapped to physical outputs, but internal variables and expressions can also be used.

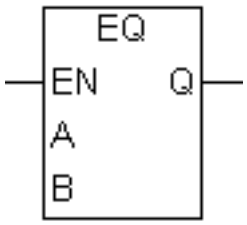
The representation of Coil instructions in logic depends on the retentive state of BOOL variables assigned to them. Coil instructions are represented in logic as:

- **Retentive.** The assigned parameter's value is saved when the Controller is shut down or reset and restored when the Controller is started.
- **Non-retentive.** The assigned parameter's value is lost when the Controller shuts down and set its Initial Value when the Controller starts.

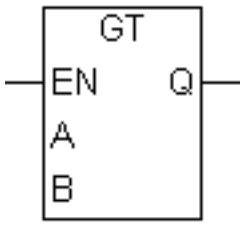
Instruction	Non-Retentive	Retentive
Coil (OUT)		
Negated Coil (NEG)		
Reset Coil (RST)		
Set Coil (SET)		

Comparison Instructions

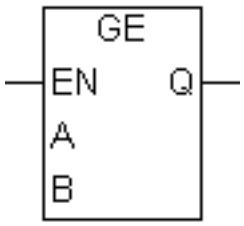
These instructions are used to compare the values of two DINT or LREAL parameters and control the power flow through the logic based on the results.



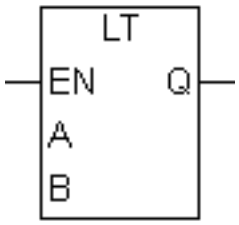
Equal (EQ)



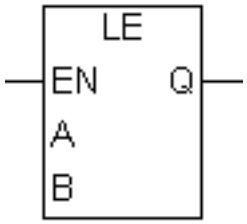
Greater Than (GT)



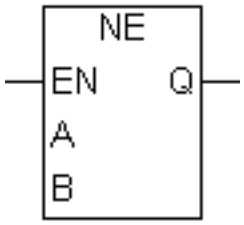
Greater Than or Equal (GE)



Less Than (LT)



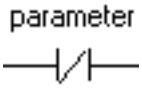
Less Than or Equal (LE)



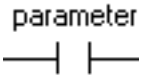
Not Equal (NE)

Contact Instructions

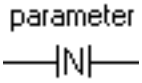
These instructions are used to control the power flow through the logic. They can be thought of as switches controlled by the BOOL parameters assigned to them.



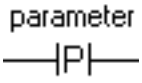
Normally Closed Contact (NC)



Normally Open Contact (NO)



Negative Transition Contact (NT)



Positive Transition Contact (PT)

Copy Instructions

These instructions are used to copy data from one parameter to another, including arrays.

Copy (COPY)

Move (MOV)

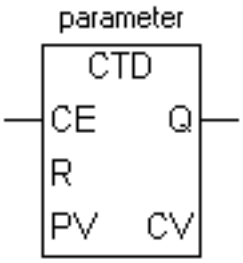
Block Move (BMOV)

Fill Move (FMOV)

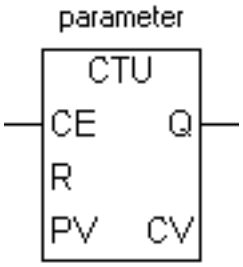
See also the copy instruction in FBD.

Counter Instructions

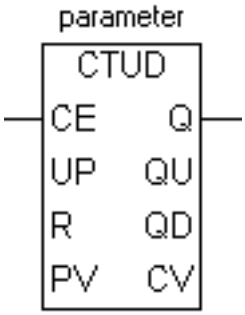
These instructions are used to count events. The parameter assigned is a COUNTER structure variable.



Down Counter (CTD)



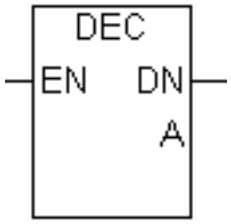
Up Counter (CTU)



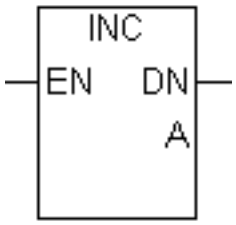
Up/Down Counter (CTUD)

Increment/Decrement Instructions

These instructions are used to add or subtract one from the assigned DINT parameters.



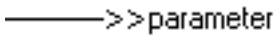
Decrement (DEC)



Increment (INC)

Program Flow Instructions

The following instructions are used to transfer program control (that is, instruction execution) to a rung other than that immediately following the current rung.

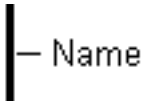


Jump (JMP)

Jump Subroutine (JSR)

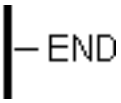
Return

The following components are used to indicate the beginning and/or end of sections of ladder logic.



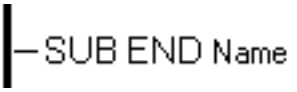
Label

START label



END label

SUB START label



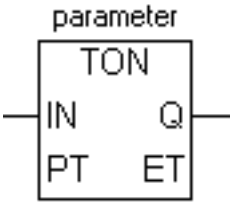
SUB END label

ACT START label

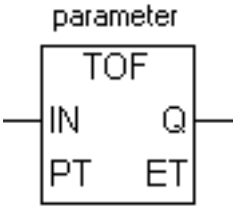
ACT END label

Timer Instructions

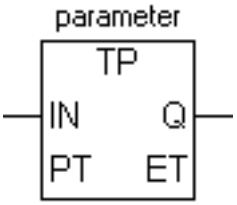
These instructions are used to time events (in milliseconds). The parameter assigned is a TIMER structure variable.



Timer On Delay (TON)



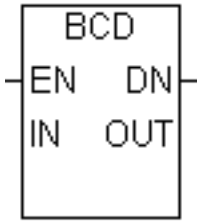
Timer Off Delay (TOF)



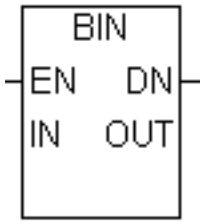
Timer Pulse (TP)

Conversion Instructions

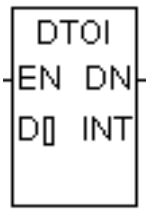
These instructions are used to convert parameters to different binary number formats and data types.



Convert to BCD (BCD)



Convert to BIN (BIN)

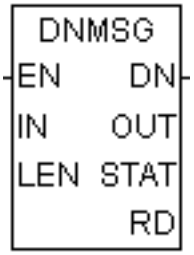


BOOL to DINT (DTOI)

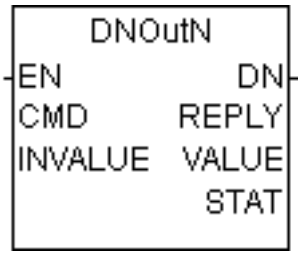
DINT to BOOL (ITOD)

DeviceNet Instructions

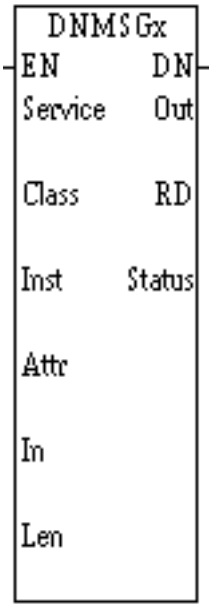
These DeviceNet Message instructions can be used to send and receive explicit messages to I/O when the DeviceNet I/O driver is configured in your project.



DeviceNet Message (DNMSG)



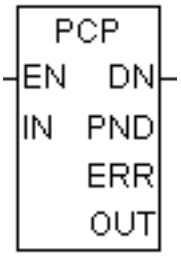
DeviceNet Message (DNOutN)



DeviceNet Message (DNMSGx)

Interbus-S Instructions

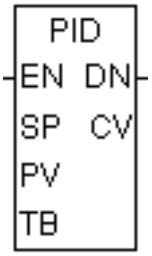
The Interbus-S PCP Message instruction is used to send parameter data to compatible I/O when the Interbus-S I/O driver is configured in your project.



Interbus-S PCP Message (PCP)

Process Control Instructions

The PID (Proportional-Integral-Derivative) instruction is used to implement closed loop control of a specific process.



PID

Sequencer Instructions

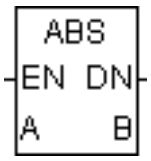
Sequencer instructions work with a group of BOOL I/O points and a DINT sequence array to monitor and control a sequential operation.

Sequencer Input (SQI)

Sequencer Output (SQO)

Sequencer Load (SQL)

Absolute Value (ABS)



The ABS instruction moves the absolute (positive) value of **A** into **B**.

This instruction always passes power.

Parameter Data Types

The following data types can be used with the ABS instruction.

If **A** is a:	then **B** is a:
LREAL	LREAL
DINT	DINT

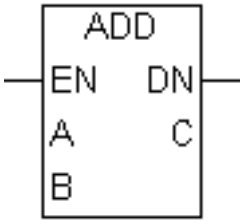
ACT END Label

An ACT END label marks the end of an action and is automatically added to a ladder program when you insert an action. A name uniquely identifies the action the ACT END label belongs to. When execution of an SFC encounters an ACT END label, program control is transferred to the next action specified in a Step instruction or to the Transition following a Step instruction.

ACT START Label

An ACT START label marks the beginning of an action and is automatically added to a ladder program when you insert an action. A name uniquely identifies the action the ACT START label belongs to. When execution of an SFC encounters an ACT START label, program flow picks up at the rung following the label and continues until a ACT END label is encountered.

Add (ADD)



The ADD instruction adds **A** to **B**, placing the sum in **C**.

This instruction always passes power.

Variations

If either **A** or **B** are LREALs (or LREAL constants), the instruction performs a floating-point addition. Otherwise, it performs a faster DINT addition.

Parameter Data Types

The following parameter data types can be used in the ADD instruction:

If **A** is a:	and **B** is a:	then **C** is a:
DINT	DINT	DINT or LREAL
DINT constant	DINT constant	
LREAL	LREAL	
LREAL constant	LREAL constant	
TIME	TIME	TIME
DATE_AND_TIME	TIME	DATE_AND_TIME

Notes

- #Overflow is set if the result **C** is too large for the destination variable. The result of a DINT addition is truncated; the result of a floating-point addition is undefined.
- If either **A** or **B** are LREALs, both are converted to LREALs prior to the addition. The results are placed in **C** and truncated if **C** is a DINT.

Arc Cosine (ACOS)

The ACOS instruction calculates the arc cosine of **A** and stores the result in **B**.

This instruction always passes power.

A must be in the range -1 to +1. The angle is returned in radians, in the range 0 to *pi*.

Parameter Data Types

The following data types can be used with the ACOS instruction.

If **A** is a:	then **B** is a:
DINT or LREAL	LREAL

Arc Sine (ASIN)

The ASIN instruction calculates the arc sine of **A** and stores the result in **B**.

This instruction always passes power.

A must be in the range -1 to +1. The angle is returned in radians, in the range -*pi*/2 to +*pi*/2.

Parameter Data Types

The following data types can be used with the ASIN instruction.

If **A** is a:	then **B** is a:
DINT or LREAL	LREAL

Arc Tangent (ATAN)

The ATAN instruction calculates the arc tangent of **A** and stores the result in **B**.

This instruction always passes power.

The angle is returned in radians, in the range -*pi*/2 to +*pi*/2.

Parameter Data Types

The following data types can be used with the ATAN instruction.

If **A** is a:	then **B** is a:
DINT or LREAL	LREAL

See also ATAN2.

Arc Tangent of Y / X (ATAN2)

The ATAN2 instruction calculates the arc tangent of **Y** / **X**, and stores the result in **Z**.

This instruction always passes power.

The angle is returned in radians, in the range -*pi* to +*pi*.

This function produces accurate results even when **X** is near 0. See also ATAN.

Parameter Data Types

The following data types can be used with the ATAN2 instruction.

If **X** is a:	and **Y** is a:	then **Z** is a:
DINT or LREAL	DINT or LREAL	LREAL

Base 10 Logarithm (LOG)

The LOG instruction calculates the logarithm (base 10) of **A** and stores the result in **B**.

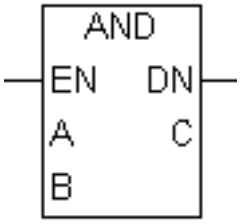
This instruction always passes power.

Parameter Data Types

The following data types can be used with the LOG instruction.

If **A** is a:	then **B** is a:
DINT or LREAL	LREAL

Bitwise AND



The Bitwise AND instruction turns each bit in **C** ON(1) if the corresponding bit in **A** is ON(1) AND the corresponding bit in **B** is ON(1). Otherwise, the bit is turned OFF(0).

Truth Table

A AND	**B** =	**C**
1	1	1
1	0	0
0	1	0
0	0	0

Variations

The following variations apply to the Bitwise AND instruction:

- If none of the parameters is an array, a simple 32-bit AND is performed.
- **A** and **C** may be arrays, with **B** being a non-array DINT. In this case, each element of **A** is ANDed with **B** (and the bits in each element are ANDed in turn). Each result is placed in the corresponding element in **C**. The arrays must be the same size.
- All three parameters may be arrays of the same size. In this case, array **A** is ANDed with array **B**. The results are placed in array **C**.

Parameter Data Types

Valid parameter data types for the AND instruction are as follows:

If **A** is:	and **B** is:	then **C** is:
DINT	DINT	DINT
DINT array-size N	DINT array-size N	DINT array-size N
DINT	DINT constant	DINT
DINT array-size N	DINT constant	DINT array-size N
DINT array-size N	DINT	DINT array-size N

Example

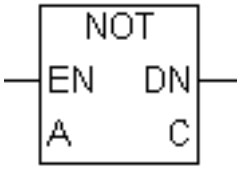
The following example shows the result of two DINTs being ANDed together. 0 = OFF, 1 = ON.

A = 0 1 1 0 ... 0 1 0 0
B = 0 1 0 0 ... 0 0 0 0
C = 0 1 0 0 ... 0 0 0 0

Tips

- DINTs used in a Bitwise AND should be displayed in binary format as other number formats may present misleading information.
- To AND two BOOLs use the following ladder logic:

Bitwise NOT



The NOT instruction turns each bit in **C** ON(1) if the corresponding bit in **A** is OFF(0), and vice versa.

This instruction always passes power.

Truth Table

A NOT =	**C**
1	0
0	1

Variations

There are two variations on the NOT instruction:

- If both variables are DINTs, a simple 32-bit NOT is performed.
- If both variables are arrays of the same size, the entire **A** array is NOTed and placed in **C**.

Parameter Data Types

Valid parameter data types for the NOT instruction are as follows:

If **A** is:	then **C** is:
DINT	DINT
DINT Array	DINT Array

Example

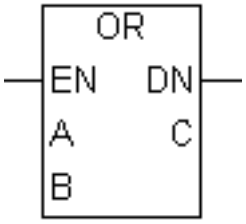
The following example shows the result when a DINT is NOTed. 0=OFF, 1=ON

A = 0 1 1 0 ... 1 1 0 0
C = 1 0 0 1 ... 0 0 1 1

Tips

- DINTs used in a Bitwise NOT should be displayed in binary format as other number formats may present misleading information.
- To NOT a BOOL use the following ladder logic:

Bitwise OR



The Bitwise OR instruction turns each bit in **C** ON(1) if the corresponding bit in **A** is ON(1) OR the corresponding bit in **B** is ON(1). Otherwise, the bit is turned OFF(0).

This instruction always passes power.

Truth Table

A	OR	**B**	=	**C**
1		1		1
1		0		1
0		1		1
0		0		0

Variations

There are three variations for the Bitwise OR instruction:

- If both **A** and **B** are DINTs, a simple 32-bit OR is performed.
- **A** and **C** may be arrays, with **B** being a non-array. In this case, each element of **A** is ORed with **B** (and the bits in each element are ORed in turn). Each result is placed in the corresponding element in **C**. The arrays must be the same size.
- All three parameters may be arrays of the same size. In this case, array **A** is ORed with array **B**. The results are placed in array **C**.

Parameter Data Types

Valid parameter data types for the Bitwise OR instruction are as follows:

If **A** is:	and **B** is:	then **C** is:
DINT	DINT	DINT
DINT array-size N	DINT array-size N	DINT array-size N
DINT	DINT constant	DINT
DINT array-size N	DINT constant	DINT array-size N
DINT array-size N	DINT	DINT array-size N

Example

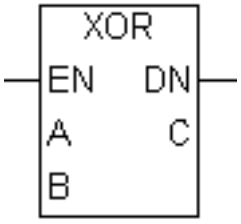
The following example shows the result of two DINTs being ORed together. 1 = ON, 0 = OFF

A = 0 1 1 0 ... 1 1 0 0
B = 1 1 0 0 ... 0 0 0 1
C = 1 1 1 0 ... 1 1 0 1

Tips

- DINTs used in a Bitwise OR should be displayed in binary format as other number formats may present misleading information.
- To OR two BOOLs use the following ladder logic:

Bitwise XOR



The Bitwise XOR instruction turns each bit in **C** ON(1) if the corresponding bit in **A** is ON(1), or the corresponding bit in **B** is ON(1), but not both. Otherwise, the bit in **C** is turned OFF(0).

This instruction always passes power.

Truth Table

A XOR	**B** =	**C**
1	1	0
1	0	1
0	1	1
0	0	0

Variations

There are three variations for the Bitwise XOR instruction:

- If both **A** and **B** are DINTs, a simple 32-bit exclusive OR is performed.
- **A** and **C** may be arrays, with **B** being a non-array. In this case, each element of **A** is exclusive ORed with **B** (and the bits in each element of **A** are exclusive ORed in turn). Each result is placed in the corresponding element in **C**. The arrays must be the same size.
- All three parameters may be arrays of the same size. In this case, array **A** is exclusive ORed with array **B**. The results are placed in array **C**.

Parameter Data Types

Valid parameter data types for the XOR instruction are as follows:

If **A** is:	and **B** is:	then **C** is:
DINT	DINT	DINT
DINT array-size N	DINT array-size N	DINT array-size N
DINT	DINT constant	DINT
DINT array-size N	DINT constant	DINT array-size N
DINT array-size N	DINT	DINT array-size N

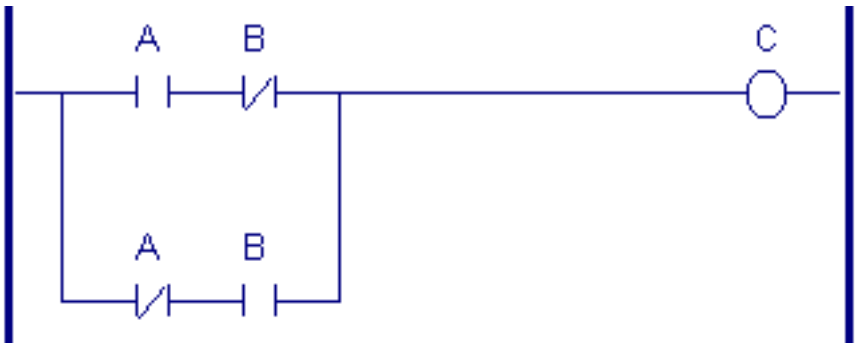
Example

The following example shows the result of two DINTs being exclusive ORed together. 0=OFF, 1=ON

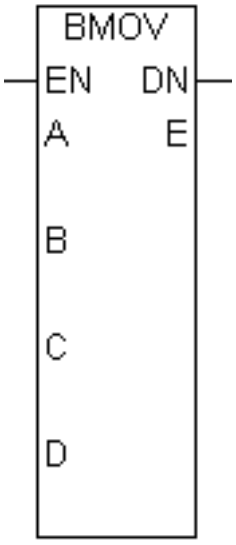
A = 0 1 1 0 ... 1 1 0 0
B = 1 1 0 0 ... 1 0 0 1
C = 1 0 1 0 ... 0 1 0 1

Tips

- DINTs used in a Bitwise XOR should be displayed in binary format as other number formats may present misleading information.
- To exclusive OR two BOOLs use the following ladder logic:



Block Move (BMOV)



The BMOV instruction copies some of the elements of one DINT array into some of the elements of another DINT array. Specifically, **D** elements of array **A**, starting at index **B**, are copied into array **E**, starting at index **C**.

The BMOV instruction always passes power.

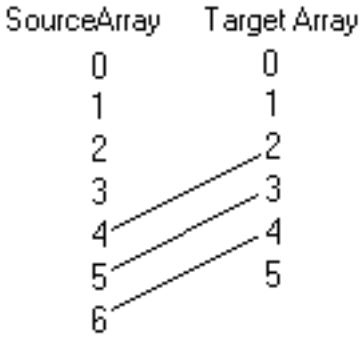
Parameter Data Types

DINT arrays of different sizes can be used.

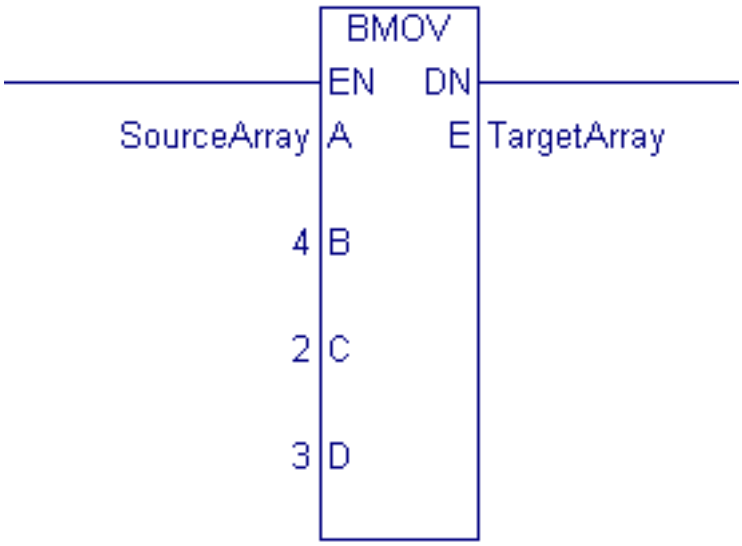
A,E can be:	**B,C,D** can be:
DINT Array	DINT
	DINT Constant

Example

To copy three elements of SourceArray[7] into TargetArray[6] as shown below,



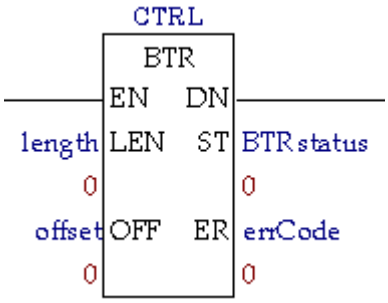
configure the BMOV instruction like this:



Note: If the instruction tries to access an element of an array that does not exist, #Status will indicate a major fault and #FaultCode will indicate an array bounds error.

BTR custom function block

Use the BTR instruction to transfer a block of input data from an intelligent module.



The following table describes the parameters assigned to the BTR instruction

Parameter	Valid types	Min. Size	Req'd	Description
CTRL	Integer Array	2	Yes	Control block.
LEN	Integer		No	Length of block transfer data buffer. See Paragraph 2 in the note below.
OFF	Integer		No	Offset into block transfer data buffer. See Paragraph 3 in the note below.
ST	Integer		Yes	Status of the block transfer request if

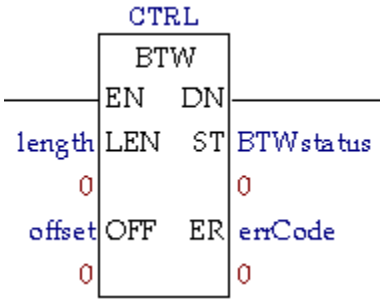
			the instruction is enabled. 0 = Idle 1 = block transfer is queued to driver. 2 = block transfer is queued to card 3 = A response or an error was received. See Paragraph 1 in the note below.
ER	Integer	No	Error code

Notes:

- The enable line (EN) should be held high until the ST variable is set to 2, indicating the block transfer request has been accepted by the KTx scanner card and is in progress. All requests are accepted at the end of the scan unless the KTx queue is full. The KTx can queue up to 64 block transfer requests (only 1 request per ASB adapter).
- If LEN is not defined or set to 0, the entire data buffer is transmitted.
- If OFF is not defined or set to 0, the block transfer starts at the beginning of the transfer buffer. An offset might be required for certain modules, such as the PID module. In most cases, it can be safely set to 0.
- When the block transfer has completed, the ST variable is set to 3 and DN or ER is set indicating if the transfer was successful or an error occurred. When the next transfer is initiated, DN and ER are automatically cleared. If EN is held high, then another request is immediately queued on completion of the last.

BTW custom function block

Use the BTW instruction to transfer a block of output data to an intelligent module.



The following table describes the parameters assigned to the BTW instruction.

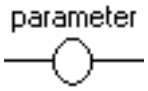
Parameter	Valid types	Min. Size	Req'd	Description
CTRL	Integer Array	2	Yes	Control block
LEN	Integer		No	Length of block transfer data buffer. See Paragraph 2 in the note below.
OFF	Integer		No	Offset into block transfer data buffer. See Paragraph 3 in the note below.
ST	Integer		Yes	Status of the block transfer request if

			the instruction is enabled. 0 = Idle 1 = block transfer is queued to driver. 2 = block transfer is queued to card 3 = A response or an error was received. See Paragraph 1 in the note below.
ER	Integer	No	Error code

Notes:

- The enable line (EN) should be held high until the ST variable is set to 2, indicating the block transfer request has been accepted by the KTx scanner card and is in progress. All requests are accepted at the end of the scan unless the KTx queue is full. The KTx can queue up to 64 block transfer requests (only 1 request per ASB adapter).
- If LEN is not defined or set to 0, the entire data buffer is transmitted.
- If OFF is not defined or set to 0, the block transfer starts at the beginning of the transfer buffer. An offset might be required for certain modules, such as the PID module, but in most cases are safely set to 0.
- When the block transfer has completed, the ST variable is set to 3 and DN or ER is set indicating if the transfer was successful or an error occurred. When the next transfer is initiated, DN and ER are automatically cleared. If EN is held high then another request is immediately queued on completion of the last.

Coil (OUT)



When power is applied to an OUT instruction the parameter turns ON(1). When no power is applied, the parameter is OFF(0).

Because OUT is an output instruction, it should appear at the end of a rung.

Variations

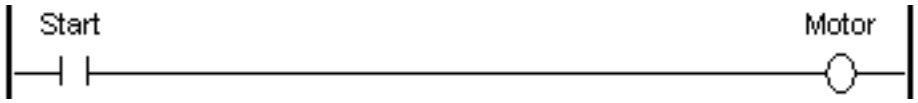
When the assigned parameter is specified as retentive the following alternate symbol is displayed in the logic.

Parameter Data Types

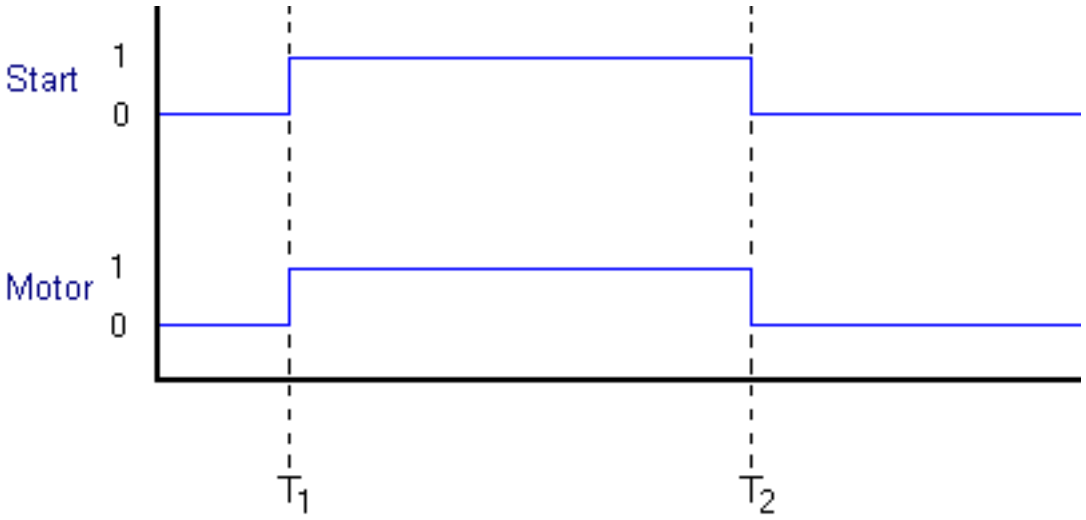
The parameter is a BOOL data type. By default, the BOOL variable automatically created when inserting an OUT instruction will be non-retentive (set to its initial state when the Controller starts following a shutdown or reset).

Example

The following rung diagram illustrates a typical application of the OUT instruction.



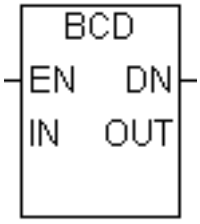
When the BOOL variable 'Start' is ON(1), power passes to the OUT instruction, causing the variable 'Motor' to turn ON(1). The operation is further illustrated in the following timing diagram.



T_1: 'Start' turns ON(1) closing the normally open contact. Power flows to the OUT coil causing 'Motor' to turn ON(1).

T_2: 'Start' turns OFF(0) opening the normally open contact. Power is removed from the OUT coil causing 'Motor' to turn OFF(0).

Convert to BCD (BCD)



The BCD instruction converts a binary number assigned to **IN**, to binary-coded-decimal

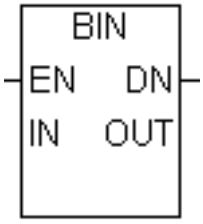
The instruction passes power unless an error occurs.

Parameter Data Types

IN can be:	**OUT** can be:
DINT	DINT
DINT Constant	

Note: The largest value of **IN** that can be converted is 16#5F5E0FF. If **OUT** is too large #FaultCode is updated with the error code and #Overflow is turned ON(1).

Convert to BIN (BIN)



The BIN instruction converts a binary coded decimal number assigned to **IN**, to binary format and places the result in **OUT**.

The instruction will pass power unless an error occurs.

Parameter Data Types

IN can be:	**OUT** can be:
DINT	DINT
DINT Constant	

Note: The largest value of **OUT** that can be generated is 16x5F5E0FF. If **IN** is not a valid BCD number #FaultCode will be updated with the error code and #Overflow will be turned ON(1).

Copy (COPY)

The COPY instruction copies **IN** to **OUT**.

This instruction always passes power.

Parameter Data Types

The parameters **IN** and **OUT** must be the same data types. Arrays and structures can be copied, but they must be of identical type and size.

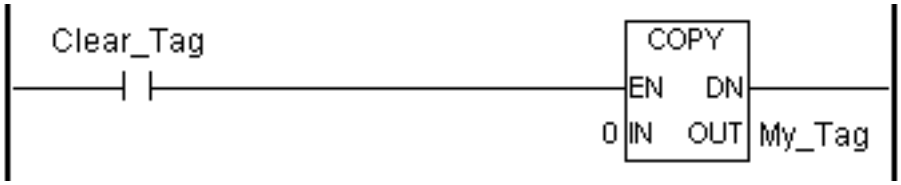
Valid parameter data types for the COPY instruction are as follows:

If **IN** is:	then **OUT** is:
BOOL	BOOL
BOOL array size n	BOOL array size n
DINT	DINT
DINT constant	DINT
DINT array size n	DINT array size n
LREAL	LREAL
LREAL constant	LREAL
LREAL array size n	LREAL array size n
STRUCTURE	STRUCTURE

Tip: One-dimensional arrays can be copied to two-dimensional arrays (and vice versa) if the total number of elements assigned to **IN** and **OUT** are equal.

Example 1

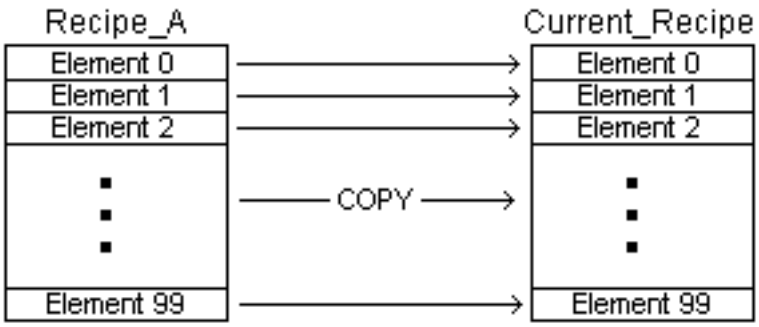
A variable can be cleared by using the COPY instruction to copy a 0 into it. When 'Clear_Tag' is set to ON(1), a 0 is copied into 'My_Tag'.



Tip: In Example 1, My_Tag must be a DINT because the 0 assigned to **IN** is a DINT.

Example 2

A block copy can be performed with the COPY instruction by specifying two arrays of the same type and size. In the following example, two 100 element arrays are used to store recipes. The recipes can be copied with a single COPY instruction. When 'Transfer_Recipe_A' is set to ON(1), the content of array 'Recipe_A' is copied into 'Current_Recipe'.



Tip: In Example 2, the array assigned to **IN** or **OUT** (or both) could be a two-dimensional array, if the total number of elements in each array is equal.

See also the assign (:=) instruction in ST and the MOV instructions in ladder and FBD.

Cosine (COS)

The COS instruction calculates the cosine of **A** and stores the result in **B**.

This instruction always passes power.

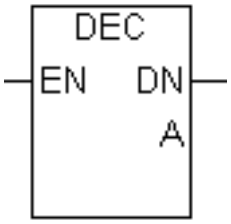
The angle is specified in radians.

Parameter Data Types

The following data types can be used with the COS instruction.

If **A** is a:	then **B** is a:
DINT or LREAL	LREAL

Decrement (DEC)



The DEC instruction subtracts one from **A**, placing the result in **A**.

The instruction always passes power.

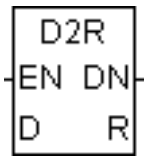
Parameter Data Types

A can be:
DINT

Notes

- **A** cannot be a variable indexed array element. For example, 'MyIntArray[Index]' is not a valid parameter.
- #Overflow is set if **A** decrements from 16x80000000 to 16x7FFFFFFF (from the largest negative value for a DINT to the largest positive value).

Degrees to Radians (D2R)



The D2R instruction converts the angle **D** from degrees to radians, and stores the result in **R**.

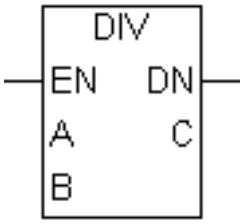
This instruction always passes power.

Parameter Data Types

The following data types can be used with the D2R instruction.

If **D** is a:	then **R** is a:
DINT or LREAL	LREAL

Divide (DIV)



The DIV instruction divides **A** by **B**, placing the quotient in **C**.

This instruction always passes power.

Variations

If both **A** and **B** are of DINT (or DINT constant) type, the instruction performs a DINT division. Otherwise, it performs a possibly slower floating-point (LREAL) division.

Parameter Data Types

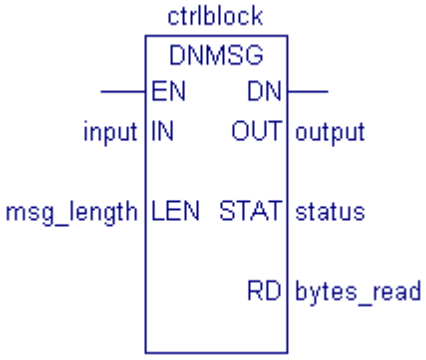
The following data types can used with the DIV instruction:

A, B can be:	**C** can be:
DINT DINT constant LREAL LREAL constant	DINT or LREAL

Notes

- #Overflow is set if the result is too large for the destination parameter, or on division by zero. In either case **C** is undefined.
- If either **A** or **B** are LREALs, both are converted to LREALs prior to the division. The results are placed in **C** and truncated if **C** is a DINT.

DNMSG custom function block



This instruction sends a DNMSG explicit message to a device. The device control blocks determines which device the message is sent to. The following table describes the variables assigned to the DNMSG instruction.

Parameter	**Valid types**	**Min. Size**	**Required**	**Description**
CTRL	DINT Array	8 + LEN/4	Y	Control block.
IN	DINT Array	1 + LEN/4	N	Data to send.
LEN	DINT		Y	Number of bytes to copy from IN.
OUT	DINT Array	(see below)	N	Buffer to receive message response.
ST	DIN		Y	Status of message

AT	T		. (see below)
R D	DIN T	N	Number of bytes copied to OUT.

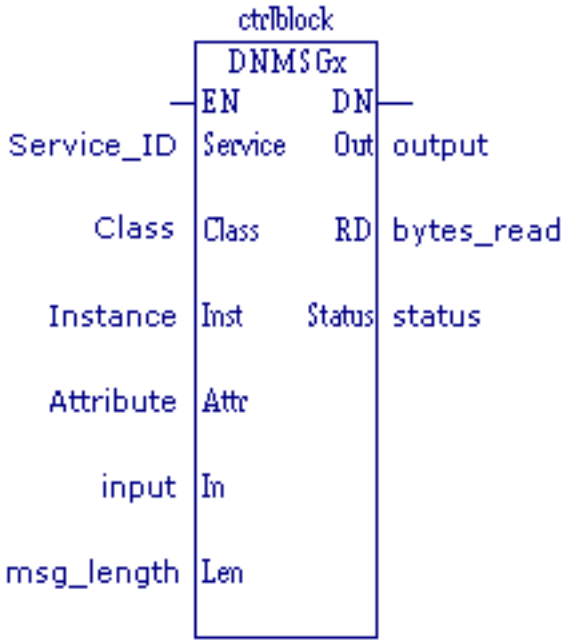
At the top of the instruction is the control block. This DINT array variable should be mapped to the Msg terminal of the device the message is for. The array must be large enough to hold eight elements plus the data copied from IN.

The DINT array variable assigned to IN contains the data to send as an array of packed bytes. The variable assigned to LEN is the number of bytes to copy from IN, and is ignored if no variable is assigned to IN.

Any response to a message will be contained in the DINT array variable assigned to OUT as an array of packed bytes. The size of the variable assigned to OUT must be great enough to contain the response message. The size of the response message depends on the attribute of the object of the device the message is sent to. Refer to the device's data sheet and the DeviceNet specification. The variable assigned to RD is the number of bytes copied to OUT.

The variable assigned to STAT contains the status of this DNMSG instruction and should not be used by any other instruction. The value of the STAT variable is changed depending upon the state of the instruction (see DNMSG Timing) and is used to determine power flow for the instruction. If an error occurs during message transmission, the runtime error code is stored in the STAT variable.

DNMSGx Custom Function Block



Tip: Double-click the DNMSGx instruction to configure it.

This instruction sends an explicit message to a DeviceNet slave. The device control blocks determine which slave the message is sent to. The following table describes the variables assigned to the DNMSGx instruction.

Parameter	Valid Types	Minimum Size	Required/ Optional	Description
CTRL	DINT Array	8 + LEN/4	Required	Control Block
SERVICE	DINT		Required	Service ID
CLASS	DINT		Required	Class ID
INST	DINT		Required	Instance
ATTRIB	DINT		Optional	Attribute ID
IN	DINT Array	1 + LEN/4	Optional	Data to send
LEN	DINT		Required	Number of bytes to copy from IN

OUT	DINT Array	Required	Buffer to receive message response
STAT	DINT	Required	Status of message
RD	DINT	Required	Number of bytes copied to OUT

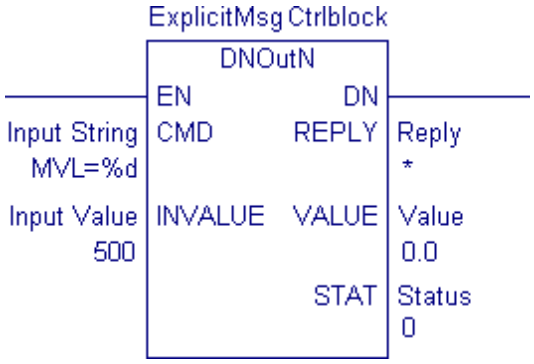
At the top of the instruction is the control block. This DINT array variable should be mapped to the Msg terminal of the slave the message is for. The array must be large enough to hold eight elements plus the data copied from IN.

The DINT array variable assigned to IN contains the data to send as an array of packed bytes. The variable assigned to LEN is the number of bytes to copy from IN.

Any response to a message will be contained in the DINT array variable assigned to OUT as an array of packed bytes. The size of the variable assigned to OUT must be great enough to contain the response message. The size of the response message depends on the attribute of the object of the slave the message is sent to. Refer to the slave's data sheet and the DeviceNet specification. The variable assigned to RD is the number of bytes copied to OUT.

The variable assigned to STAT contains the status of this DNMSGx instruction and should not be used by any other instruction. The value of the STAT variable changes depending on the state of the instruction (see DNMSGx Timing) and is used to determine power flow for the instruction. If an error occurs during message transmission, the runtime error code is stored in the STAT variable.

DNOutN custom function block



The DNOutN instruction sends an explicit message to a Whedco or GE motion controller. The ExplicitMsg Ctrlblock determines which motion controller the message is sent to. The following table describes the variables assigned to the DNOutN instruction.

Parameter	Valid types	Min. Size	Required	Description
CTRL	DINT Array	20	Y	ExplicitMsg Ctrlblock.
CMD	STRING	20	N	Input string command to send.
INVALUE	BOOL, DINT, LREAL		N	Numerical value of the data to send.
REPLY	STRING	20	N	Buffer to receive message response.
VALUE	BOOL, DINT, LREAL		N	Numerical response from a motion controller

ST AT	DIN T	Y	. Status message of the DNOutN custom function block.

At the top of the instruction is the ExplicitMsg Ctrlblock. This DINT array variable should be mapped to the DNOutN Msg terminal of a Whedco or GE motion controller the message is for. The array must be large enough to hold a STRING variable, plus ten elements, plus the data copied from INVALUE.

The STRING variable assigned to CMD contains the input string information to send to a motion controller. As an option, you can pass DINTs %d or LREALs %f indirectly to CMD through INVALUE (e.g. MAC = %d). The %d will be replaced with a BOOL or DINT value from INVALUE. The %f will be replaced with an LREAL from INVALUE.

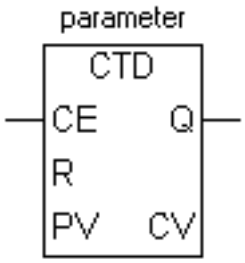
Note: %d can contain either a BOOL or DINT value.

The variable assigned to INVALUE is the numerical value to send to the motion controller, and is ignored if no variable is assigned to CMD.

Any response to a message will be contained in the STRING variable assigned to REPLY. The size of the variable assigned to REPLY must be great enough to contain the response message. The size of the response message depends on the attribute of the object of the device the message is sent to. Refer to the device's data sheet and the DeviceNet specification.

The variable assigned to STAT contains the status of this DNOutN instruction and should not be used by any other instruction. The value of the STAT variable is changed depending upon the state of the instruction and is used to determine power flow for the instruction. If an error occurs during message transmission, the runtime error code is stored in the STAT variable.

Down Counter (CTD)



For every scan the CTD instruction receives power, it will decrement by one, starting from a preset value. The instruction passes power when it has decremented to zero or less.

Parameter Data Types

The parameter assigned to the CTD instruction is a COUNTER structure variable described in the following table.

Element	Description	Data type	Example
PV	Preset value	DINT	MyCounter.PV
CV	Current value	DINT	MyCounter.CV
R	Reset bit	BOOL	MyCounter.R
UP	Counting UP bit	BOOL	MyCounter.UP
QU (n/a)	Done UP bit	BOOL	MyCounter.QU
QD	Done DOWN bit	BOOL	MyCounter.QU
Q	Output enable bit	BOOL	MyCounter.Q

The preset value 'MyCounter.PV' must be set manually or by another instruction executed before the CTD instruction.

Detailed Operation

The current value 'MyCounter.CV' is decremented by one when the CTD instruction receives power and the reset enable bit 'MyCounter.R' is OFF(0).

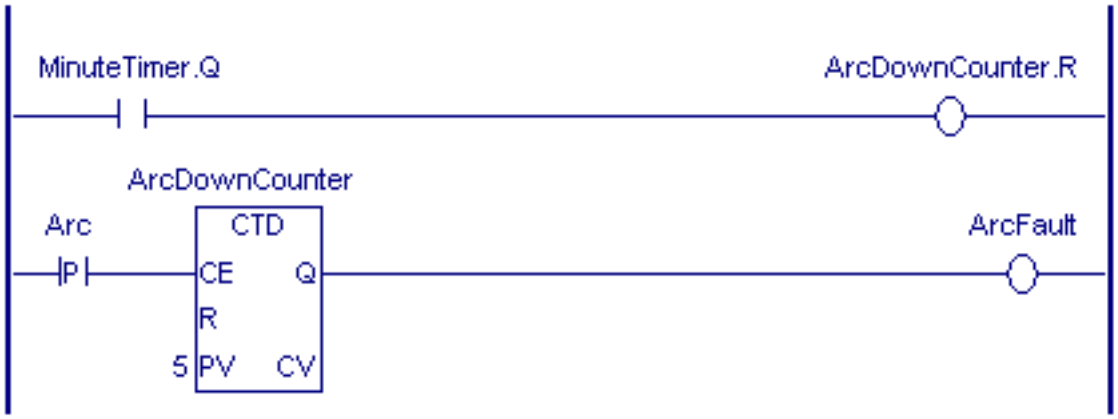
The output enable bit 'MyCounter.Q' is turned ON(1) and the instruction

passes power when the current value 'MyCounter.CV' is less than or equal to zero after decrementing.

When the reset enable bit 'MyCounter.R' is ON(1), the current value 'MyCounter.CV' is reset to zero.

Example

The following example shows how a CTD instruction is used to flag a fault after 5 arcs have been counted in a 1 minute period. 'MinuteTimer' will reset the counter every minute.

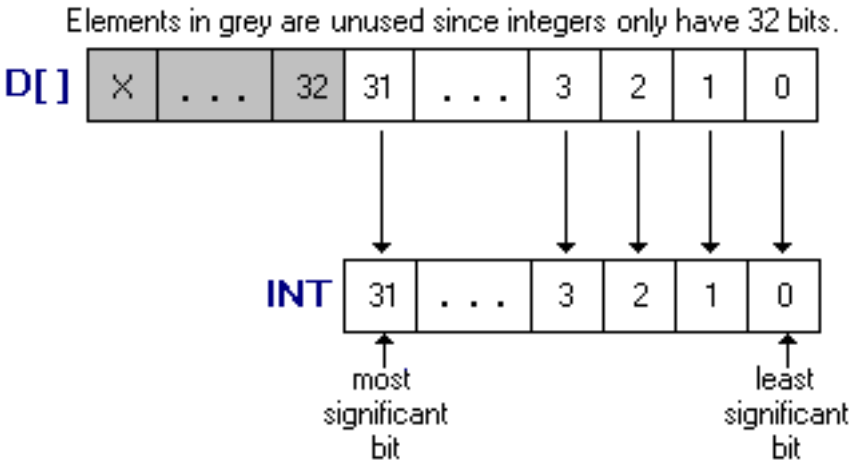


Note: COUNTERs increment or decrement with every logic scan when powered. To count events, as in the example above, you will need an instruction placed on the rung before the COUNTER enable (CE) input.

DTOI Instruction

The BOOL-TO-DINT (DTOI) instruction encodes a BOOL array into a DINT value. Each BOOL array element is encoded to a consecutive DINT bit (starting with the least significant bit).

This instruction always passes power.



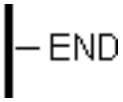
Note: If the BOOL array has fewer than 32 elements, the same number of bits are used in the DINT value and zeros are placed in the remaining bits.

Parameter	Type	Description
D[]	BOOL	Up to 32 elements can be mapped. If D[] has more than 32 elements, only the first 32 are used and the rest are ignored.

INT	DINT	Up to 32 INT bits can be used. Each INT bit is set or cleared by a corresponding D[] element.

Note: To decode DINT bits to a BOOL array, use the ITOD instruction.

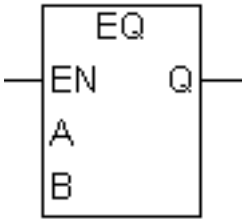
END Label



The END label indicates the end of the main program area and is included by default in every ladder program. Rungs between the START and END labels are executed during every scan. No instructions can be inserted on the same rung as the END label.

Note: The END label can be specified as the destination of a JMP instruction, if the JMP is positioned between the START and END labels.

Equal (EQ)



The EQ instruction passes power if **A** is exactly equal to **B**.

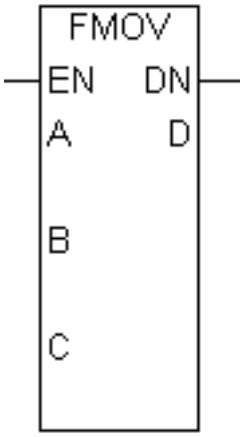
Parameter Data Types

The following parameter data types can be used with the EQ instruction.

A, B can be:
- DINT
- DINT constant
- LREAL
- LREAL constant

Warning: Comparing LREAL values may produce unexpected results. For example, a calculation might result in 1.99999999999, which is not equal to 2.00000000000.

Fill Move (FMOV)



The FMOV instruction fills **C** elements of DINT array **D** with value **A**, starting at index **B**.

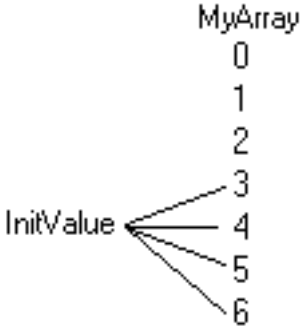
The FMOV instruction always passes power.

Parameter Data Types

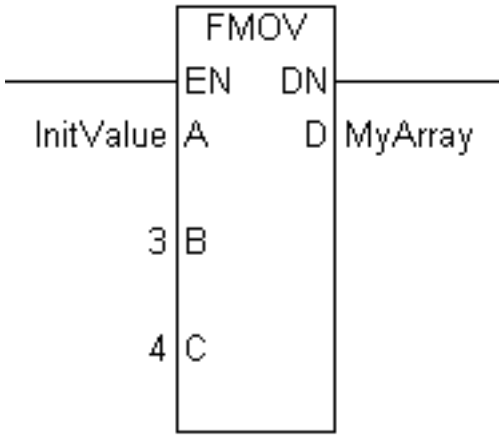
A, B, C can be:	**D** can be:
DINT	DINT array
DINT constant	

Example

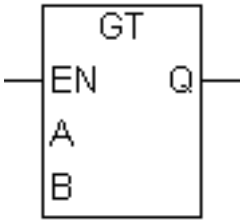
To copy an initial value, 'InitValue', into some of the elements of the seven element array 'MyArray', as shown below,



configure the FMOV instruction like this:


FMOV
EN
DN
InitValue
A
D
MyArray
3
B
4
C

Greater Than (GT)



The GT instruction passes power if **A** is greater than **B**.

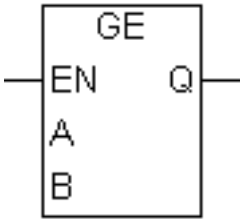
Parameter Data Values

The following parameter data types can be used with the GT instruction.

A, B can be:
DINT
DINT constant
LREAL
LREAL constant

Warning: Comparing LREAL values may produce unexpected results. For example, a calculation might result in 2.000000000001, which is greater than 2.

Greater Than or Equal (GE)



The GE instruction passes power if **A** is greater than, or equal to **B**.

Parameter Data Values

The following parameter data types can be used with the GE instruction.

A, **B** can be:
DINT
DINT constant
LREAL
LREAL constant

Warning: Comparing LREAL values may produce unexpected results. For example, a calculation might result in 1.99999999999, which is not greater than or equal to 2.

Hyperbolic Cosine (COSH)

The COSH instruction calculates the hyperbolic cosine of **A** and stores the result in **B**.

This instruction always passes power.

Parameter Data Types

The following data types can be used with the COSH instruction.

If **A** is a:	then **B** is a:
DINT or LREAL	LREAL

Hyperbolic Sine (SINH)

The SINH instruction calculates the hyperbolic sine of **A** and stores the result in **B**.

This instruction always passes power.

Parameter Data Types

The following data types can be used with the SINH instruction.

If **A** is a:	then **B** is a:
DINT or LREAL	LREAL

Hyperbolic Tangent (TANH)

The TANH instruction always calculates the hyperbolic tangent of **A** and stores the result in **B**.

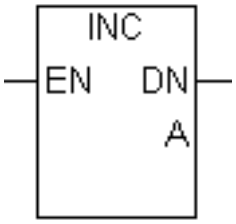
This instruction always passes power.

Parameter Data Types

The following data types can be used with the TANH instruction.

If **A** is a:	then **B** is a:
DINT or LREAL	LREAL

Increment (INC)



The INC instruction adds one to **A**, placing the result in **A**.

The INC instruction always passes power.

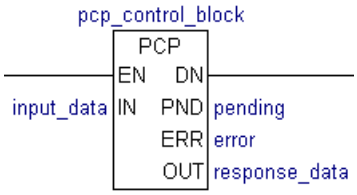
Parameter Data Types

A can be:
DINT

Notes

- **A** cannot be a variable indexed array element. For example, 'MyIntArray[Index]' is not a valid parameter.
- #Overflow is set if **A** increments from 16x7FFFFFFF to 16x80000000.

PCP instruction



The PCP instruction is used to send parameter data to a PCP compatible device. The PCP control block variable must be an integer array of at least five elements and be mapped to the PCP Msg terminal for the device or the instruction is ineffective.

Except for the PCP control block variable, all other variables on the instruction are optional. Below are descriptions of each parameter. The values in brackets () are the parameter's data type.

IN parameter (integer array)

This contains the data that is sent to a device.

While IN is an array of integers, it is treated as an array of packed BYTES (8 bit values) with 4 bytes per integer.

For example, to send the data 16#18, 16#20, 16#EF, 16#FE, 16#13, and 16#08 (where 16#18 was the first byte and 16#08 is the last byte), set element 1 of IN to 16#FEEF2018 and element 2 to 16#00000813. The length defined in the PCP Message setup dialog box determines how many bytes are actually sent.

PND parameter (discrete)

This parameter is turned on if a message is being sent.

ERR parameter (discrete)

This parameter is turned on if an error occurred.

OUT parameter (integer array)

This parameter displays the results of the message.

OUT contains the response received from the device. The response depends on the message sent. Refer to the module's data sheet for more information.

While OUT is defined as an array of integers, it is treated as an array of packed WORDS (16 bit values).

Depending on whether an error occurred, it contains the following information:

If no error occurred:

High Word

- Data Length

- Data word (2)
- Data word (n)

Low Word

- Result (+)
- Data word (1)
- Data word (3)

If an error occurred:
High Word

- Extra Info

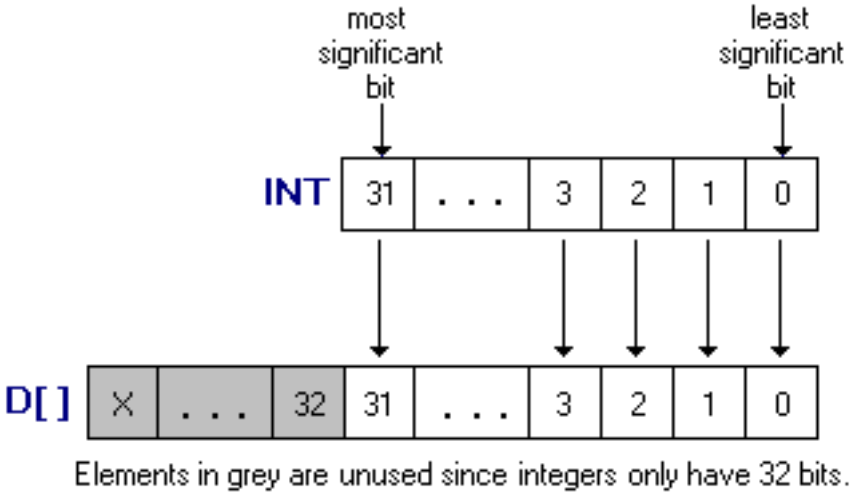
Low Word

- Error Class/Code

ITOD Instruction

The DINT-TO-BOOL (ITOD) instruction decodes the bits of a DINT value into a BOOL array. Each DINT bit is decoded to a consecutive BOOL array element (starting from the least significant bit).

This instruction always passes power.



Note: If the BOOL array has fewer than 32 elements, the same number of DINT bits are decoded and the remaining DINT bits are unchanged.

Parameter	Type	Description

INT	DINT	Up to 32 DINT bits can be used. Each DINT bit decodes to one D[] element. You cannot use a constant DINT or expression.
D[]	BOOL	Each D[] element value is taken from the corresponding DINT bit. Only the elements between 0 and 31 are used.

Note: To encode a BOOL array to a DINT value, use the DTOI Instruction.

Jump (JMP)

———>>parameter

The JMP instruction transfers control to the rung specified by the parameter when this instruction receives power. Unlike the Jump Subroutine (JSR) instruction, control does not automatically return to the rung following the JMP rung.

This must be the last instruction on a rung.

Notes

- A label must be inserted in the logic before a JMP instruction can use it as a destination.
- A jump cannot be made over a START, SUB START , SUB END, ACT START or ACT END label.
- The START label can be the destination of a jump only if the JMP instruction is positioned before the START rung.

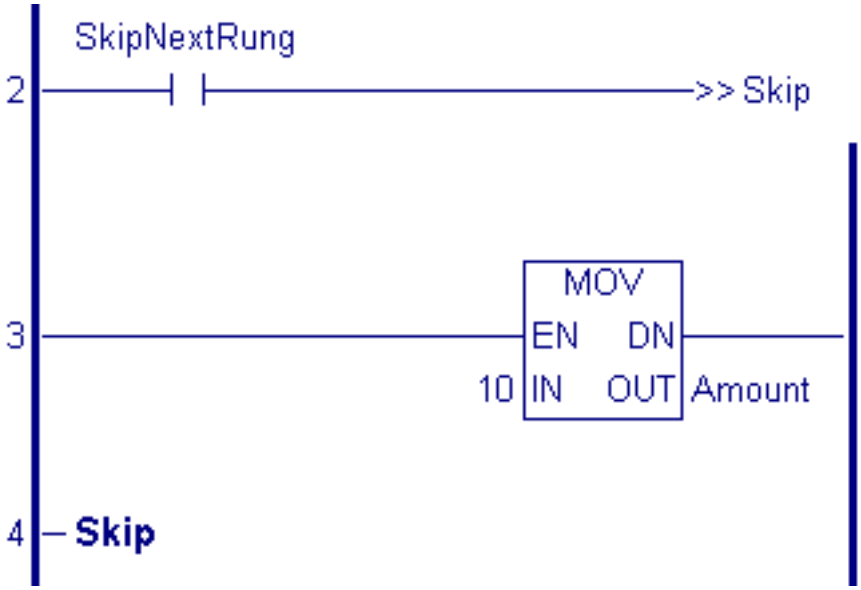
Parameter Data Types

The parameter is the name of a user-defined label, the START or END label.

Example

The following diagram illustrates the use of the JMP instruction.

If 'SkipNextRung' is ON(1), control will transfer to rung 4 (to the label 'Skip'). If 'SkipNextRung' is OFF(0) rung 3 will be executed and 10 will be moved into 'Amount'.



Warning: Use caution when jumping upward as this might create an infinite loop. Control must periodically reach the END rung to reset the watchdog timer.

Jump Subroutine (JSR)

—>> parameter <<

The JSR instruction transfers control to the SUB START label specified by the parameter, when the instruction receives power. After the subroutine executes (the SUB END label is reached), control returns to the rung following this instruction.

This must be the last instruction on a rung.

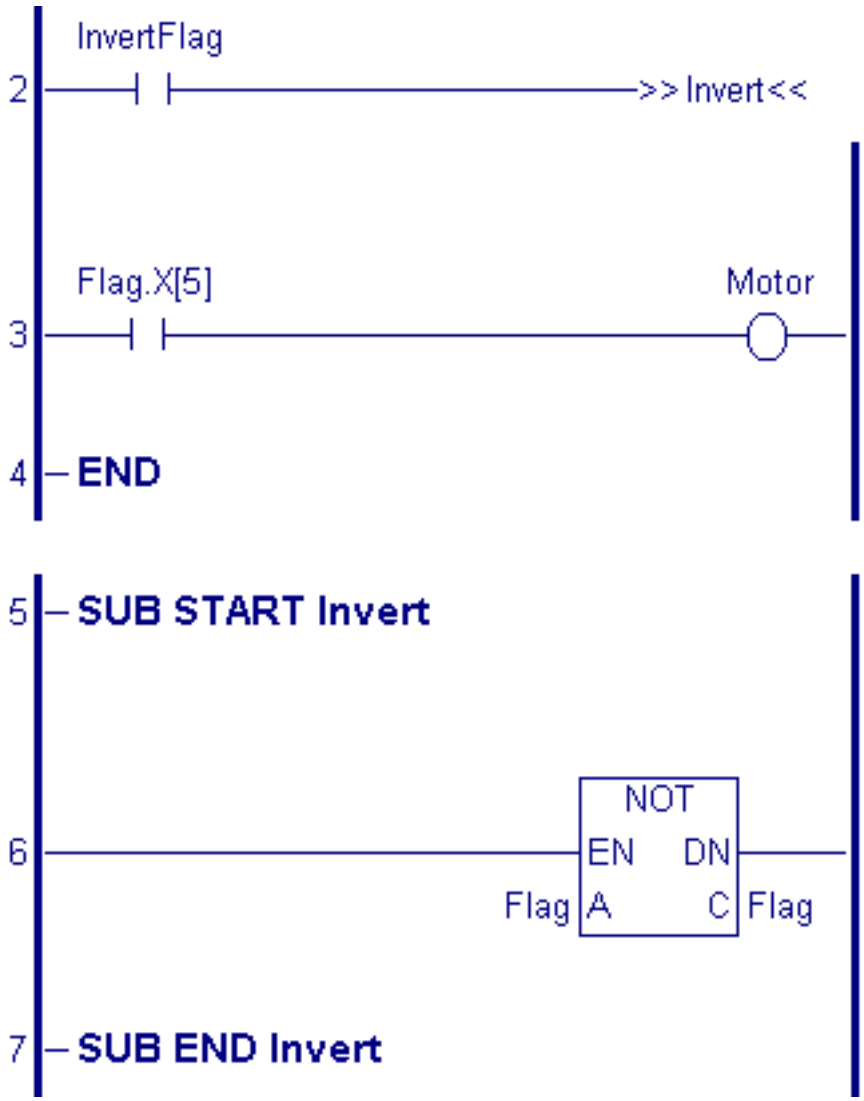
Parameter Data Type

The parameter is the name of a user-defined subroutine.

Example

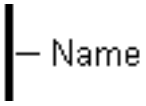
The following diagram illustrates the use of a JSR instruction.

If 'InvertFlag' is ON(1), control is transferred to rung 5 (to the SUB START label 'Invert'). Execution continues on rung 6 where each of the bits in 'Flag' are inverted by the NOT instruction. Control then returns to rung 3. If the sixth bit of 'Flag' is ON(1), then 'Motor' turns ON(1).



Note: A subroutine can be exited prematurely (that is, before execution reaches the SUB END label) by inserting a RETURN instruction.

Label

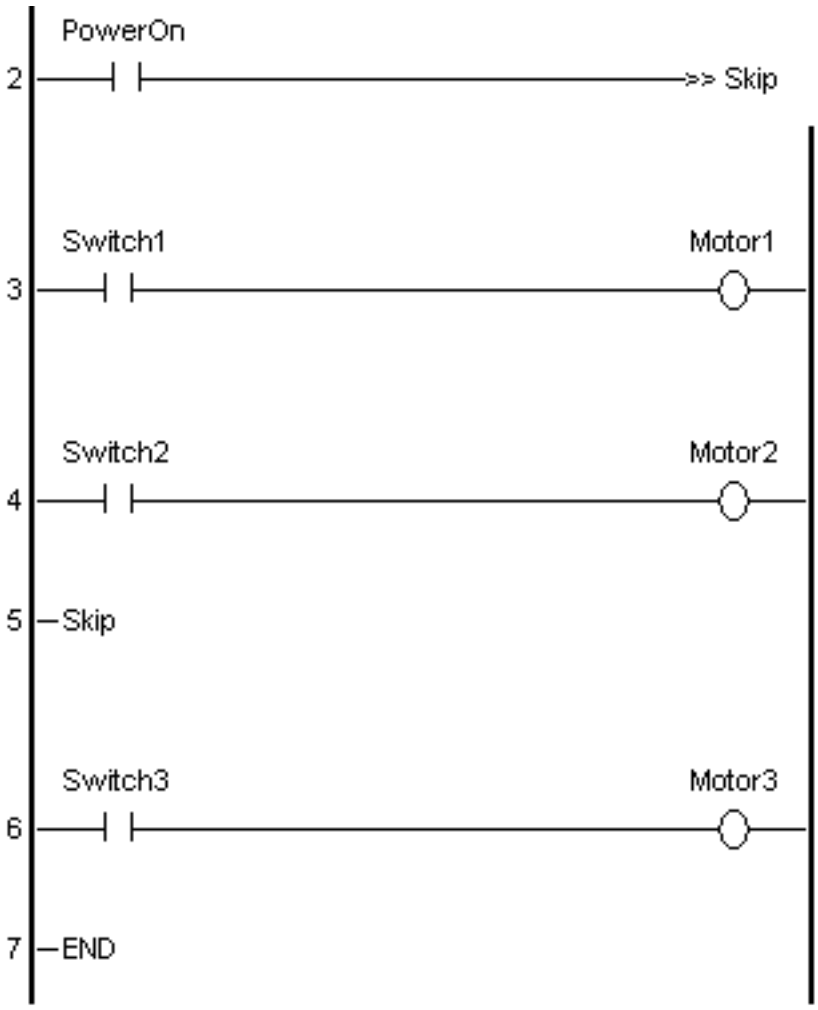


A label marks a position within the logic that can be the destination of a Jump. When control transfers to a label, execution continues on the rung following it.

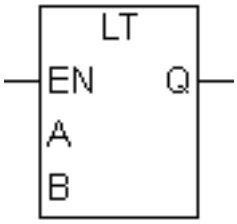
A name is required.

Example

If 'PowerOn' is ON(1), rungs 3 and 4 are not executed as control jumps over them at the jump instruction 'Skip'. Control is transferred to rung 5 (to the label 'Skip') and rung 6 is executed. If 'PowerOn' is OFF(0), rungs 3 and 4 are executed.


PowerOn
2
>> Skip
Switch1
Motor1
3
Switch2
Motor2
4
5
Skip
Switch3
Motor3
6
7
END

Less Than (LT)



The LT instruction passes power if **A** is less than **B**.

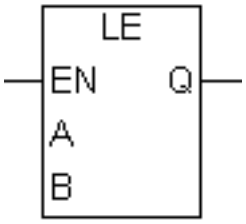
Parameter Data Types

The following parameter data types can be used with the LT instruction.

A, B can be:
- DINT
- DINT constant
- LREAL
- LREAL constant

Warning: Comparing LREAL values may produce unexpected results. For example, a calculation might result in 1.99999999999, which is less than 2.

Less Than or Equal (LE)



The LE instruction passes power if **A** is less than or equal to **B**.

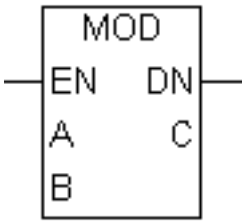
Parameter Data Types

The following parameter data types can be used with the LE instruction.

A, B can be:
- DINT
- DINT constant
- LREAL
- LREAL constant

Warning: Comparing LREAL values may produce unexpected results. For example, a calculation might result in 2.000000000001, which is not less than or equal to 2.

Modulus (MOD)



The MOD instruction divides **A** by **B**, placing the remainder in **C**. It performs a DINT mode of operation only.

This instruction always passes power.

Parameter Data Types

Valid parameter data types for the MOD instruction are as follows:

A, **B** can be:	**C** can be:
DINT	DINT
DINT constant	

Example

In the following example a DINT (27) is divided by 5 and the result (2) is placed in **C**.

A = 27
B = 5

$$5\overline{)27}\quad 5,\ 25,\ 2$$

C = 2

Warning: #Overflow is turned ON(1) upon division by zero and the result **C** is undefined.

Move (MOV)

The MOV instruction copies **IN** to **OUT**.

This instruction always passes power.

Parameter Data Types

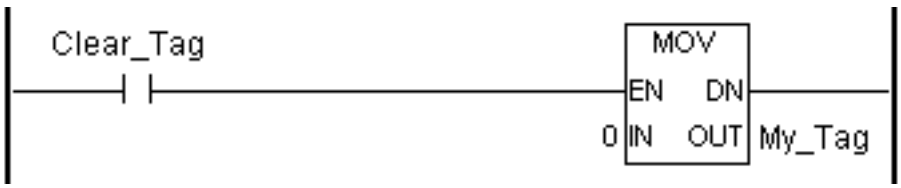
If **IN** and **OUT** parameters are of different types, the result is converted to the type of **OUT**. Arrays can be moved, but they must be of identical type and size.

Valid parameter data types for the MOV instruction are as follows:

If **IN** is:	then **OUT** is:
BOOL array-size N	BOOL array-size N
DINT	DINT or LREAL
DINT array-size N	DINT array-size N or Text
DINT constant	DINT or LREAL
LREAL	DINT or LREAL
LREAL array-size N	LREAL array-size N
LREAL constant	DINT or LREAL
Text	Text or DINT array

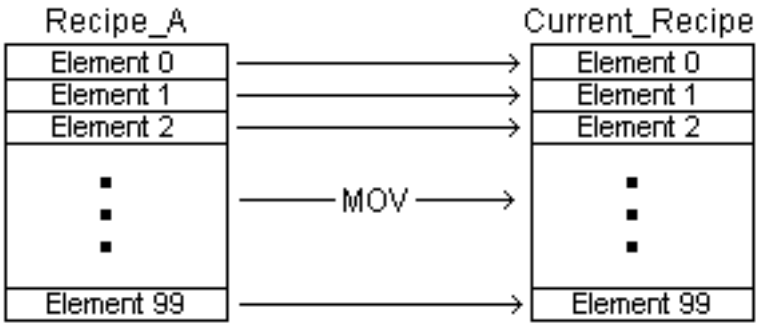
Example 1

A variable can easily be cleared by using the MOV instruction to move a 0 into it. When 'Clear_Tag' turns ON(1), a 0 is moved into 'My_Tag'.



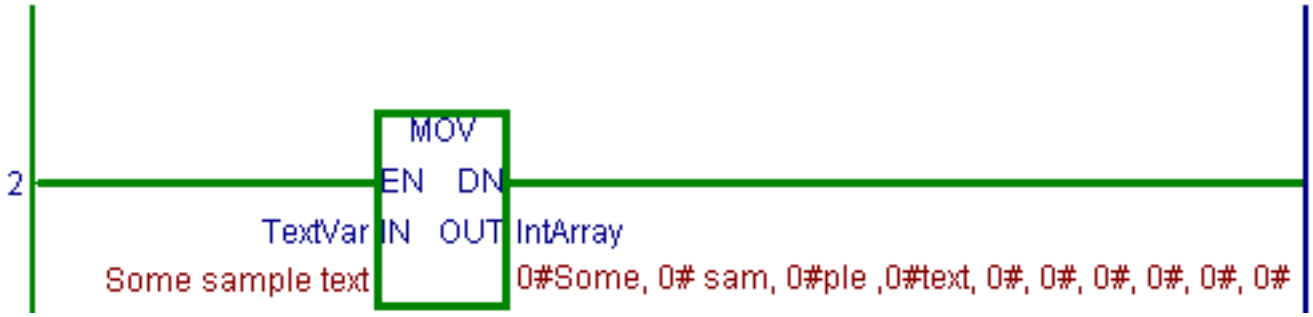
Example 2

A block move can be performed with the MOV instruction simply by specifying two arrays of the same type and size. In the following example, two 100 element arrays are used to store recipes. The recipes can easily be transferred with a single MOV instruction. When 'Transfer_Recipe_A' turns ON(1), the content of array 'Recipe_A' is copied into 'Current_Recipe'.



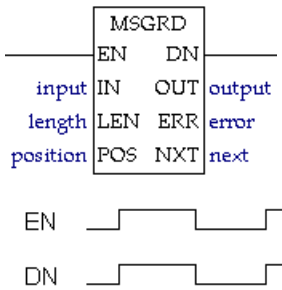
Example 3

You can use a MOV instruction to initialize a DINT array with a text string. Each element of the array can hold up to four ASCII characters. This is useful when working with some Control I/O drivers that require text to be passed in a DINT array. The size of the array does not have to match the number of characters you are moving into it; unused elements of the array are simple set to zero. It is the user's responsibility to ensure the array can hold all the text moved into it. The following example shows how the content of a text variable is copied to a DINT array.



Warning: #Overflow is turned ON(1) if the operation involves a conversion from LREAL to another data type and the value is too large to move. In this instance the result is undefined.

MSGRD overview

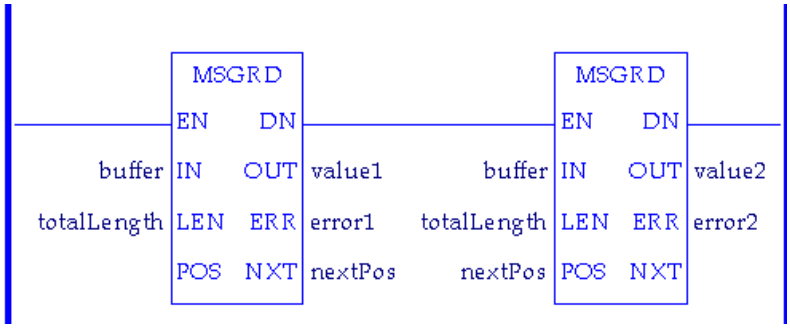


This instruction parses a message buffer. Typically, the buffer output from a MSGRX instruction is passed to this instruction to extract a value. The two instructions are separate because a buffer may need to be parsed more than once or in more than one way. The following table summarizes the valid types for each parameter:

Parameter	Valid types	Required	Description
IN	Integer array or Text variable	Y	Input buffer to parse. IN contains the byte packed message to parse. This buffer typically comes from a MSGRX instruction, but could also come from a MSGWR instruction.
LEN	Integer	Y	Length input buffer. LEN holds the number of bytes in IN and typically comes from a MSGRX instruction. POS is an optional character offset to start parsing from.
POS	Integer	N	Offset to start parsing from.
OUT	Integer, Real, or Text Variable	Y	Output value. OUT stores the value parsed from IN. If any errors are found while parsing IN, the format field number that caused the error is

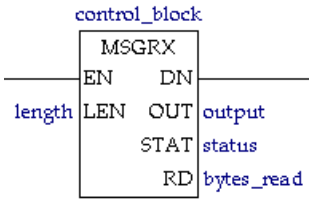
			stored in ERR.
ERR	Integer	N	Error code or 0. A nonzero value specifies the format field where the error occurred.
NXT	Integer	N	Offset where parsing stopped. NXT is an optional character offset that parsing stopped at.

To extract more than one value from a buffer, use more than one MSGRD instruction. When specifying the format string in the MSGRD configuration dialog box, make sure the string ends with a value specifier. The instruction doesn't know to check beyond that value and saves the position of the next character in NXT. Use the value of NXT as POS on the next MSGRD instruction to extract a second value. Continue this method until all values have been extracted. If a variable number of values needs to be extracted, continue passing NXT to POS until ERR is not zero.



Note: If any online programming changes are made to the instruction while it is powered, the power must be brought low before the changes take effect.

MSGRX overview



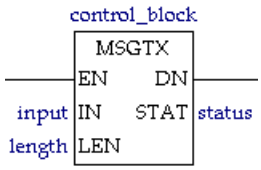
This instruction reads the specified number of bytes from the appropriate resource as identified by the control block. The following shows the valid types for each parameter:

Parameter	Valid types	Required	Description
CTRL	Integer Array	Y	Control block. This variable should be mapped to the Ctrl terminal of a communication resource in the ASCII communications I/O driver.
LEN	Integer	Y	Size of data buffer (is the desired number of characters to read). No more than LEN characters are read, but fewer than LEN characters may be read depending on how the instruction is configured in the MSGRX configuration dialog box.
OUT	Integer Array or Text Variable	Y	Buffer to receive into. OUT receives the data read from the communication resource. The easiest way to parse the output buffer is with the MSGRD instruction.
STAT	Integer	Y	STAT holds the

			status of the instruction. The value of STAT is used to determine power flow for the instruction, making it important not to use the same status variable more than once.
RD	Integer	Y	RD holds the number of characters actually read and copied to OUT.

Note: If any online programming changes are made to the instruction while it is powered, the power must be brought low before the changes take effect.

MSGTX overview



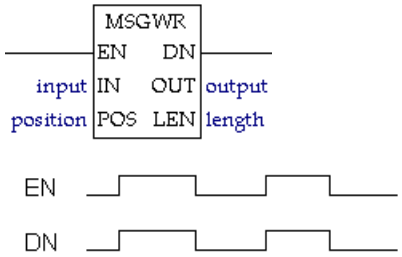
This instruction transmits the specified buffer through the appropriate resource as identified by the control block. The following table shows the valid types for each parameter.

Parameter	Valid types	Required	Description
CTRL	Integer array	Y	Control block. This variable should be mapped to the Ctrl terminal of a communication resource in the ASCII Communications I/O driver. IN contains the buffer to send. The easiest way to build a buffer is using the MSGWR instruction.
IN	Integer array or Text Variable	Y	Data to transmit.
LEN	Integer	Y	Length of data to transmit, (the length of the input buffer in bytes.)
STAT	Integer	Y	Status of instruction. STAT holds the status of the instruction and The value of STAT is used to determine power flow for the instruction, making it important not to use the same status variable more than once. If an error occurs during

transmission, the runtime error code is stored in STAT. To determine the number of bytes sent to the port, check the value of the third element of the control block (Word 2) once a MSGTX has completed.

Note: If any online programming changes are made to the instruction while it is powered, the power must be brought low before the changes take effect.

MSGWR overview



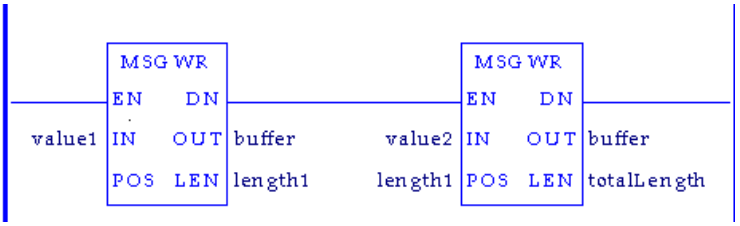
This instruction formats a message buffer suitable for transmitting to a resource. Normally, a MSGWR instruction is followed by a MSGTX instruction. The two instructions are separate because the same buffer may need to be sent to more than one resource or only need to be formatted once. The following table summarizes the valid parameter types:

Parameter	Valid types	Required	Description
IN	Integer, real, or text variable	N	Input data to format.
POS	Integer	N	Offset in output buffer to copy the result to.
OUT	Integer array or text variable	Y	Output buffer.
LEN	Integer	Y	Length of output buffer.

In the following paragraphs, the parameter names refer to the variables mapped to the parameter.

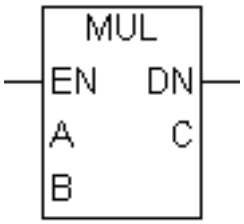
When enabled during Runtime, the instruction uses the value of IN and applies it to a format string specified in the MSGWR configuration dialog box. If a variable is not mapped to IN and the format string includes a specifier, the value of IN is taken as an integer value of zero. POS is an optional integer that specifies an offset into the output buffer. If no variable is mapped to POS, the offset is assumed to be zero (start of the output buffer). The resulting string is byte packed to OUT and the length of the string is stored in LEN.

Complex messages can be built up by cascading MSGWR instructions, similar to concatenating strings. When the string is copied to OUT, it is offset by the number of characters specified by POS. For example, to build a string containing an integer and a real, use one MSGWR to build the string with the integer and another to build the string with the real. Use the value of LEN from the first MSGWR as POS on the second MSGWR.



Note: If any online programming changes are made to the instruction while it is powered, the power must be brought low before the changes take effect.

Multiply (MUL)



The MUL instruction multiplies **A** by **B**, placing the product in **C**.

This instruction always passes power.

Variations

If both **A** and **B** are of DINT (or DINT constant) type, the instruction performs a DINT multiplication. Otherwise, it performs a slower floating-point (LREAL) multiplication.

Parameter Data Types

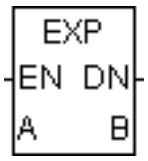
The following data types can be used with the MUL instruction.

A, B can be:	**C** can be:
DINT DINT constant LREAL LREAL constant	DINT or LREAL

Notes

- #Overflow is turned ON(1) if the result is too large for **C**, the destination parameter. In this case the result is undefined.
- If either **A** or **B** are LREALs, both are converted to LREALs prior to the multiplication. The result is placed in **C** and truncated if **C** is a DINT.

Natural Exponent (EXP)



The EXP instruction calculates the value of *e* to the power of **A** and stores the result in **B**.

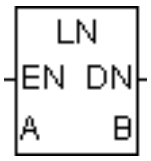
This instruction always passes power.

Parameter Data Types

The following data types can be used with the EXP instruction.

If **A** is a:	then **B** is a:
DINT or LREAL	LREAL

Natural Logarithm (LN)



The LN instruction calculates the natural logarithm (base *e*) of **A** and stores the result in **B**.

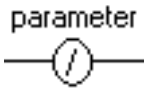
This instruction always passes power.

Parameter Data Types

The following data types can be used with the LN instruction.

If **A** is a:	then **B** is a:
DINT or LREAL	LREAL

Negated Coil (NEG)



The NEG instruction turns the parameter OFF(0) when the coil receives power and ON(1) when it doesn't.

This is an output instruction, so it must be the last one on a rung.

Variations

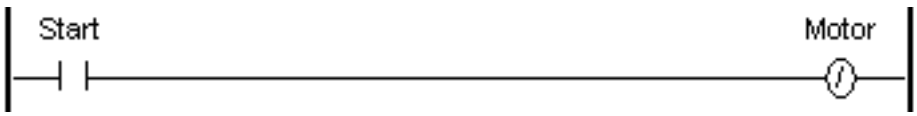
When the assigned parameter is specified as retentive the following alternate symbol is displayed in the logic.

Parameter Data Types

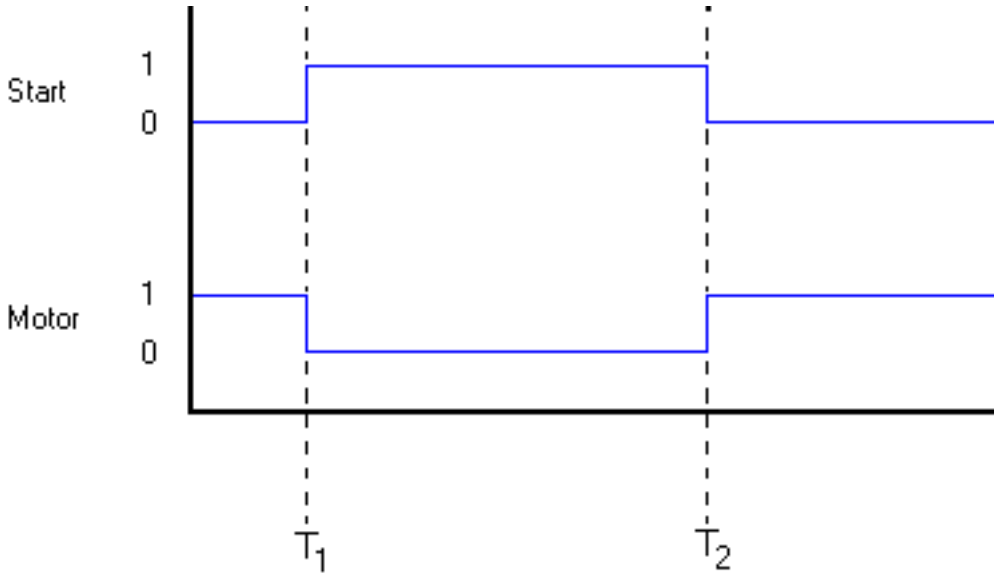
The parameter is a BOOL data type. By default, the BOOL variable automatically created when inserting an NEG instruction will be non-retentive (set to its initial state when the Controller starts following a shutdown or reset).

Example

The following rung diagram illustrates a typical application of a NEG instruction. When 'Start' turns ON(1), 'Motor' turns OFF(0) and vice versa.



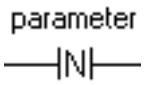
The following timing diagram further illustrates the operation. 0=OFF, 1=ON



T_1: 'Start' turns ON(1) closing the normally open contact. Power flows to the NEG coil turning 'Motor' OFF(0).

T_2: 'Start' turns OFF(0) opening the normally open contact. Power is removed from the NEG coil turning 'Motor' ON(1).

Negative Transition Contact (NT)



The NT instruction allows power to pass if the parameter was ON(1) during the previous scan but is OFF(0) now. In other words, the parameter has gone through a negative transition during the last scan and is now OFF(0).

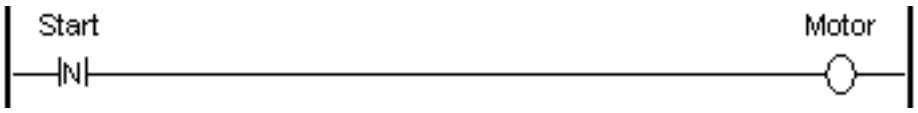
For the first scan, the previous state of the parameter is assumed to be OFF(0). Therefore, this instruction never passes power on the first scan.

Parameter Data Types

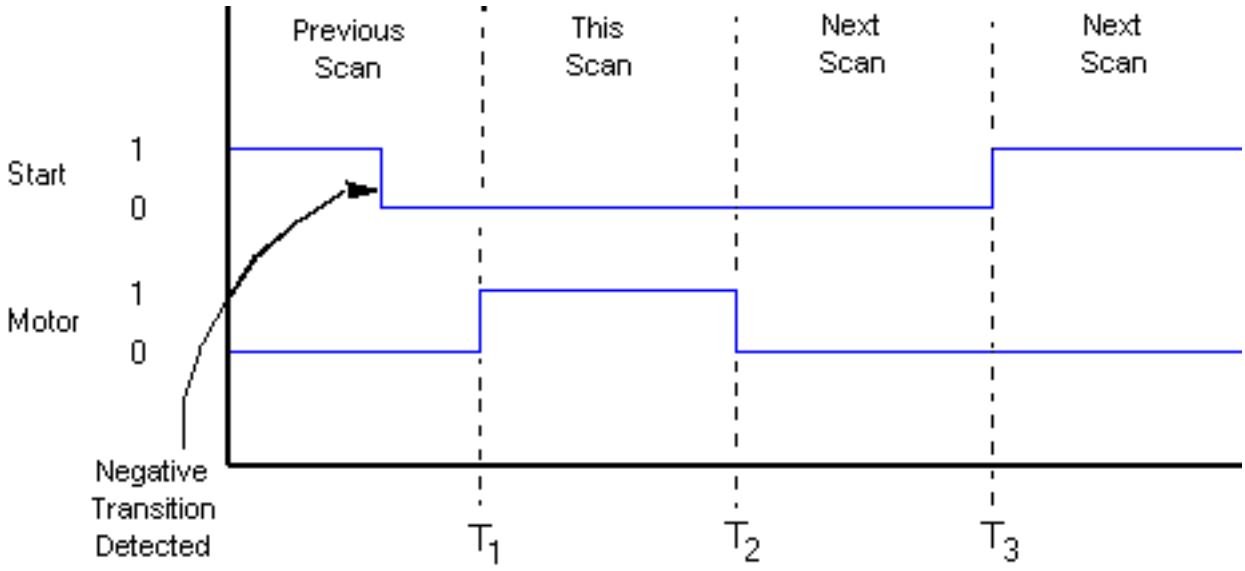
The parameter is a BOOL data type.

Example

The following rung diagram illustrates a typical application of the NT instruction. If 'Start' is OFF(0) and was ON(1) during the last scan, 'Motor' will turn ON(1).



The following timing diagram further illustrates the operation. 0=OFF, 1=ON

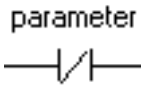


T_1: 'Start' had a negative transition during the previous scan and is OFF(0), so the NT contact closes. Power flows to the OUT coil turning 'Motor' ON(1).

T_2: A negative transition of 'Start' is not detected during the last scan, so the NT contact opens. Power is removed from the OUT coil turning 'Motor' OFF(0).

T_3: 'Start' turns ON(1) but the NT contact remains open because a negative transition has not occurred. No power flows to the OUT coil instruction thus 'Motor' remains OFF(0).

Normally Closed Contact (NC)



The NC instruction allows power to pass when the parameter is OFF(0).

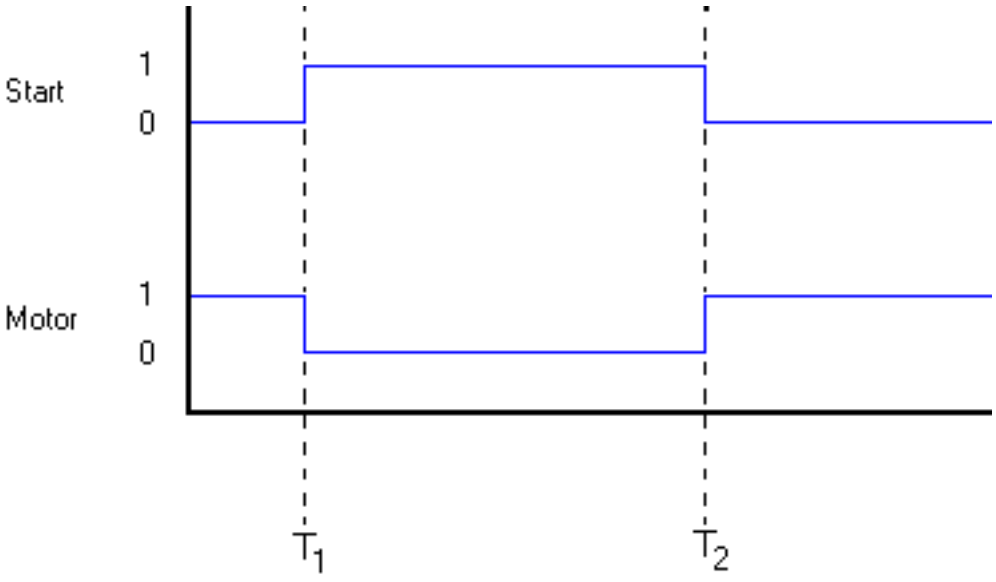
Parameter Data Types

The parameter is a BOOL data type.

Example

The following rung diagram illustrates a typical application of the NC instruction. When 'Start' turns ON(1), 'Motor' is turned OFF(0).

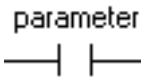
The following timing diagram further illustrates the operation. 0=OFF,1=ON



T_1: 'Start' turns ON(1) opening the NC contact. Power is removed from the OUT coil turning 'Motor' OFF(0).

T_2: 'Start' turns OFF(0) closing the NC contact. Power flows to the OUT coil turning 'Motor' ON(1).

Normally Open Contact (NO)



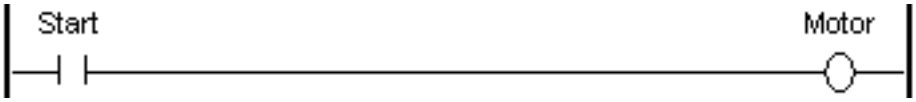
The NO instruction allows power to pass when the parameter is ON(1).

Parameter Data Types

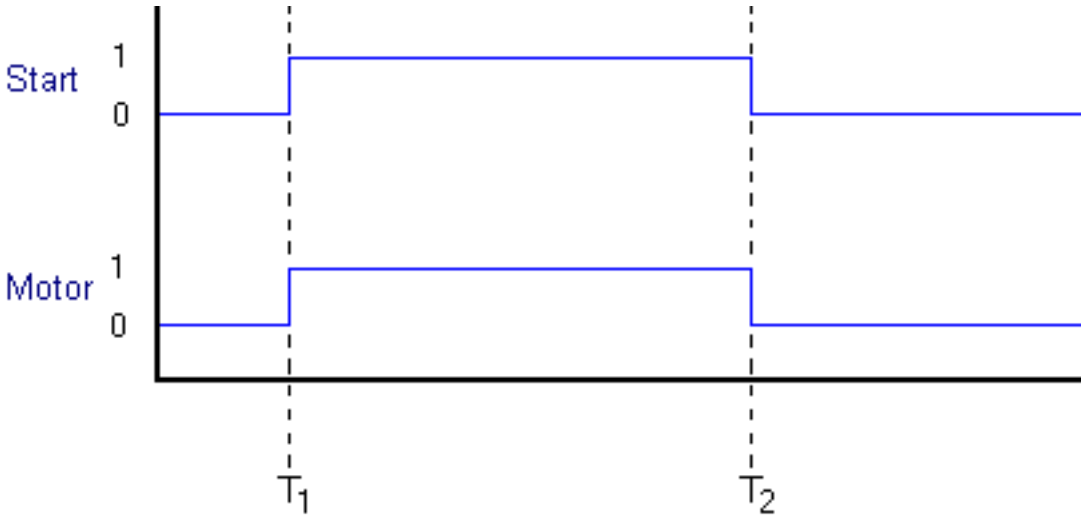
The parameter is a BOOL type.

Example

The following rung diagram illustrate a typical application of the NO instruction. When 'Start' is ON(1), power will pass and 'Motor' will turn ON(1).



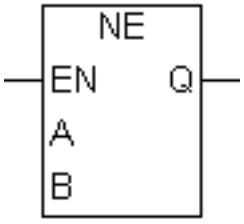
The following timing diagram further illustrates the operation. 0=OFF, 1=ON



T_1: 'Start' turns ON(1) closing the normally open contact. Power flows to the OUT coil turning 'Motor' ON(1).

T_2: 'Start' turns OFF(0) opening the normally open contact. Power is removed from the OUT coil turning 'Motor' OFF(0).

Not Equal (NE)



The NE instruction passes power if **A** is not equal to **B**.

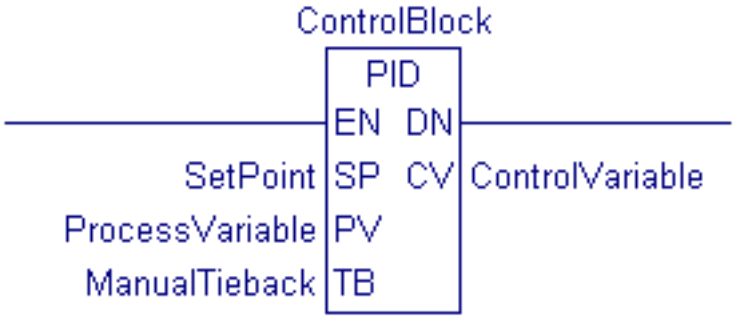
Parameter Data Types

The following parameter data types can be used with the NE instruction.

A, B can be:
- DINT
- DINT constant
- LREAL
- LREAL constant

Warning: Comparing LREAL values may produce unexpected results. For example, a calculation might result in 1.99999999999, which is not equal to 2.

PID Instruction Parameters

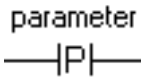


Parameter	Description
Control Block	Holds status bits and values that can be tuned.
EN	Enable flag PID is in automatic mode if true (powered), or manual mode if not. (BOOL input)
Setpoint (SP)	DINT input
Process Variable (PV)	DINT input
Manual Tieback (TB)	This is copied to CV when in manual mode, and is used in bumpless transfer calculation. (DINT input)
DN	Done flag Instruction always passes power, so this has same state as EN. (BOOL output)
Controlled Variable (CV)	DINT output

See Also

What is a PID?, PID Equation, Logic Developer - PC PID Dialog Box

Positive Transition Contact (PT)



The PT instruction allows power to pass if the parameter was OFF(0) during the previous scan but is ON(1) now. In other words, the PT instruction will pass power if the parameter has gone from OFF(0) to ON(1) during the last scan.

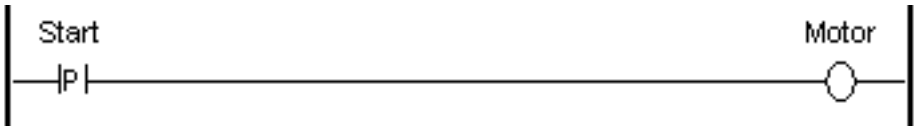
For the first scan the previous state is considered to be OFF(0); the PT instruction will pass power if the parameter has turned ON(1) since starting the program.

Parameter Data Types

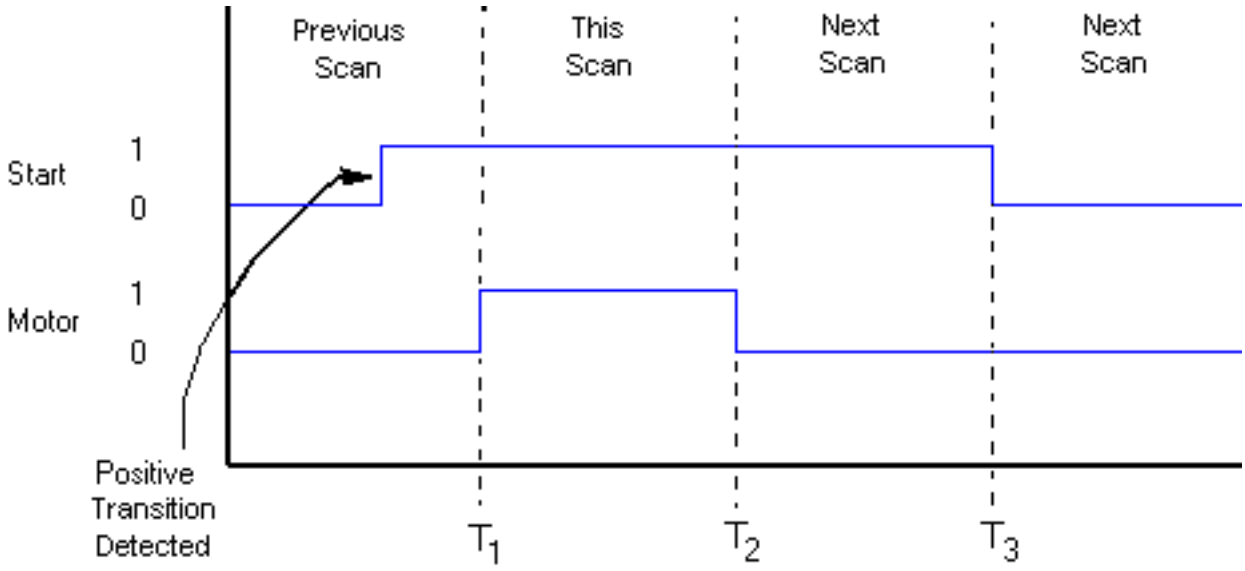
The parameter is a BOOL type.

Example

The following rung diagram illustrates a typical application of the PT instruction. If 'Start' has changed from OFF(0) to ON(1) since the last time the rung was scanned, 'Motor' will be turned ON(1).



The following timing diagram further illustrates the operation. 0=OFF, 1=ON

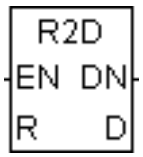


T_1: 'Start' had a positive transition during the previous scan and is ON(1), so the PT contact closes. Power flows to the OUT coil turning 'Motor' ON(1).

T_2: A positive transition of 'Start' is not detected during the last scan, so the PT contact opens. Power is removed from the OUT coil turning 'Motor' OFF(0).

T_3: 'Start' turns OFF(0) but the PT contact remains open because a positive transition has not occurred. No power flows to the OUT coil instruction thus 'Motor' remains OFF(0).

Radians to Degrees (R2D)



The R2D instruction converts the angle **R** from radians to degrees, and stores the result in **D**.

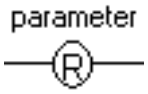
This instruction always passes power.

Parameter Data Types

The following data types can be used with the R2D instruction.

If **R** is a:	then **D** is a:
DINT or LREAL	LREAL

Reset Coil (RST)



The RST instruction turns the parameter OFF(0) when the coil receives power. The parameter remains OFF(0) until explicitly turned ON(1) by another instruction such as a Set Coil (SET).

This is an output instruction, so it must be the last instruction on a rung.

Variations

When the assigned parameter is specified as retentive the following alternate symbol is displayed in the logic.

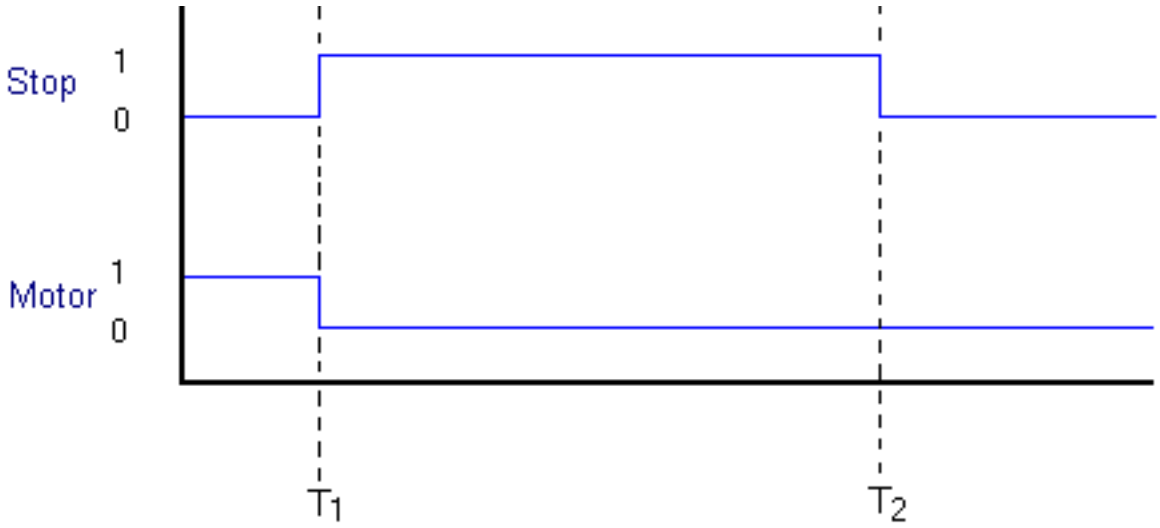
Parameter Data Types

The parameter is a BOOL data type. By default, the BOOL variable automatically created when inserting an RST instruction will be non-retentive (set to its initial state when the Controller starts following a shutdown or reset).

Example

The following rung diagram illustrates a typical application of an RST instruction. When 'Stop' turns ON(1) 'Motor' is set OFF(0). After execution of this rung, 'Motor' will remain OFF(0) until turned ON(1) with another instruction.

The following timing diagram further illustrates the operation.



T_1: 'Stop' turns ON(1) closing the normally open contact. Power flows to the RST coil, resetting 'Motor' OFF(0).

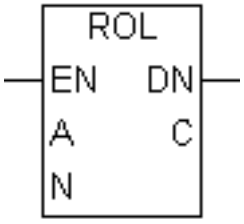
T_2: 'Stop' turns OFF(0) opening the normally open contact. Power is removed from the RST coil, having no effect. 'Motor' remains OFF(0) until turned ON(1) by another instruction elsewhere in the ladder logic program.

Return

When the Return instruction receives power it causes control to return prematurely from a subroutine. Normally, control is returned automatically by the SUB END label. Execution continues at the rung following the Jump Subroutine (JSR) instruction that called the subroutine.

Return must be the last instruction on a rung.

Rotate Left (ROL)



The ROL instruction shifts the bits in **A** left **N** positions. Bits shifted off the left end (most significant bit) are rotated back into the right end (least significant bit). The result is placed in **C**.

The ROL instructions always passes power.

Variations

There are two variations:

- If **A** and **C** are DINTs, a simple 32-bit rotation is done. **N** must be between 0 and 31.
- **A** and **C** may be arrays of the same size. In this case, the array is treated as a large DINT. This means bits are shifted from one element to the next, rather than rotating only within each element. The most significant bit of the highest number element of the array is rotated into the least significant bit of element 0. **N** must be between 0 and (32 x array size, less 1), inclusive.

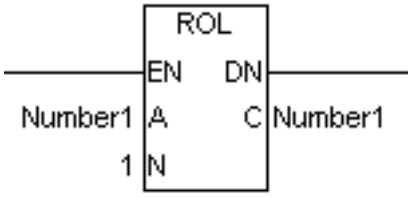
Parameter Data Types

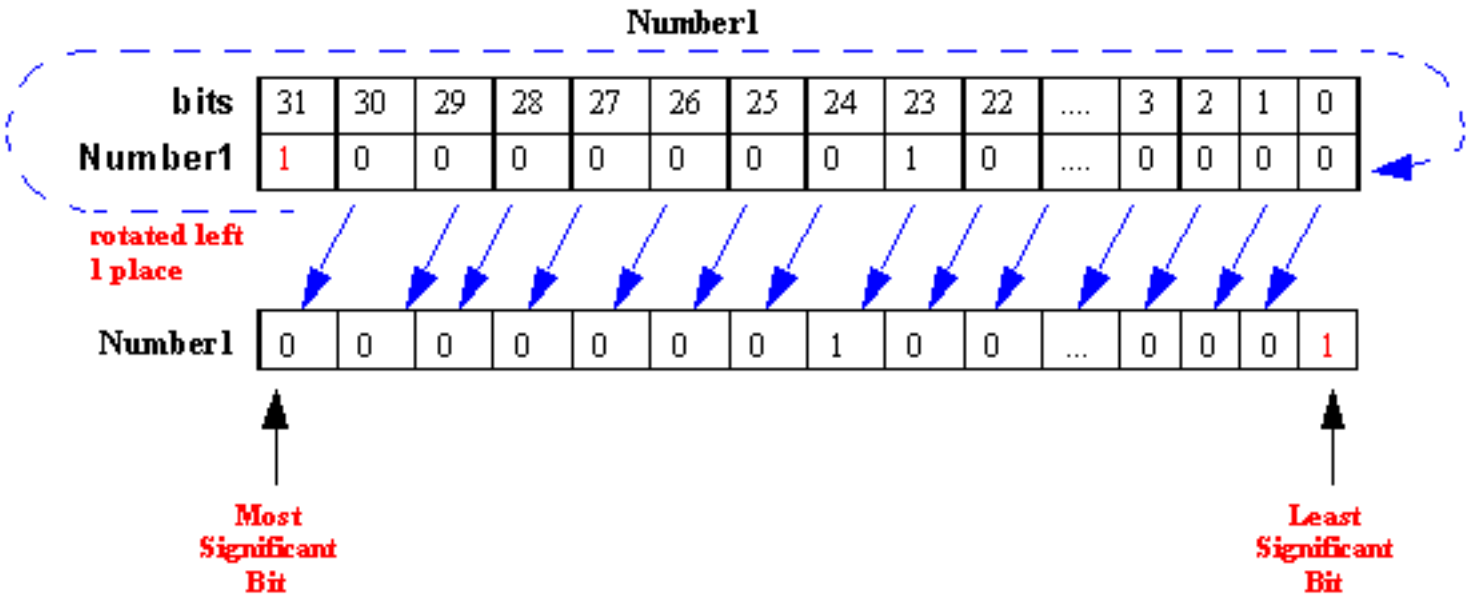
The parameters assigned to the ROL instruction must be of the following types:

If **A** is:	then **C** is:	and **N** is:
DINT	DINT	DINT or DINT constant
DINT array	DINT array of the same size as **A**	DINT or DINT constant
DINT constant	DINT	DINT or DINT constant

Example

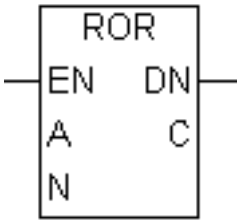
The following example shows a single-bit rotate left of the DINT variable 'Number1'.





Note: #Overflow is turned ON(1) if **N** is out of range. The result is undefined.

Rotate Right (ROR)



The ROR instruction shifts the bits in **A** right **N** positions. Bits shifted out of the right end (least significant bit) are rotated back into the left end (most significant bit). The result is placed in **C**.

Variations

There are two variations:

- If neither **A** nor **C** is an array, a simple 32-bit rotation is performed. **N** must be between 0 and 31, inclusive.
- **A** and **C** can be arrays of the same size. In this case, the array is treated as a large DINT. That is, bits are shifted from one element to the next, rather than rotating only within each

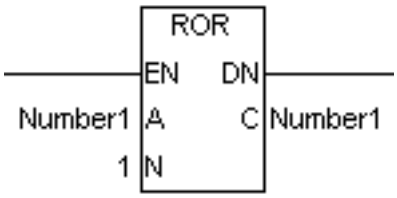
Parameter Data Types

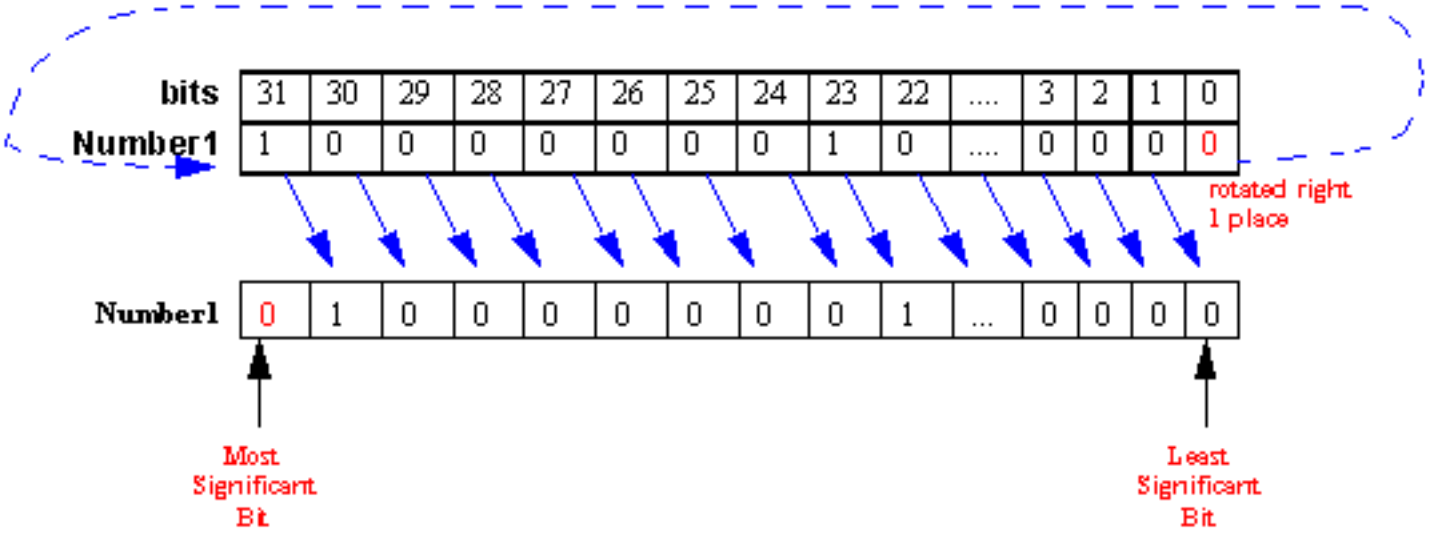
The parameters assigned to the ROL instruction must be of the following types:

If **A** is:	then **C** is:	and **N** is:
DINT	DINT	DINT or DINT constant
DINT array	DINT Array of the same size as **A**	DINT or DINT constant
DINT constant	DINT	DINT or DINT constant

Example

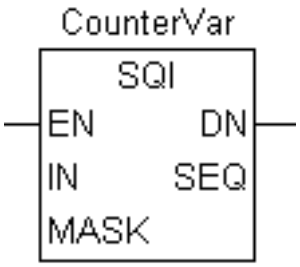
The following diagram illustrates a single-bit rotate right of the DINT variable 'Number1'.





Note: #Overflow is turned ON(1) if **N** is out of range. The result is undefined.

Sequencer Input (SQI)

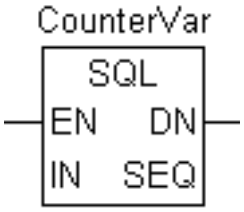


The SQI instruction does the following when it receives power:

1. Ensures that the current step, *CounterVar.CV*, is within the range of the size of the SEQ array. If it is not within this range, the SQI instruction does nothing more and doesn't pass power.
2. Compares the input states with the bits in the current step of the sequencer array, comparing only those bits whose corresponding mask bit is on.
3. Passes power if all the specified bits are equal.

Unlike SQO, this instruction does not increment the current step.

Sequencer Load (SQL)



The SQL instruction does the following when it receives power:

1. The instruction resets *CounterVar.CV* to 1 if the EN input just changed from false to true and any of the following conditions are met
   - This is the first transition.
   - A previous sequence has finished (as indicated by *CounterVar.Q* being *on).*
   - The reset bit *CounterVar.R* has been set.

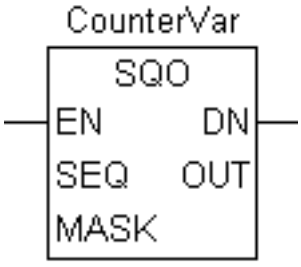
   Otherwise, it moves to the next step by adding 1 to *CounterVar.CV*.

1. Ensures the current step, *CounterVar.CV*, is within the range of the size of the SEQ array and copies the states of the inputs to the current step of the sequencer array. If *CounterVar.CV* is out of the range the states of the inputs will not be copied.
2. If *CounterVar.CV* = *CounterVar.PV* or *CounterVar.CV* points to the last element of the sequencer array, *CounterVar.Q* and *CounterVar.QU* are turned on and pass power. Otherwise, *CounterVar.Q* and *CounterVar.QU* are turned off.
3. Turns *CounterVar.R* off.

Notes

- On the first scan, the previous rung state is considered to be false.
- On the first scan only, the instruction leaves *CounterVar.CV* unchanged.

Sequencer Output (SQO)



The SQO instruction does the following when it receives power:

1. The instruction resets *CounterVar.CV* to 1 if the EN input just changed from false to true and any of the following conditions are met
    - This is the first transition.
    - A previous sequence has finished (as indicated by *CounterVar.Q* being on).
    - The reset bit *CounterVar.R* has been set.

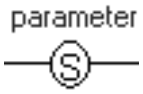
    Otherwise, it moves to the next step by adding 1 to *CounterVar.CV*.

1. Makes sure the current step, *CounterVar.CV*, is within the range of the size of the SEQ array and copies the bits in the current step of the sequencer array to the outputs, but only if the corresponding mask bit is on. Otherwise, it leaves the outputs unchanged.
2. If *CounterVar.CV* = *CounterVar.PV* or *CounterVar.CV* points to the last element of the sequencer array, *CounterVar.Q* and *CounterVar.QU* are turned on and pass power. Otherwise, *CounterVar.Q* and *CounterVar.QU* are turned off.
3. Turns *CounterVar.R* off.

Notes

- On the first scan, the previous rung state is considered to be false.
- On the first scan only, the instruction leaves *CounterVar.CV* unchanged.

Set Coil (SET)



The SET instruction turns the parameter ON(1) if the coil receives power. It will stay ON(1) until explicitly turned OFF(0) by another instruction such as a Reset Coil (RST).

SET is an output instruction, so it must be the last one on a rung.

Variations

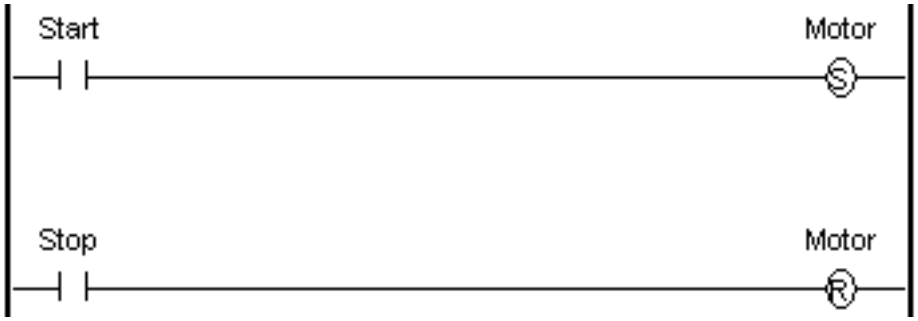
When the assigned parameter is specified as retentive the following alternate symbol is displayed in the logic:

Parameter Data Types

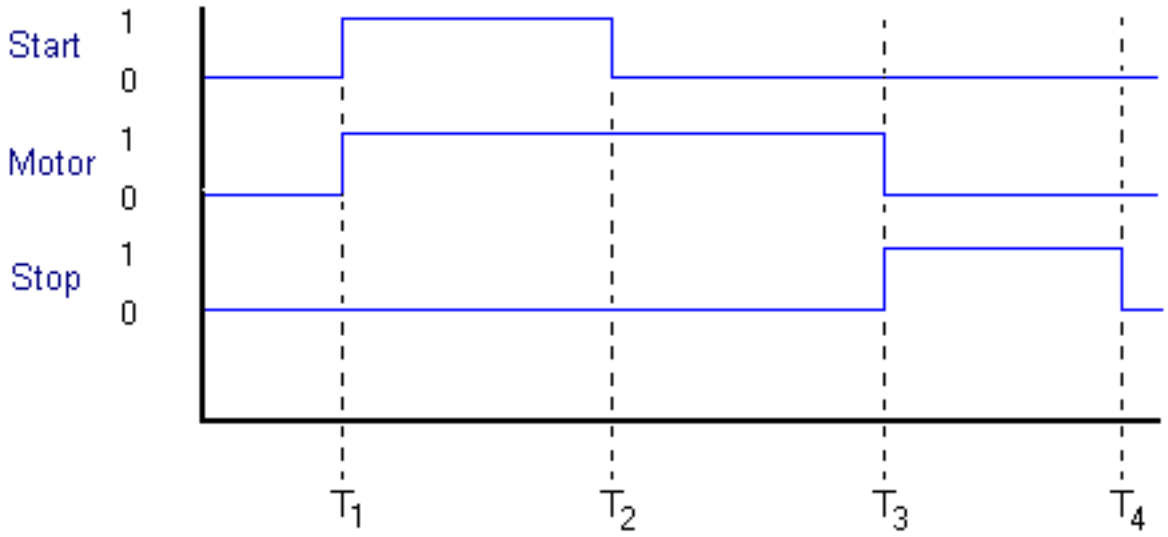
The parameter is a BOOL type. By default, the BOOL variable automatically created when inserting an RST instruction will be non-retentive (set to its initial state when the Controller starts following a shutdown or reset).

Example

The following rung diagram shows how the Set Coil instruction could be used to set the BOOL variable 'Motor' ON(1) when another BOOL variable 'Start' turns ON(1). When 'Stop' turns ON(1) 'Motor' is reset OFF(0) by the Reset Coil instruction.



The following timing diagram further illustrates the operation.



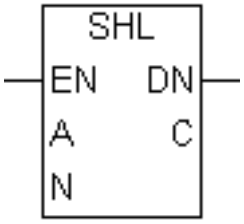
T_1: 'Start' turns ON(1) closing the normally open contact. Power flows to the SET coil setting 'Motor' ON(1).

T_2: 'Start' turns OFF(0) opening the normally open contact. Power is removed from the SET coil, having no affect on 'Motor'.

T_3: 'Stop' turns ON(1) closing the normally open contact. Power flows to the RST coil resetting 'Motor' OFF(0).

T_4: 'Stop' turns OFF(0) opening the normally open contact. Power is removed from the RST coil, having no affect on 'Motor'.

Shift Left (SHL)



The SHL instruction shifts the bits in **A** left **N** positions. Bits shifted off the left end (most significant bit) are lost, and the now-empty positions at the right end (least significant bit) are turned OFF(0). The result is placed in **C**. Each scan causes **A** to be shifted **N** times.

This instruction always passes power.

Variations

There are two variations:

- If neither **A** nor **C** is an array, a simple 32-bit shift is done. **N** must be between 0 and 31, inclusive.
- **A** and **C** can be arrays of the same size. In this case, the array is treated as a large DINT. That is, bits are shifted from one element to the next, rather than being lost off the left end of each element. The most significant bit of the highest numbered element of the array is shifted out and lost while the least significant bit of element 0 is turned OFF. **N** must be between 0 and (32 x array size, less 1), inclusive.

Parameter Data Types

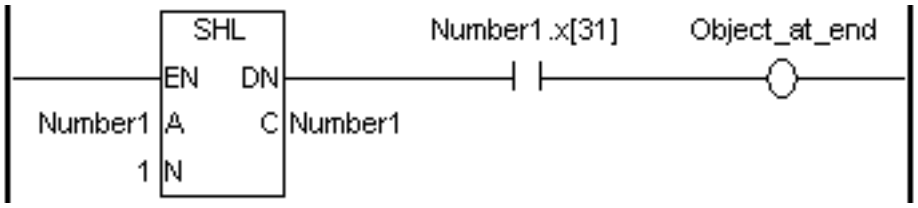
The parameters assigned to the SHL instruction must be of the following types.

If **A** is:	then **C** is:	and **N** is:
DINT	DINT	DINT or DINT constant
DINT array	DINT array of the same size as **A**	DINT or DINT constant

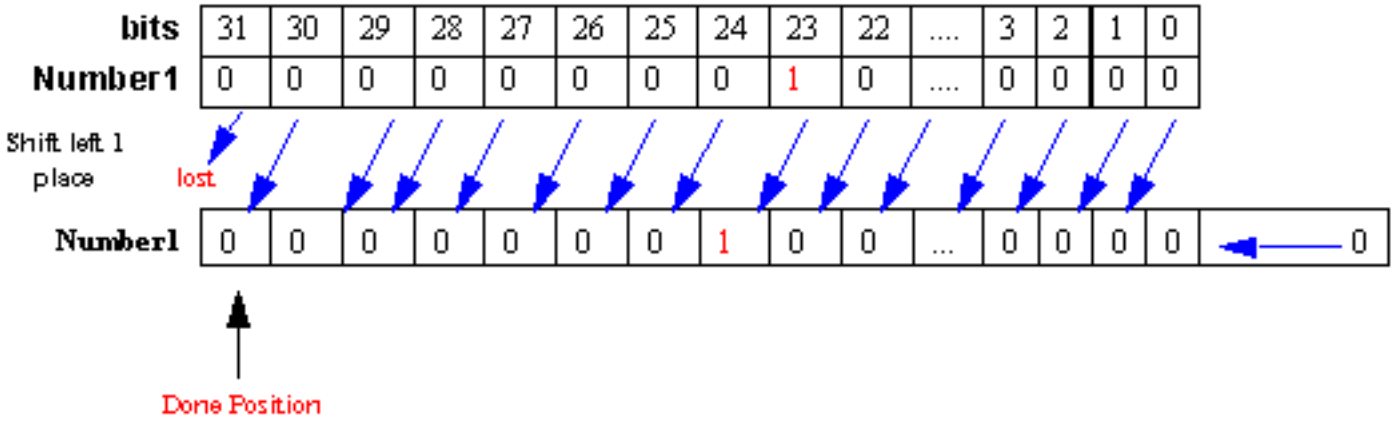
DINT constant DINT DINT or DINT constant

Example

The following example shows a single-bit shift used to track the position of an object. The object is represented as an ON(1) bit and each bit position of the DINT variable 'Number1' represents a position in the real world. All other bits in the variable are OFF(0). Each program scan (or any interval) moves the object (shift left) to the next position. When it reaches the last bit in the DINT (31) the parameter 'Number1.X[31]' is turned ON(1) causing 'Object_at_end' to turn ON(1), signaling the operation is complete.

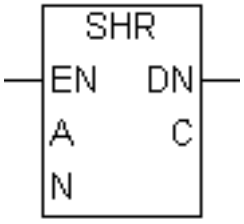


The following diagram further illustrates the operation.



Note: #Overflow is turned ON(1) if **N** is out of range. The result is undefined.

Shift Right (SHR)



The SHR instruction shifts the bits in **A** right **N** positions. Bits shifted off the right end (least significant bit) are lost, and the now-empty positions at the left end (most significant bit) are turned OFF(0). The result is placed in **C**. For each scan **A** is shifted **N** times.

This instruction always passes power.

Variations

There are two variations:

- If neither **A** nor **C** is an array, a simple 32-bit shift is done. **N** must be between 0 and 31, inclusive.
- **A** and **C** can be arrays of the same size. In this case, the array is treated as a large DINT. That is, bits are shifted from one element to the next, rather than being lost off the right end of each element. The least significant bit of element 0 is shifted out and lost while the most significant bit of the highest numbered element of the array is turned OFF(0). **N** must be between 0 and (32 x array size, less 1), inclusive.

Parameter Data Types

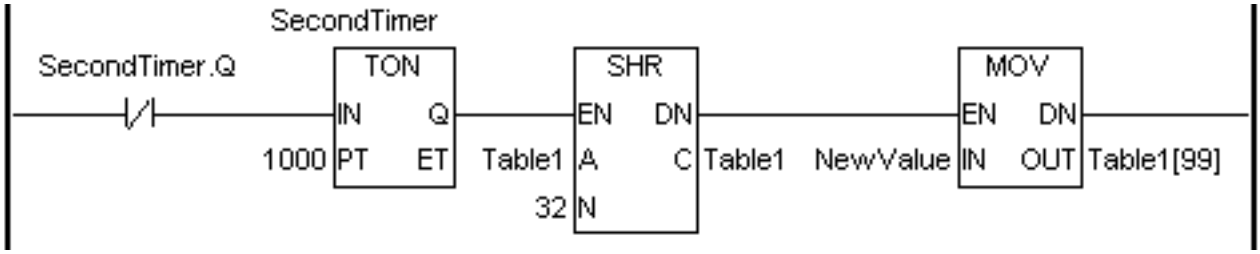
The parameters assigned to the SHR instruction must be of the following types:

If **A** is:	then **C** is:	and **N** is:
DINT	DINT	DINT or DINT constant
DINT array	DINT array of the same size as **A**	DINT or DINT constant

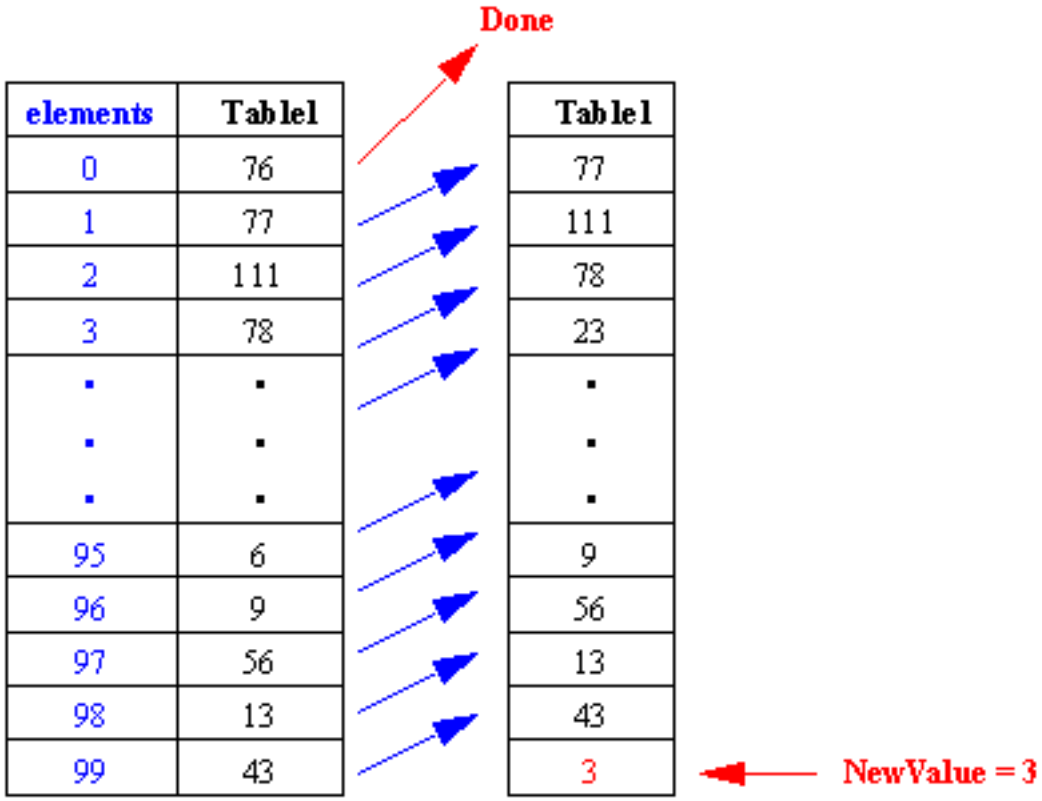
DINT constant DINT DINT or DINT constant

Example

The following example shows how a SHR instruction can be used to move values through a queue. A shift of 32 bits shifts the entire 32 bit DINT. The queue is the 100 element DINT array 'Table1'. Every second the DINT values are moved 1 place up in the queue (towards position 0) and a new value is placed at the end of the queue ('Table1[99]').



The following diagram further illustrates the operation.



Note: #Overflow is turned ON(1) if **N** is out of range. The result is undefined.

Sine (SIN)

The SIN instruction calculates the sine of **A** and stores the result in **B**.

This instruction always passes power.

The angle is specified in radians.

Parameter Data Types

The following data types can be used with the SIN instruction.

If **A** is a:	then **B** is a:
DINT or LREAL	LREAL

Square Root (SQRT)

The SQRT instruction calculates the positive square root of **A** and stores the result in **B**.

This instruction always passes power.

Parameter Data Types

The following data types can be used with the SQRT instruction.

If **A** is a:	then **B** is a:
DINT or LREAL	LREAL

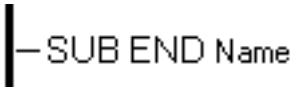
START Label

The START label marks the beginning of the main program area and is included by default in every ladder program.

Rungs between the START and END labels are executed during every scan. Rungs before the START label are executed only during the first scan (initialization logic). No instructions can be inserted on the START rung.

Note: The START label can be specified as the destination of a JMP instruction if the JMP is inserted before the START label.

SUB END Label

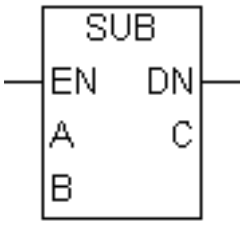


A SUB END label marks the end of a subroutine and is automatically placed when you insert a subroutine. A name is required (that is, the name of the subroutine). When execution encounters this label, control is transferred to the rung following the Jump Subroutine (JSR) instruction that called this subroutine.

SUB START Label

A SUB START label marks the beginning of a subroutine and is automatically placed when you insert a subroutine. A name is required (that is, the name of the subroutine). When program control jumps to a subroutine (using a JSR instruction), execution proceeds from the rung following the SUB START label.

Subtract (SUB)



The SUB instruction subtracts **B** from **A**, placing the difference in **C**.

This instruction always passes power.

Variations

If both **A** and **B** are DINTs (or DINT constants), the instruction performs a DINT subtraction. If either **A** or **B** are LREALs, both are converted to LREALs prior to performing a slower floating point subtraction.

Parameter Data Types

The following data types can be used with the SUB instruction.

If **A** is :	and **B** is:	then **C** is:
DINT DINT constant LREAL LREAL constant	DINT DINT constant LREAL LREAL constant	DINT or LREAL
TIME	TIME	TIME
DATE_AND_TIME	TIME	DATE_AND_TIME
DATE_AND_TIME	DATE_AND_TIME	TIME

Notes

- #Overflow is turned ON(1) if the result is too large for the destination variable (**C**). The result of a DINT subtraction is truncated; the result of a floating-point subtraction is undefined.
- If either **A** or **B** are LREALs, both are converted to LREALs prior to the subtraction. The result is placed in **C** and truncated if **C** is a DINT.

Tangent (TAN)

The TAN instruction calculates the tangent of **A** and stores the result in **B**.

This instruction always passes power.

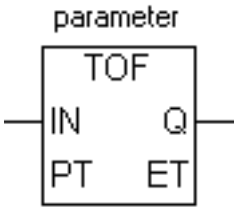
The angle is specified in radians.

Parameter Data Types

The following data types can be used with the TAN instruction.

If **A** is a:	then **B** is a:
DINT or LREAL	LREAL

Timer Off Delay (TOF)



The Timer Off Delay (TOF) instruction provides a delay of **PT** milliseconds after power is removed from **IN**. The TOF instruction passes power as soon as power is applied to **IN** and will continue to pass power until **PT** milliseconds after power has been removed from **IN**.

Parameter Data Types

The parameter assigned to the TOF instruction is a TIMER structure variable described in the following table.

Element	Description	Data type	Example
PT	Preset time	DINT	MyTimer.PT
ET	Elapsed time	DINT	MyTimer.ET
TI	Timing bit	BOOL	MyTimer.TI
Q	Output enable bit	BOOL	MyTimer.Q

The preset time 'MyTimer.PT' must be set manually or by another instruction executed before the TOF instruction.

Detailed Operation

When the TOF instruction receives power:

- The elapsed time 'MyTimer.ET' is reset to zero.
- The timing bit 'MyTimer.TI' is turned OFF(0).
- The output enable bit 'MyTimer.Q' is turned ON(1) (the instruction passes power).

When the instruction stops receiving power:

- The elapsed time 'MyTimer.ET' begins counting upward (in milliseconds).
- The timing bit 'MyTimer.TI' is turned ON(1).
- The output enable bit 'MyTimer.Q' remains ON(1).

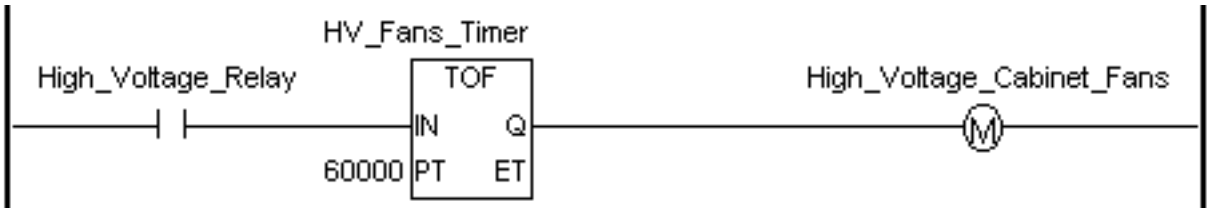
When the elapsed time 'MyTimer.ET' equals the preset time 'MyTimer.PT' after incrementing:

- The elapsed time 'MyTimer.ET' stays fixed at the preset value.
- The timing bit 'MyTimer.TI' is turned OFF(0).
- The output enable bit 'MyTimer.Q' is turned OFF(0).

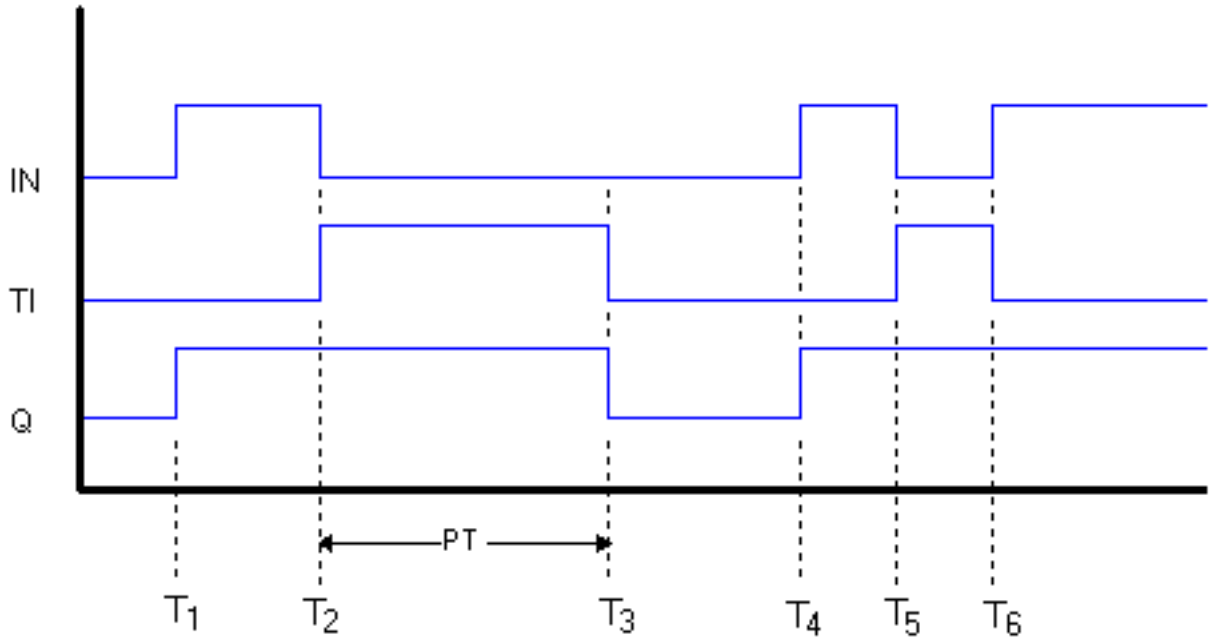
Example

The following rung diagram illustrates how a Timer Off Delay would be used to keep the fans in a high voltage cabinet running for 1 minute (60,000 ms) after the high voltage is turned off (as well as keep them on while high voltage is on).

As long as 'High_Voltage_Relay' is ON(1) the TOF instruction passes power and 'High_Voltage_Cabinet_Fans' is ON(1). When 'High_Voltage_Relay' turns OFF(0), the TOF begins counting down. One minute later the TOF stops passing power and 'High_Voltage_Cabinet_Fans' turns OFF(0).



The following timing diagram further illustrates the operation of a TOF.



T_1: The timer input **IN** receives power, timing bit **TI** remains OFF(0), the output enable bit **Q** turns ON(1), elapsed time **ET** is reset to 0.

T_2: Power is removed from the timer input **IN**, the timer starts timing (**TI** turns ON(1)), the output enable bit **Q** remains ON(1).

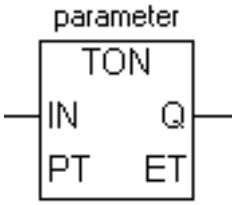
T_3: After elapsed time **ET** equals preset time **PT**, the output enable bit **Q** turns OFF(0), timer stops timing (**TI** turns OFF(0)), and the elapsed time stays fixed at preset time (**ET**=**PT**).

T_4: The timer input **IN** receives power, timing bit **TI** remains OFF(0), the output enable bit **Q** turns ON(1), elapsed time **ET** is reset to 0.

T_5: Power is removed from the timer input **IN**, the timer starts timing (**TI** turns ON(1)), the output enable bit **Q** remains ON(1).

T_6: The timer input **IN** receives power before elapsed time **ET** equals preset time **PT**, the timer stops timing (**TI** turns OFF(0)), the output enable bit **Q** remains ON(1), elapsed time **ET** is reset to 0.

Timer On Delay (TON)



The Timer On Delay (TON) instruction adds a delay of **PT** milliseconds from when an event actually occurs (that is, from when **IN** receives power). TON will pass power **PT** milliseconds after **IN** has power applied, as long as power is still applied to **IN**.

Parameter Data Types

The parameter assigned to the TON instruction is a TIMER structure variable described in the following table. The example is shown for a TIMER variable named 'MyTimer'.

Element	Description	Data type	Example
PT	Preset time	DINT	MyTimer.PT
ET	Elapsed time	DINT	MyTimer.ET
TI	Timing bit	BOOL	MyTimer.TI
Q	Output enable bit	BOOL	MyTimer.Q

An DINT constant data type can optionally be assigned to the **PT** input. The preset time 'MyTimer.PT' can also be set by another instruction executed before the TOF instruction.

Detailed Operation

When the TON instruction receives power:

- The elapsed time 'MyTimer.ET' begins counting upward (in milliseconds).
- The timing bit 'MyTimer.TI' is turned ON(1).
- The output enable bit 'MyTimer.Q' is turned OFF(0).

When the elapsed time 'MyTimer.ET' equals the preset time 'MyTimer.PT' after incrementing:

- The elapsed time 'MyTimer.ET' stays fixed at the preset value.
- The timing bit 'MyTimer.TI' is turned OFF(0).
- The output enable bit 'MyTimer.Q' is turned ON(1) (the instruction passes power).

When the instruction stops receiving power:

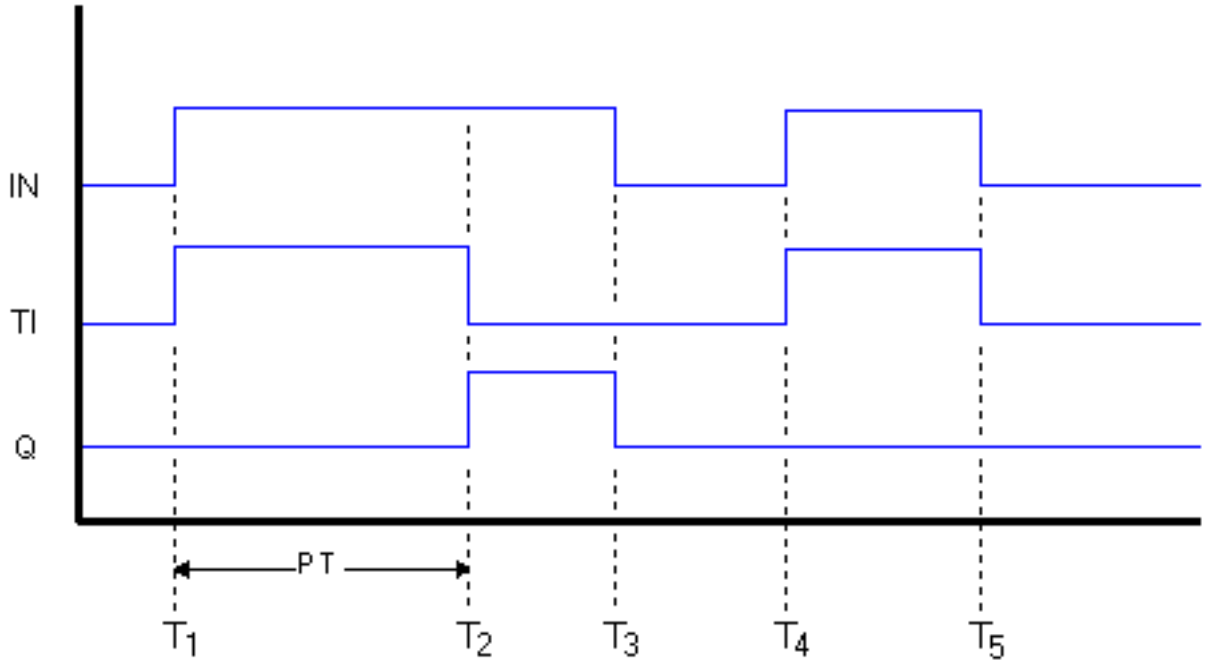
- The elapsed time 'MyTimer.ET' is reset to zero.
- The timing bit 'MyTimer.TI' is turned OFF(0).
- The output enable bit 'MyTimer.Q' is turned OFF(0).

Example

The following rung diagram illustrates how a Timer On Delay would be used to delay a start signal. A drive needs 5 seconds after it is enabled (powered) to charge up capacitors before it can be started.

When 'Enable_Drive' turns ON the TON starts counting. Five seconds later the TON passes power and 'Start_Drive' turns ON(1).

The following timing diagram further illustrates the operation of the TON instruction.



T_1: Power is applied to the timer input **IN**, timing bit **TI** turns ON(1), the timer begins timing (**ET** increments). **Q** remains turned OFF(0).

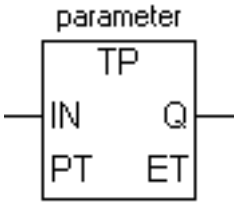
T_2: After time **PT**, elapsed time **ET** equals preset time **PT**, the output enable bit **Q** turns ON(1) and the elapsed time stays fixed at the preset time (**ET**=**PT**).

T_3: Power is removed from **IN**, the output enable bit **Q** is turned OFF(0), and elapsed time **ET** is reset to 0.

T_4: Power is applied to the timer input **IN**, the timing bit **TI** turns ON(1), and the timer begins timing (**ET** increments).

T_5: Power is removed from **IN** before elapsed time **ET** equals preset time **PT**, the output enable bit **Q** remains OFF(0), elapsed time **ET** is reset to 0.

Timer Pulse (TP)



The Timer Pulse (TP) instruction outputs a pulse **Q** for a duration of **PT** milliseconds once triggered by **IN**.

Parameter Data Types

The parameter assigned to the TP instruction is a TIMER structure variable described in the following table. The example is given for a TIMER variable named 'MyTimer'.

Element	Description	Data type	Example
PT	Preset time	DINT	MyTimer.PT
ET	Elapsed time	DINT	MyTimer.ET
TI	Timing bit	BOOL	MyTimer.TI
Q	Output enable bit	BOOL	MyTimer.Q

The preset time 'MyTimer.PT' must be set manually or by another instruction executed before the TOF instruction.

Detailed Operation

When the TP instruction receives power:

- The elapsed time 'MyTimer.ET' begins counting upward (in milliseconds).
- The timing bit 'MyTimer.TI' is turned ON(1).
- The output enable bit 'MyTimer.Q' is turned ON(1) as the instruction passes power.

When the elapsed time 'MyTimer.ET' equals the preset time 'MyTimer.PT' after timing:

- The elapsed time 'MyTimer.ET' stays fixed at the preset value if the instruction is still receiving power. If not, it resets immediately to zero.
- The timing bit 'MyTimer.TI' is turned OFF(0).
- The output enable bit 'MyTimer.Q' is turned OFF(0) (the instruction stops outputting power).

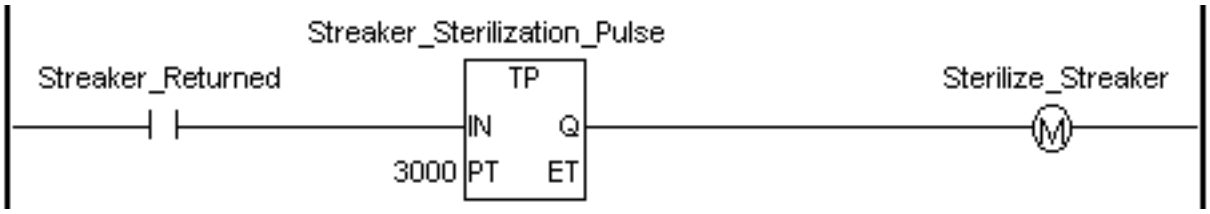
When the instruction stops receiving power:

- The elapsed time 'MyTimer.ET' is reset to zero only if it has already reached the value of the preset time 'MyTimer.PT'. Otherwise, it keeps timing and 'MyTimer.Q' remains ON(1).

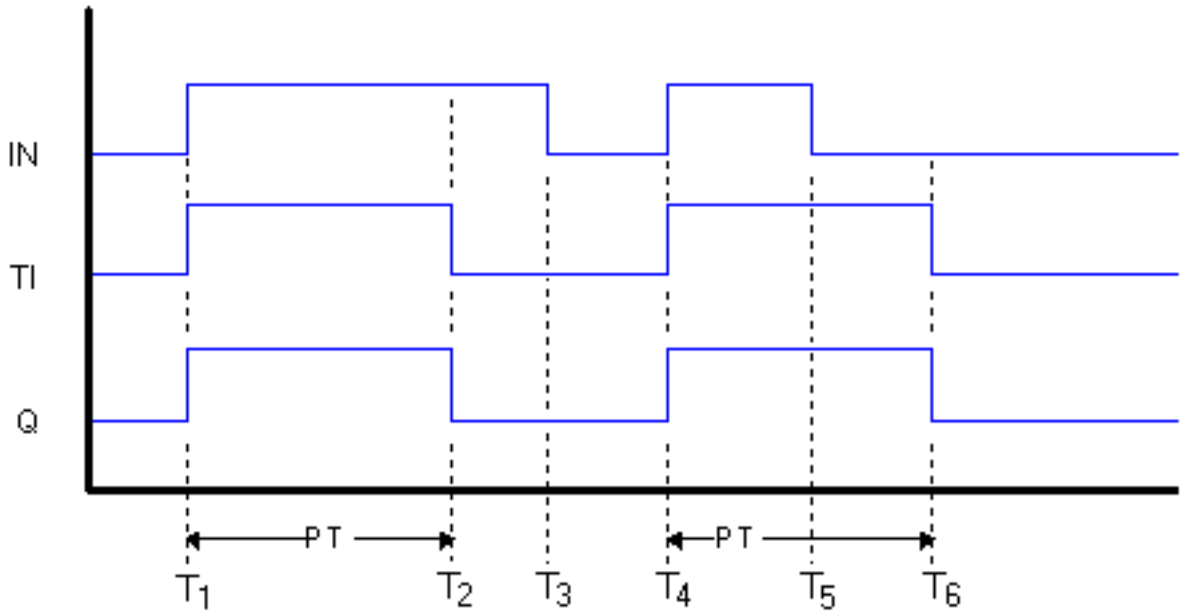
Example

The following example demonstrates how a Timer Pulse instruction is used to turn on an output for a 3 second pulse. The Streaker is sterilized for 3 seconds after it is returned to its home position.

When 'Streaker_Returned' turns ON(1), the TP instruction passes power and 'Sterilize_Streaker' turns ON(1). Three seconds later, the timer output turns OFF(0) and so does 'Sterilize_Streaker'.



The following timing diagram further illustrates the operation of the TP instruction.



T_1: The timer input **IN** is true, the timer starts timing (**TI** turns ON(1)), the output enable bit **Q** turns ON(1).

T_2: After the elapsed time **ET** equals the preset time **PT**, the output enable bit **Q** turns OFF(0), timer stops timing (**TI** turns OFF(0)), and the elapsed time stays fixed at preset time (**ET**=**PT**).

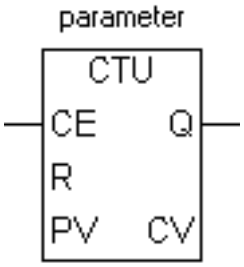
T_3: The timer input **IN** is false, and the elapsed time **ET** is reset to 0.

T_4: The timer input **IN** is true, the timer starts timing (**TI** turns ON(1)), the output enable bit **Q** remains ON(1).

T_5: The timer input **IN** is false, the timer keeps timing (**TI** stays ON(1)), the output enable bit **Q** remains ON(1).

T_6: After time **PT** elapsed time **ET** equals the preset time **PT**, the output enable bit **Q** turns OFF(0), timer stops timing (**TI** turns OFF(0)), and the elapsed time **ET** is reset to 0 since the timer input **IN** is false.

Up Counter (CTU)



For every scan the CTU instruction receives power, it will increment by one. The instruction passes power when it has incremented up to a preset value or greater.

Parameter Data Types

The parameter assigned to the CTU instruction is a COUNTER structure variable described in the following table.

Element	Description	Data type	Example
PV	Preset value	DINT	MyCounter.PV
CV	Current value	DINT	MyCounter.CV
R	Reset bit	BOOL	MyCounter.R
UP	Counting UP bit	BOOL	MyCounter.UP
QU	Done UP bit	BOOL	MyCounter.QU
QD (n/a)	Done DOWN bit	BOOL	MyCounter.QD
Q	Output enable bit	BOOL	MyCounter.Q

The preset value 'MyCounter.PV' must be set manually or by another instruction executed before the CTU instruction.

Detailed Operation

The current value 'MyCounter.CV' is incremented by one when the CTU instruction receives power and the reset enable bit 'MyCounter.R' is OFF(0).

The output enable bit 'MyCounter.Q' is turned ON(1) and the instruction passes power when the current value 'MyCounter.CV' is equal to or greater than the preset value 'MyCounter.PV', after incrementing.

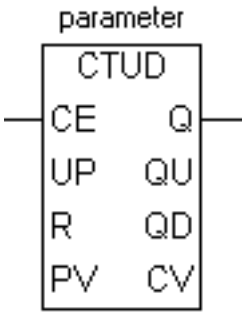
When the reset enable bit 'MyCounter.R' is ON(1), the current value 'MyCounter.CV' is reset to zero.

Example

In the following example the CTU will flag a fault after 5 arcs have been counted in a 1 minute period.

'MinuteTimer.Q' turns ON(1) for one scan each minute, causing the reset bit 'ArcCounter.R' to turn ON(1), thus zeroing the current value 'ArcCounter.CV'. If an arc occurred during this scan 'Arc' turns ON(1) and power is applied to the CTU instruction. If five arcs are counted within a minute, the CTU instruction passes power causing 'ArcFault' to turn ON(1).

Up/Down Counter (CTUD)



For every scan the CTUD receives power it will either increment or decrement by one. The CTUD instruction behaves like an Up Counter when **UP** is ON(1), otherwise it behaves like a Down Counter.

Parameter Data Types

The parameter assigned to the CTUD instruction is a COUNTER structure variable described in the following table.

Element	Description	Data type	Example
PV	Preset value	DINT	MyCounter.PV
CV	Current value	DINT	MyCounter.CV
R	Reset bit	BOOL	MyCounter.R
UP	Counting UP bit	BOOL	MyCounter.UP
QU	Done UP bit	BOOL	MyCounter.QU
QD	Done DOWN bit	BOOL	MyCounter.QD
Q	Output enable bit	BOOL	MyCounter.Q

An additional parameter of DINT constant type can optionally be assigned to the **PV** input. The preset value 'MyCounter.PV' can also be set by another instruction executed before the CTUD instruction.

Detailed Operation

When 'MyCounter.UP' is ON(1):

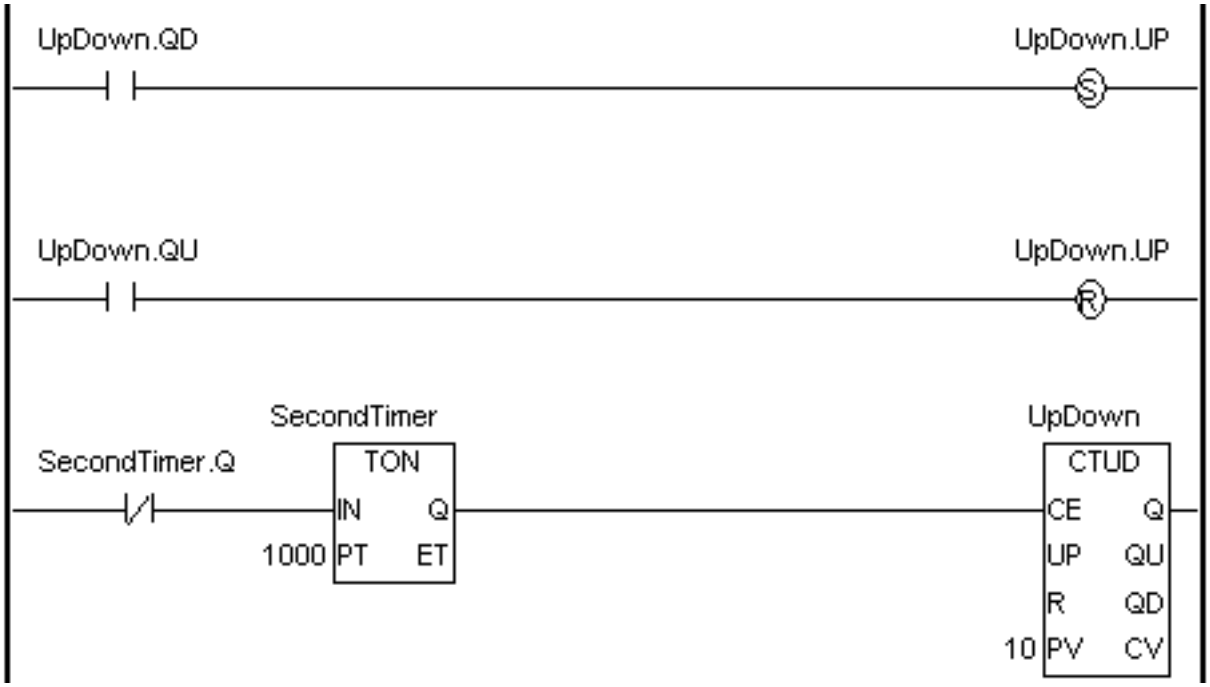
- If the current value 'MyCounter.CV', after incrementing, is equal to (or greater than) the preset value 'MyCounter.PV', 'MyCounter.Q' and 'MyCounter.QU' are turned ON(1).

When 'MyCounter.UP' is OFF(0):

- If the current value 'MyCounter.CV', after decrementing, is equal to (or less than) zero, 'MyCounter.Q' and 'MyCounter.QD' are turned ON(1).

Example

The following example uses a CTUD instruction to repeatedly count from 0 to 10 then back down to 0. The SecondTimer outputs a pulse to the up down counter every second. The **UP** bit is turned ON(1) when the counter hits 0, and turned OFF(0) when the counter hits the preset (10).



X to the Power of Y (EXPT)

The EXPT instruction calculates **X** to the power of **Y**, and stores the result in **Z**.

This instruction always passes power.

Parameter Data Types

The following data types can be used with the EXPT instruction.

If **X** is a:	and **Y** is a:	then **Z** is a:
DINT or LREAL	DINT or LREAL	LREAL

Move (MOV)

The MOV instruction copies **IN** to **OUT**.

This instruction always passes power.

Parameter Data Types

If **IN** and **OUT** parameters are of different data types, the result is converted to the data type of **OUT**. Some arrays can be moved, but they must be of identical type and size.

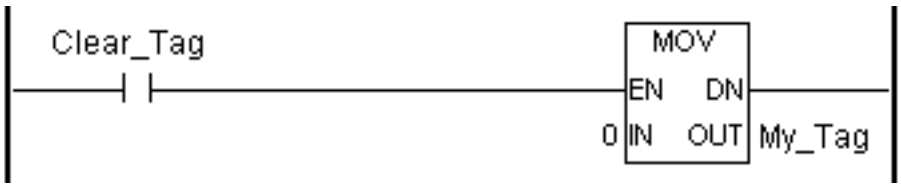
Valid parameter data types for the MOV instruction are as follows:

If **IN** is:	then **OUT** is:
BOOL array-size n	BOOL array-size n
DINT	DINT or LREAL
DINT array size n	DINT array size n or STRING variable
DINT constant	DINT or LREAL
LREAL	DINT or LREAL
LREAL array size n	LREAL array size n
LREAL constant	DINT or LREAL
STRING	STRING variable or DINT array

Note: The MOV instruction doesn't support simple BOOL variables, STRING arrays, and roots of structure variables.

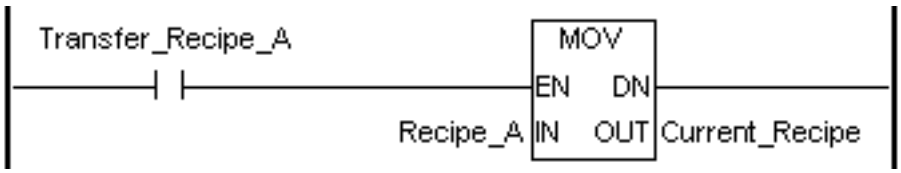
Example 1

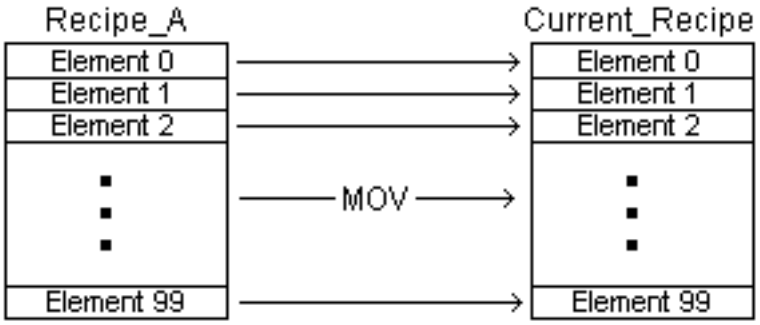
A variable can be cleared by using the MOV instruction to move a 0 into it. When 'Clear_Tag' is set to ON(1), a 0 is moved into 'My_Tag'.



Example 2

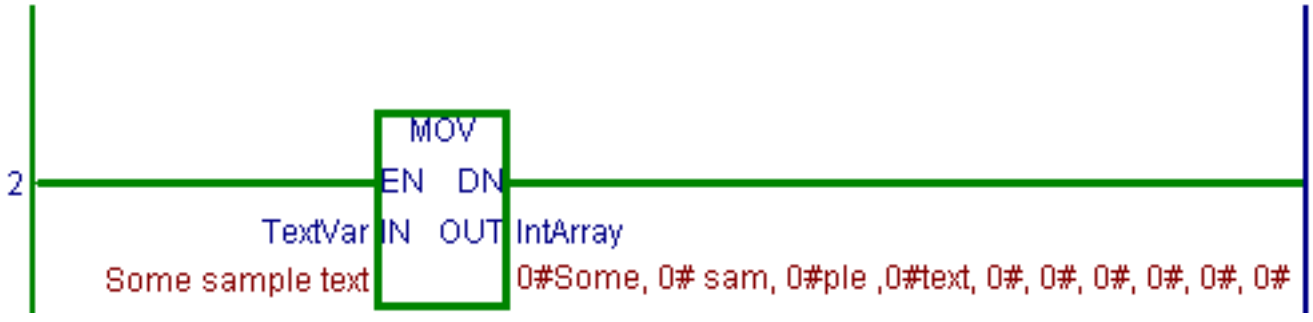
A block move can be performed with the MOV instruction by specifying two arrays of the same data type and size. In the following example, two 100 element arrays are used to store recipes. The recipes can be transferred with a single MOV instruction. When 'Transfer_Recipe_A' is set to ON(1), the content of array 'Recipe_A' is copied into 'Current_Recipe'.





Example 3

You can use a MOV instruction to initialize a DINT array with a text string. Each element of the array can hold up to four ASCII characters. This is useful when working with some Control I/O drivers that require text to be passed in a DINT array. The size of the array doesn't have to match the number of characters you are moving into it; unused elements of the array are set to zero. Ensure that the array can hold all the text moved into it. The following example shows how the content of a text variable is copied to a DINT array.



Warning: #Overflow is set to ON(1) if the operation involves a conversion from LREAL to another data type and the value is too large to move. In this instance the result is undefined.

See also the assign (:=) instruction in ST and the MOV instruction in FBD.

Block Move (BMOV)

The BMOV instruction copies some of the elements of one DINT array into some of the elements of another DINT array. Specifically, **LEN** elements of array **IN**, starting at index **START**, are copied into array **OUT**, starting at index **DEST**.

The BMOV instruction always passes power.

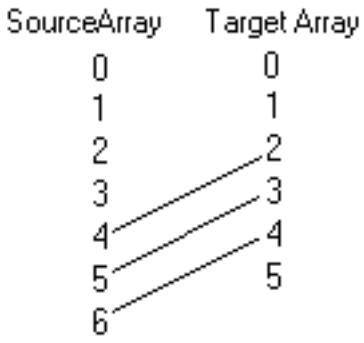
Parameter Data Types

DINT arrays of different sizes can be used.

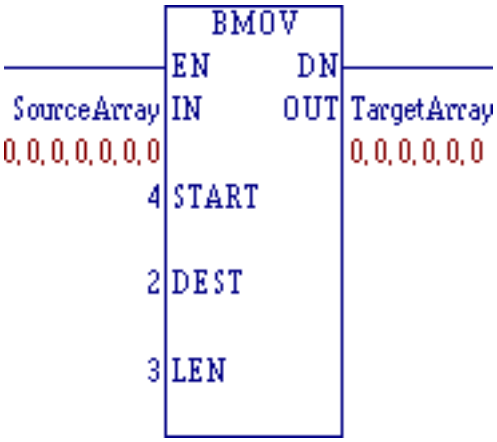
IN, OUT can be:	**START, DEST, LEN** can be:
DINT Array	DINT
	DINT Constant

Example

To copy three elements of SourceArray[7] into TargetArray[6] as shown below,



configure the BMOV instruction like this:



Note: If the instruction tries to access an element of an array that does not exist, #Status will indicate a major fault and #FaultCode will indicate an array bounds error.

See also the MOV instruction in FBD.

Fill Move (FMOV)

The FMOV instruction fills **LEN** elements of DINT array **OUT** with value **VALUE**, starting at index **INDEX**.

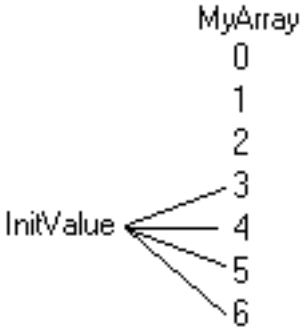
The FMOV instruction always passes power.

Parameter Data Types

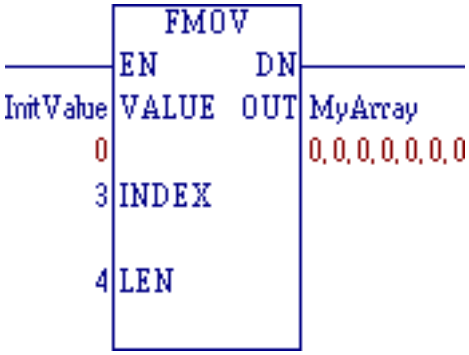
VALUE, INDEX, LEN can be:	**OUT** can be:
DINT	DINT array
DINT constant	

Example

To copy an initial value, 'InitValue', into some of the elements of the seven element array 'MyArray', as shown below,



configure the FMOV instruction like this:



See also the MOV instruction in FBD.