



Allen-Bradley

Logix5000™ Controllers General Instructions

**1756 ControlLogix®,
1769 CompactLogix™,
1789 SoftLogix™,
1794 FlexLogix™, PowerFlex
700S with DriveLogix**

Reference Manual

**Rockwell
Automation**

Important User Information

Because of the variety of uses for the products described in this publication, those responsible for the application and use of these products must satisfy themselves that all necessary steps have been taken to assure that each application and use meets all performance and safety requirements, including any applicable laws, regulations, codes and standards. In no event will Rockwell Automation be responsible or liable for indirect or consequential damage resulting from the use or application of these products.

Any illustrations, charts, sample programs, and layout examples shown in this publication are intended solely for purposes of example. Since there are many variables and requirements associated with any particular installation, Rockwell Automation does not assume responsibility or liability (to include intellectual property liability) for actual use based upon the examples shown in this publication.

Allen-Bradley publication SGI-1.1, *Safety Guidelines for the Application, Installation and Maintenance of Solid-State Control* (available from your local Rockwell Automation office), describes some important differences between solid-state equipment and electromechanical devices that should be taken into consideration when applying products such as those described in this publication.

Reproduction of the contents of this copyrighted publication, in whole or part, without written permission of Rockwell Automation, is prohibited.

Throughout this publication, notes may be used to make you aware of safety considerations. The following annotations and their accompanying statements help you to identify a potential hazard, avoid a potential hazard, and recognize the consequences of a potential hazard:

ATTENTION



Identifies information about practices or circumstances that can lead to personal injury or death, property damage, or economic loss.

IMPORTANT

Identifies information that is critical for successful application and understanding of the product.

Allen-Bradley, ControlLogix, DH+, Logix5000, PLC-2, PLC-3, PLC-5 RSLinx, RSLogix 5000, RSNetWorx, and SLC are trademarks of Rockwell Automation

ControlNet is a trademark of ControlNet International, Ltd.

DeviceNet is a trademark of the Open DeviceNet Vendor Association.

Introduction

This release of this document contains new and updated information. To find new and updated information, look for change bars, as shown next to this paragraph.

Updated Information

This document contains the following changes:

Change:	See chapter/appendix:
Instruction locator table now includes the name of each instruction	Instruction Locator
Table that shows the change in PLC/SLC error codes from R9.x and earlier to R10.x and later.	3
Updates to how to choose a cache option for a message. Changes include the number of connections that you can cache.	3
Additional product codes for the ProductCode attribute of the CONTROLLERDEVICE object	3
New attributes for the TASK object: <ul style="list-style-type: none">• DisableUpdateOutputs• EnableTimeOut• InhibitTask• OverlapCount• Status	3
The Set System Value (SSV) instruction now lets you programmatically change the priority and rate (period) of a task.	3
For an event task, use the Rate attribute of the TASK object to get or set the timeout value for an event task.	3
Immediate Output (IOT) instruction. Use an IOT instruction to immediately update output data or trigger an event task in another controller.	3
Clarification of how the SFC Reset (SFR) instruction effects stored actions.	10
Trigger Event Task (EVENT) instruction. Use an EVENT instruction to trigger the execution of an event task.	10
Information on how to choose function block elements, including IREFs, OREFs, ICONs, and OCONs	B
Updated information on using function block instructions in the periodic timing mode.	B
Clarification of the difference between the structured text CASE construct and the C/C++ switch statement.	C

Notes:

Where to Find an Instruction

Use this locator to find the reference details about Logix instructions (the grayed-out instructions are available in other manuals). This locator also lists which programming languages are available for the instructions.

If the locator lists:	The instruction is documented in:
a page number	this manual
process control	<i>Logix5000 Controllers Process Control and Drives Instruction Set Reference Manual</i> , publication 1756-RM006
motion	<i>Logix5000 Controllers Motion Instruction Set Reference Manual</i> , publication 1756-RM007

Instruction:	Location:	Languages:
ABL ASCII Test For Buffer Line	16-5	relay ladder structured text
ABS Absolute Value	5-29	relay ladder structured text function block
ACB ASCII Chars in Buffer	16-8	relay ladder structured text
ACL ASCII Clear Buffer	16-10	relay ladder structured text
ACOS Arc Cosine	13-14	structured text
ACS Arc Cosine	13-14	relay ladder function block
ADD Add	5-6	relay ladder structured text function block
AFI Always False Instruction	10-23	relay ladder
AHL ASCII Handshake Lines	16-12	relay ladder structured text
ALM Alarm	process control	structured text function block
AND Bitwise AND	6-23	relay ladder structured text function block
ARD ASCII Read	16-16	relay ladder structured text
ARL ASCII Read Line	16-19	relay ladder structured text
ASIN Arc Sine	13-11	structured text

Instruction:	Location:	Languages:
ASN Arc Sine	13-11	relay ladder function block
ATAN Arc Tangent	13-17	structured text
ATN Arc Tangent	13-17	relay ladder function block
AVE File Average	7-38	relay ladder
AWA ASCII Write Append	16-23	relay ladder structured text
AWT ASCII Write	16-28	relay ladder structured text
BAND Boolean AND	6-35	structured text function block
BNOT Boolean NOT	6-44	structured text function block
BOR Boolean OR	6-38	structured text function block
BRK Break	11-5	relay ladder
BSL Bit Shift Left	8-2	relay ladder
BSR Bit Shift Right	8-5	relay ladder
BTD Bit Field Distribute	6-11	relay ladder
BTDT Bit Field Distribute with Target	6-14	structured text function block
BTR Message	3-2	relay ladder structured text

Instruction:	Location:	Languages:
BTW Message	3-2	relay ladder structured text
BXOR Boolean Exclusive OR	6-41	structured text function block
CLR Clear	6-17	relay ladder structured text
CMP Compare	4-2	relay ladder
CONCAT String Concatenate	17-3	relay ladder structured text
COP Copy File	7-28	relay ladder structured text
COS Cosine	13-5	relay ladder structured text function block
CPS Synchronous Copy File	7-28	relay ladder structured text
CPT Compute	5-2	relay ladder
CTD Count Down	2-28	relay ladder
CTU Count Up	2-24	relay ladder
CTUD Count Up/Down	2-32	structured text function block
D2SD Discrete 2-State Device	process control	structured text function block
D3SD Discrete 3-State Device	process control	structured text function block
DDT Diagnostic Detect	12-10	relay ladder
DEDT Deadtime	process control	structured text function block
DEG Degrees	15-2	relay ladder structured text function block
DELETE String Delete	17-5	relay ladder structured text
DERV Derivative	process control	structured text function block
DFF D Flip-Flop	process control	structured text function block

Instruction:	Location:	Languages:
DIV Divide	5-15	relay ladder structured text function block
DTOS DINT to String	18-8	relay ladder structured text
DTR Data Transitional	12-18	relay ladder
EOT End of Transition	10-25	relay ladder structured text
EQU Equal to	4-7	relay ladder structured text function block
ESEL Enhanced Select	process control	structured text function block
EVENT Trigger Event Task	10-31	relay ladder structured text
FAL File Arithmetic and Logic	7-7	relay ladder
FBC File Bit Comparison	12-2	relay ladder
FFL FIFO Load	8-8	relay ladder
FFU FIFO Unload	8-14	relay ladder
FGEN Function Generator	process control	structured text function block
FIND Find String	17-7	relay ladder structured text
FLL File Fill	7-34	relay ladder
FOR For	11-2	relay ladder
FRD Convert to Integer	15-9	relay ladder function block
FSC File Search and Compare	7-19	relay ladder
GEQ Greater than or Equal to	4-11	relay ladder structured text function block
GRT Greater Than	4-15	relay ladder structured text function block
GSV Get System Value	3-34	relay ladder structured text

Instruction:	Location:	Languages:
HLL High/Low Limit	process control	structured text function block
HPF High Pass Filter	process control	structured text function block
ICON Input Wire Connector	B-1	function block
INSERT Insert String	17-9	relay ladder structured text
INTG Integrator	process control	structured text function block
IOT Immediate Output	3-57	relay ladder structured text
IREF Input Reference	B-1	function block
JKFF JK Flip-Flop	process control	structured text function block
JMP Jump to Label	10-2	relay ladder
JSR Jump to Subroutine	10-4	relay ladder structured text function block
JXR Jump to External Routine	10-14	relay ladder
LBL Label	10-2	relay ladder
LDL2 Second-Order Lead Lag	process control	structured text function block
LDLG Lead-Lag	process control	structured text function block
LEQ Less Than or Equal to	4-19	relay ladder structured text function block
LES Less Than	4-23	relay ladder structured text function block
LFL LIFO Load	8-20	relay ladder
LFU LIFO Unload	8-26	relay ladder
LIM Limit	4-27	relay ladder function block
LN Natural Log	14-2	relay ladder structured text function block

Instruction:	Location:	Languages:
LOG Log Base 10	14-4	relay ladder structured text function block
LOWER Lower Case	18-14	relay ladder structured text
LPF Low Pass Filter	process control	structured text function block
MAAT Motion Apply Axis Tuning	motion	relay ladder structured text
MAFR Motion Axis Fault Reset	motion	relay ladder structured text
MAG Motion Axis Gear	motion	relay ladder structured text
MAH Motion Axis Home	motion	relay ladder structured text
MAHD Motion Apply Hookup Diagnostics	motion	relay ladder structured text
MAJ Motion Axis Jog	motion	relay ladder structured text
MAM Motion Axis Move	motion	relay ladder structured text
MAOC Motion Arm Output Cam	motion	relay ladder structured text
MAPC Motion Axis Position Cam	motion	relay ladder structured text
MAR Motion Arm Registration	motion	relay ladder structured text
MAS Motion Axis Stop	motion	relay ladder structured text
MASD Motion Axis Shutdown	motion	relay ladder structured text
MASR Motion Axis Shutdown Reset	motion	relay ladder structured text
MATC Motion Axis Time Cam	motion	relay ladder structured text
MAVE Moving Average	process control	structured text function block
MAW Motion Arm Watch	motion	relay ladder structured text
MAXC Maximum Capture	process control	structured text function block

Instruction:	Location:	Languages:
MCCD Motion Coordinated Change Dynamics	motion	relay ladder structured text
MCCM Motion Coordinated Circular Move	motion	relay ladder structured text
MCCP Motion Calculate Cam Profile	motion	relay ladder structured text
MCD Motion Change Dynamics	motion	relay ladder structured text
MCLM Motion Coordinated Linear Move	motion	relay ladder structured text
MCR Master Control Reset	10-19	relay ladder
MCS Motion Coordinated Stop	motion	relay ladder structured text
MCSD Motion Coordinated Shutdown	motion	relay ladder structured text
MCSR Motion Coordinated Shutdown Reset	motion	relay ladder structured text
MDF Motion Direct Drive Off	motion	relay ladder structured text
MDO Motion Direct Drive On	motion	relay ladder structured text
MDOC Motion Disarm Output Cam	motion	relay ladder structured text
MDR Motion Disarm Registration	motion	relay ladder structured text
MDW Motion Disarm Watch	motion	relay ladder structured text
MEQ Mask Equal to	4-33	relay ladder structured text function block
MGS Motion Group Stop	motion	relay ladder structured text
MGSD Motion Group Shutdown	motion	relay ladder structured text
MGSP Motion Group Strobe Position	motion	relay ladder structured text

Instruction:	Location:	Languages:
MGSR Motion Group Shutdown Reset	motion	relay ladder structured text
MID Middle String	17-11	relay ladder structured text
MINC Minimum Capture	process control	structured text function block
MOD Modulo	5-19	relay ladder structured text function block
MOV Move	6-3	relay ladder
MRAT Motion Run Axis Tuning	motion	relay ladder structured text
MRHD Motion Run Hookup Diagnostics	motion	relay ladder structured text
MRP Motion Redefine Position	motion	relay ladder structured text
MSF Motion Servo Off	motion	relay ladder structured text
MSG Message	3-2	relay ladder structured text
MSO Motion Servo On	motion	relay ladder structured text
MSTD Moving Standard Deviation	process control	structured text function block
MUL Multiply	5-12	relay ladder structured text function block
MUX Multiplexer	process control	function block
MVM Masked Move	6-5	relay ladder
MVMT Masked Move with Target	6-8	structured text function block
NEG Negate	5-26	relay ladder structured text function block
NEQ Not Equal to	4-38	relay ladder structured text function block
NOP No Operation	10-24	relay ladder

Instruction:	Location:	Languages:
NOT Bitwise NOT	6-32	relay ladder structured text function block
NTCH Notch Filter	process control	structured text function block
OCN Output Wire Connector	B-1	function block
ONS One Shot	1-12	relay ladder
OR Bitwise OR	6-26	relay ladder structured text function block
OREF Output Reference	B-1	function block
OSF One Shot Falling	1-17	relay ladder
OSFI One Shot Falling with Input	1-22	structured text function block
OSR One Shot Rising	1-15	relay ladder
OSRI One Shot Rising with Input	1-19	structured text function block
OTE Output Energize	1-6	relay ladder
OTL Output Latch	1-8	relay ladder
OTU Output Unlatch	1-10	relay ladder
PI Proportional + Integral	process control	structured text function block
PID Proportional Integral Derivative	12-21	relay ladder structured text
PIDE Enhanced PID	process control	structured text function block
PMUL Pulse Multiplier	process control	structured text function block
POSP Position Proportional	process control	structured text function block
RAD Radians	15-4	relay ladder structured text function block
RES Reset	2-36	relay ladder

Instruction:	Location:	Languages:
RESD Reset Dominant	process control	structured text function block
RET Return	10-4 and 11-6	relay ladder structured text function block
RLIM Rate Limiter	process control	structured text function block
RMPS Ramp/Soak	process control	structured text function block
RTO Retentive Timer On	2-10	relay ladder
RTOR Retentive Timer On with Reset	2-20	structured text function block
RTOS REAL to String	18-10	relay ladder structured text
SBR Subroutine	10-4	relay ladder structured text function block
SCL Scale	process control	structured text function block
SCRV S-Curve	process control	structured text function block
SEL Select	process control	function block
SETD Set Dominant	process control	structured text function block
SFP SFC Pause	10-27	relay ladder structured text
SFR SFC Reset	10-29	relay ladder structured text
SIN Sine	13-2	relay ladder structured text function block
SIZE Size In Elements	7-53	relay ladder structured text
SNEG Selected Negate	process control	structured text function block
SOC Second-Order Controller	process control	structured text function block
SQI Sequencer Input	9-2	relay ladder
SQL Sequencer Load	9-10	relay ladder

Instruction:	Location:	Languages:
SQO Sequencer Output	9-6	relay ladder
SQR Square Root	5-23	relay ladder function block
SQRT Square Root	5-23	structured text
SRT File Sort	7-43	relay ladder structured text
SRTP Split Range Time Proportional	process control	structured text function block
SSUM Selected Summer	process control	structured text function block
SSV Set System Value	3-34	relay ladder structured text
STD File Standard Deviation	7-48	relay ladder
STOD String To DINT	18-4	relay ladder structured text
STOR String To REAL	18-6	relay ladder structured text
SUB Subtract	5-9	relay ladder structured text function block
SWPB Swap Byte	6-19	relay ladder structured text
TAN Tangent	13-8	relay ladder structured text function block
TND Temporary End	10-17	relay ladder
TOD Convert to BCD	15-6	relay ladder function block
TOF Timer Off Delay	2-6	relay ladder
TOFR Timer Off Delay with Reset	2-17	structured text function block
TON Timer On Delay	2-2	relay ladder
TONR Timer On Delay with Reset	2-14	structured text function block
TOT Totalizer	process control	structured text function block

Instruction:	Location:	Languages:
TRN Truncate	15-11	relay ladder function block
TRUNC Truncate	15-11	structured text
UID User Interrupt Disable	10-21	relay ladder structured text
UIE User Interrupt Enable	10-21	relay ladder structured text
UPDN Up/Down Accumulator	process control	structured text function block
UPPER Upper Case	18-12	relay ladder structured text
XIC Examine If Closed	1-2	relay ladder
XIO Examine If Open	1-4	relay ladder
XOR Bitwise Exclusive OR	6-29	relay ladder structured text function block
XPY X to the Power of Y	14-6	relay ladder structured text function block

Introduction

This manual is one of several Logix5000-based instruction documents.

Task/Goal:	Documents:
Programming the controller for sequential applications	<i>Logix5000 Controllers General Instructions Reference Manual</i> , publication 1756-RM003
<div>You are here</div> <div></div>	
Programming the controller for process or drives applications	<i>Logix5000 Controllers Process Control and Drives Instructions Reference Manual</i> , publication 1756-RM006
Programming the controller for motion applications	<i>Logix5000 Controllers Motion Instruction Set Reference Manual</i> , publication 1756-RM007
Importing a text file or tags into a project	<i>Logix5000 Controllers Import/Export Reference Manual</i> , publication 1756-RM084
Exporting a project or tags to a text file	
Converting a PLC-5 or SLC 500 application to a Logix5000 application	<i>Logix5550 Controller Converting PLC-5 or SLC 500 Logic to Logix5550 Logic Reference Manual</i> , publication 1756-6.8.5




Who Should Use This Manual

This document provides a programmer with details about each available instruction for a Logix-based controller. You should already be familiar with how the Logix-based controller stores and processes data.




Novice programmers should read all the details about an instruction before using the instruction. Experienced programmers can refer to the instruction information to verify details.

Purpose of This Manual

This manual provides a description of each instruction in this format.

This section:	Provides this type of information:
Instruction name	identifies the instruction defines whether the instruction is an input or an output instruction
Operands	lists all the operands of the instruction  if available in relay ladder, describes the operands  if available in structured text, describes the operands  if available in function block, describes the operands The pins shown on a default function block are only the default pins. The operands table lists all the possible pins for a function block.
Instruction structure	lists control status bits and values, if any, of the instruction
Description	describes the instruction's use defines any differences when the instruction is enabled and disabled, if appropriate
Arithmetic status flags	defines whether or not the instruction affects arithmetic status flags see appendix Common Attributes
Fault conditions	defines whether or not the instruction generates minor or major faults if so, defines the fault type and code
Execution	defines the specifics of how the instruction operates
Example	provides at least one programming example in each available programming language includes a description explaining each example

The following icons help identify language specific information:

This icon:	Indicates this programming language:
	relay ladder
	structured text
	function block

Common Information for All Instructions

The Logix5000 instruction set has some common attributes:

For this information:	See this appendix:
common attributes	appendix Common Attributes defines: <ul style="list-style-type: none">• arithmetic status flags• data types• keywords
function block attributes	appendix Function Block Attributes defines: <ul style="list-style-type: none">• program and operator control• timing modes

Conventions and Related Terms

Set and clear

This manual uses set and clear to define the status of bits (booleans) and values (non-booleans):

This term:	Means:
set	the bit is set to 1 (ON) a value is set to any non-zero number
clear	the bit is cleared to 0 (OFF) all the bits in a value are cleared to 0

If an operand or parameter support more than one data type, the **bold** data types indicate optimal data types. An instruction executes faster and requires less memory if all the operands of the instruction use the same optimal data type, typically DINT or REAL.

Relay ladder rung condition

The controller evaluates ladder instructions based on the rung condition preceding the instruction (rung-condition-in). Based on the rung-condition-in and the instruction, the controller sets the rung condition following the instruction (rung-condition-out), which in turn, affects any subsequent instruction.



If the rung-in condition to an input instruction is true, the controller evaluates the instruction and sets the rung-out condition based on the results of the instruction. If the instruction evaluates to true, the rung-out condition is true; if the instruction evaluates to false, the rung-out condition is false.

The controller also prescans instructions. Prescan is a special scan of all routines in the controller. The controller scans all main routines and subroutines during prescan, but ignores jumps that could skip the execution of instructions. The controller executes all FOR loops and subroutine calls. If a subroutine is called more than once, it is executed each time it is called. The controller uses prescan of relay ladder instructions to reset non-retentive I/O and internal values.

During prescan, input values are not current and outputs are not written. The following conditions generate prescan:

- Toggle from Program to Run mode
- Automatically enter Run mode from a power-up condition.

Prescan does not occur for a program when:

- The program becomes scheduled while the controller is running.
- The program is unscheduled when the controller enters Run mode.

Function block states

IMPORTANT

When programming in function block, restrict the range of engineering units to $\pm 10^{\pm 15}$ because internal floating point calculations are done using single precision floating point. Engineering units outside of this range may result in a loss of accuracy if results approach the limitations of single precision floating point ($\pm 10^{\pm 38}$).

The controller evaluates function block instructions based on the state of different conditions.

Possible Condition:	Description:
prescan	Prescan for function block routines is the same as for relay ladder routines. The only difference is that the EnableIn parameter for each function block instruction is cleared during prescan.
instruction first scan	Instruction first scan refers to the first time an instruction is executed after prescan. The controller uses instruction first scan to read current inputs and determine the appropriate state to be in.
instruction first run	Instruction first run refers to the first time the instruction executes with a new instance of a data structure. The controller uses instruction first run to generate coefficients and other data stores that do not change for a function block after initial download.

Every function block instruction also includes EnableIn and EnableOut parameters:

- function block instructions execute normally when EnableIn is set.
- when EnableIn is cleared, the function block instruction either executes prescan logic, postscan logic, or just skips normal algorithm execution.
- EnableOut mirrors EnableIn, however, if function block execution detects an overflow condition EnableOut is also cleared.
- function block execution resumes where it left off when EnableIn toggles from cleared to set. However there are some function block instructions that specify special functionality, such as re-initialization, when EnableIn toggles from cleared to set. For function block instructions with time base parameters, whenever the timing mode is Oversample, the instruction always resumes where it left off when EnableIn toggles from cleared to set.

If the EnableIn parameter is not wired, the instruction always executes as normal and EnableIn remains set. If you clear EnableIn, it changes to set the next time the instruction executes.

Notes:

Bit Instructions (XIC, XIO, OTE, OTL, OTU, ONS, OSR, OSF, OSRI, OSFI)

Chapter 1

Introduction	1-1
Examine If Closed (XIC)	1-2
Examine If Open (XIO)	1-4
Output Energize (OTE)	1-6
Output Latch (OTL)	1-8
Output Unlatch (OTU)	1-10
One Shot (ONS)	1-12
One Shot Rising (OSR)	1-15
One Shot Falling (OSF)	1-17
One Shot Rising with Input (OSRI)	1-19
One Shot Falling with Input (OSFI)	1-22

Timer and Counter Instructions (TON, TOF, RTO, TONR, TOFR, RTOR, CTU, CTD, CTUD, RES)

Chapter 2

Introduction	2-1
Timer On Delay (TON)	2-2
Timer Off Delay (TOF)	2-6
Retentive Timer On (RTO)	2-10
Timer On Delay with Reset (TONR)	2-14
Timer Off Delay with Reset (TOFR)	2-17
Retentive Timer On with Reset (RTOR)	2-20
Count Up (CTU)	2-24
Count Down (CTD)	2-28
Count Up/Down (CTUD)	2-32
Reset (RES)	2-36

Input/Output Instructions (MSG, GSV, SSV, IOT)

Chapter 3

Introduction	3-1
Message (MSG)	3-2
MSG Error Codes	3-8
Error Codes	3-8
Extended Error Codes	3-10
PLC and SLC Error Codes (.ERR)	3-12
Block-Transfer Error Codes	3-14
Specify the Configuration Details	3-15
Specifying CIP Data Table Read and Write messages	3-16
Reconfigure an I/O module	3-17
Specify CIP Generic messages	3-18
Specifying PLC-5 messages	3-19
Specifying SLC messages	3-20
Specify block-transfer messages	3-21
Specifying PLC-3 messages	3-22
Specifying PLC-2 messages	3-23
MSG Configuration Examples	3-24
Specify the Communication Details	3-25

Specify a path	3-25
Specify a communication method or module address: . .	3-30
Choose a cache option:	3-31
Guidelines	3-33
Get System Value (GSV) and Set System Value (SSV)	3-34
GSV/SSV Objects.	3-36
Accessing the CONTROLLER object	3-37
Accessing the CONTROLLERDEVICE object	3-37
Accessing the CST object	3-39
Accessing the DF1 object	3-40
Accessing the FAULTLOG object.	3-43
Accessing the MESSAGE object.	3-44
Accessing the MODULE object	3-46
Accessing the MOTIONGROUP object	3-47
Accessing the PROGRAM object	3-48
Accessing the ROUTINE object.	3-49
Accessing the SERIALPORT object	3-49
Accessing the TASK object	3-51
Accessing the WALLCLOCKTIME object	3-53
GSV/SSV Programming Example	3-54
Getting fault information	3-54
Setting enable and disable flags	3-56
Immediate Output (IOT)	3-57

Chapter 4

Compare Instructions

**(CMP, EQU, GEQ, GRT, LEQ, LES,
LIM, MEQ, NEQ)**

Introduction	4-1
Compare (CMP)	4-2
CMP expressions	4-4
Valid operators	4-4
Formatting expressions	4-5
Determining the order of operation	4-5
Using strings in an expression	4-6
Equal to (EQU).	4-7
Greater than or Equal to (GEQ).	4-11
Greater Than (GRT)	4-15
Less Than or Equal to (LEQ)	4-19
Less Than (LES)	4-23
Limit (LIM)	4-27
Mask Equal to (MEQ)	4-33
Entering an immediate mask value	4-34
Not Equal to (NEQ).	4-38

Compute/Math Instructions (CPT, ADD, SUB, MUL, DIV, MOD, SQR, SQRT, NEG, ABS)

Chapter 5

Introduction	5-1
Compute (CPT).	5-2
Valid operators	5-4
Formatting expressions	5-4
Determining the order of operation	5-5
Add (ADD).	5-6
Subtract (SUB)	5-9
Multiply (MUL)	5-12
Divide (DIV).	5-15
Modulo (MOD).	5-19
Square Root (SQR)	5-23
Negate (NEG)	5-26
Absolute Value (ABS)	5-29

Move/Logical Instructions (MOV, MVM, BT, MVMT, BTDT, CLR, SWPB, AND, OR, XOR, NOT, BAND, BOR, BXOR, BNOT)

Chapter 6

Introduction	6-1
Move (MOV).	6-3
Masked Move (MVM)	6-5
Entering an immediate mask value	6-6
Masked Move with Target (MVMT)	6-8
Bit Field Distribute (BT)	6-11
Bit Field Distribute with Target (BTDT)	6-14
Clear (CLR).	6-17
Swap Byte (SWPB)	6-19
Bitwise AND (AND)	6-23
Bitwise OR (OR)	6-26
Bitwise Exclusive OR (XOR)	6-29
Bitwise NOT (NOT)	6-32
Boolean AND (BAND)	6-35
Boolean OR (BOR)	6-38
Boolean Exclusive OR (BXOR)	6-41
Boolean NOT (BNOT).	6-44

Array (File)/Misc. Instructions (FAL, FSC, COP, CPS, FLL, AVE, SRT, STD, SIZE)

Chapter 7

Introduction	7-1
Selecting Mode of Operation	7-2
All mode	7-2
Numerical mode	7-3
Incremental mode	7-5
File Arithmetic and Logic (FAL)	7-7
FAL expressions.	7-16
Valid operators	7-17
Formatting expressions	7-17
Determining the order of operation	7-18

File Search and Compare (FSC)	7-19
FSC expressions	7-24
Valid operators	7-25
Formatting expressions	7-25
Determining the order of operation	7-26
Using strings in an expression	7-27
Copy File (COP) Synchronous Copy File (CPS).	7-28
File Fill (FLL)	7-34
File Average (AVE)	7-38
File Sort (SRT).	7-43
File Standard Deviation (STD)	7-48
Size In Elements (SIZE)	7-53

Array (File)/Shift Instructions (BSL, BSR, FFL, FFU, LFL, LFU)

Chapter 8

Introduction	8-1
Bit Shift Left (BSL).	8-2
Bit Shift Right (BSR)	8-5
FIFO Load (FFL)	8-8
FIFO Unload (FFU)	8-14
LIFO Load (LFL)	8-20
LIFO Unload (LFU)	8-26

Sequencer Instructions (SQI, SQO, SQL)

Chapter 9

Introduction	9-1
Sequencer Input (SQI).	9-2
Entering an immediate mask value	9-3
Using SQI without SQO	9-5
Sequencer Output (SQO)	9-6
Entering an immediate mask value	9-7
Using SQI with SQO	9-9
Resetting the position of SQO	9-9
Sequencer Load (SQL).	9-10

Program Control Instructions (JMP, LBL, JSR, RET, SBR, JXR, TND, MCR, UID, UIE, AFI, NOP, EOT, SFP, SFR, EVENT)

Chapter 10

Introduction	10-1
Jump to Label (JMP)	
Label (LBL).	10-2
Jump to Subroutine (JSR)	
Subroutine (SBR)	
Return (RET).	10-4
Jump to External Routine (JXR)	10-14
Temporary End (TND)	10-17
Master Control Reset (MCR).	10-19
User Interrupt Disable (UID) User Interrupt Enable (UIE) .	10-21
Always False Instruction (AFI)	10-23

	No Operation (NOP)	10-24
	End of Transition (EOT)	10-25
	SFC Pause (SFP)	10-27
	SFC Reset (SFR)	10-29
	Trigger Event Task (EVENT)	10-31
	Programmatically Determine if an EVENT Instruction Triggered a Task	10-31
For/Break Instructions (FOR, FOR...DO, BRK, EXIT, RET)	Chapter 11	
	Introduction	11-1
	For (FOR)	11-2
	Break (BRK)	11-5
	Return (RET)	11-6
Special Instructions (FBC, DDT, DTR, PID)	Chapter 12	
	Introduction	12-1
	File Bit Comparison (FBC)	12-2
	Selecting the search mode	12-4
	Diagnostic Detect (DDT)	12-10
	Selecting the search mode	12-12
	Data Transitional (DTR)	12-18
	Entering an immediate mask value	12-19
	Proportional Integral Derivative (PID)	12-21
	Configuring a PID Instruction	12-26
	Specifying tuning	12-27
	Specifying configuration	12-27
	Specifying alarms	12-28
	Specifying scaling	12-29
	Using PID Instructions	12-29
	Anti-reset windup and bumpless transfer from manual to auto 12-31	
	PID instruction timing	12-32
	Bumpless restart	12-36
	Derivative smoothing	12-37
	Setting the deadband	12-38
	Using output limiting	12-38
	Feedforward or output biasing	12-39
	Cascading loops	12-39
	Controlling a ratio	12-40
	PID Theory	12-41
	PID process	12-41
	PID process with master/slave loops	12-41

Trigonometric Instructions (SIN, COS, TAN, ASN, ASIN, ACS, ACOS, ATN, ATAN)	Chapter 13 Introduction 13-1 Sine (SIN) 13-2 Cosine (COS) 13-5 Tangent (TAN) 13-8 Arc Sine (ASN) 13-11 Arc Cosine (ACS) 13-14 Arc Tangent (ATN) 13-17
Advanced Math Instructions (LN, LOG, XPY)	Chapter 14 Introduction 14-1 Natural Log (LN) 14-2 Log Base 10 (LOG) 14-4 X to the Power of Y (XPY) 14-6
Math Conversion Instructions (DEG, RAD, TOD, FRD, TRN, TRUNC)	Chapter 15 Introduction 15-1 Degrees (DEG) 15-2 Radians (RAD) 15-4 Convert to BCD (TOD) 15-6 Convert to Integer (FRD) 15-9 Truncate (TRN) 15-11
ASCII Serial Port Instructions (ABL, ACB, ACL, AHL, ARD, ARL, AWA, AWT)	Chapter 16 Introduction 16-1 Instruction Execution 16-2 ASCII Error Codes 16-4 String Data Types 16-4 ASCII Test For Buffer Line (ABL) 16-5 ASCII Chars in Buffer (ACB) 16-8 ASCII Clear Buffer (ACL) 16-10 ASCII Handshake Lines (AHL) 16-12 ASCII Read (ARD) 16-16 ASCII Read Line (ARL) 16-19 ASCII Write Append (AWA) 16-23 ASCII Write (AWT) 16-28

ASCII String Instructions (CONCAT, DELETE, FIND, INSERT, MID)	Chapter 17 Introduction 17-1 String Data Types. 17-2 String Concatenate (CONCAT). 17-3 String Delete (DELETE). 17-5 Find String (FIND) 17-7 Insert String (INSERT) 17-9 Middle String (MID) 17-11
ASCII Conversion Instructions (STOD, STOR, DTOS, RTOS, UPPER, LOWER)	Chapter 18 Introduction 18-1 String Data Types. 18-3 String To DINT (STOD). 18-4 String To REAL (STOR) 18-6 DINT to String (DTOS) 18-8 REAL to String (RTOS). 18-10 Upper Case (UPPER). 18-12 Lower Case (LOWER) 18-14
Common Attributes	Appendix A Introduction A-1 Immediate Values A-1 Data Conversions A-1 SINT or INT to DINT A-3 Integer to REAL A-5 DINT to SINT or INT A-5 REAL to an integer. A-6
Function Block Attributes	Appendix B Introduction B-1 Choose the Function Block Elements. B-1 Latching Data B-2 Order of Execution B-4 Resolve a Loop B-5 Resolve Data Flow Between Two Blocks B-6 Create a One Scan Delay B-7 Summary. B-7 Function Block Responses to Overflow Conditions. B-8 Timing Modes. B-9 Common instruction parameters for timing modes. B-11 Overview of timing modes. B-13 Program/Operator Control. B-14

Structured Text Programming**Appendix C**

Introduction	C-1
Structured Text Syntax.	C-1
Assignments	C-2
Specify a non-retentive assignment.	C-3
Assign an ASCII character to a string.	C-4
Expressions	C-4
Use arithmetic operators and functions	C-6
Use relational operators	C-7
Use logical operators	C-9
Use bitwise operators.	C-10
Determine the order of execution.	C-10
Instructions.	C-11
Constructs.	C-12
IF...THEN	C-13
CASE...OF.	C-16
FOR...DO.	C-19
WHILE...DO.	C-22
REPEAT...UNTIL.	C-25
Comments	C-28

Bit Instructions

(XIC, XIO, OTE, OTL, OTU, ONS, OSR, OSF, OSRI, OSFI)

Introduction

Use the bit (relay-type) instructions to monitor and control the status of bits.

If you want to:	Use this instruction:	Available in these languages:	See page:
enable outputs when a bit is set	XIC	relay ladder structured text ⁽¹⁾	1-2
enable outputs when a bit is cleared	XIO	relay ladder structured text ⁽¹⁾	1-4
set a bit	OTE	relay ladder structured text ⁽¹⁾	1-6
set a bit (retentive)	OTL	relay ladder structured text ⁽¹⁾	1-8
clear bit (retentive)	OTU	relay ladder structured text ⁽¹⁾	1-10
enable outputs for one scan each time a rung goes true	ONS	relay ladder structured text ⁽¹⁾	1-12
set a bit for one scan each time a rung goes true	OSR	relay ladder	1-15
set a bit for one scan each time the rung goes false	OSF	relay ladder	1-17
set a bit for one scan each time the input bit is set in function block	OSRI	structured text function block	1-19
set a bit for one scan each time the input bit is cleared in function block	OSFI	structured text function block	1-22

⁽¹⁾ There is no equivalent structured text instruction. Use other structured text programming to achieve the same result. See the description for the instruction.

Examine If Closed (XIC)

The XIC instruction examines the data bit to see if it is set.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
data bit	BOOL	tag	bit to be tested



Structured Text

Structured text does not have an XIC instruction, but you can achieve the same results using an IF...THEN construct.

```
IF data_bit THEN
    <statement>;
END_IF;
```

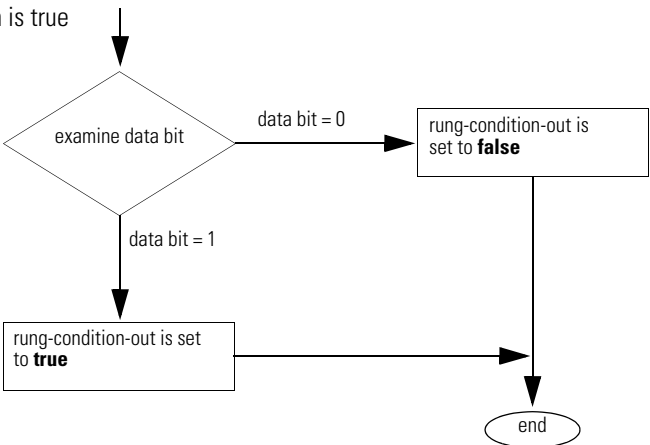
See Appendix C for information on the syntax of constructs within structured text.

Description: The XIC instruction examines the data bit to see if it is set.

Arithmetic Status Flags: not affected

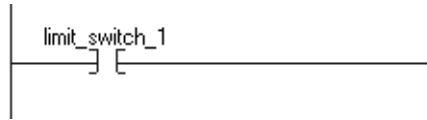
Fault Conditions: none

Execution:

Condition:	Relay Ladder Action:
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	
postscan	The rung-condition-out is set to false.

Example 1: If *limit_switch_1* is set, this enables the next instruction (the rung-condition-out is true).

Relay Ladder



Structured Text

```
IF limit_switch THEN
    <statement>;
END_IF;
```

Example 2: If *S:V* is set (indicates that an overflow has occurred), this enables the next instruction (the rung-condition-out is true).

Relay Ladder



Structured Text

```
IF S:V THEN
    <statement>;
END_IF;
```

Examine If Open (XIO)

The XIO instruction examines the data bit to see if it is cleared.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
data bit	BOOL	tag	bit to be tested



Structured Text

Structured text does not have an XIO instruction, but you can achieve the same results using an IF...THEN construct.

```
IF NOT data_bit THEN
    <statement>;
END_IF;
```

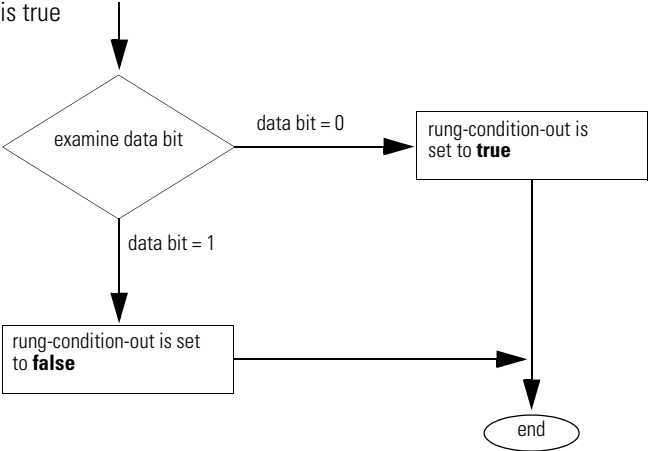
See Appendix C for information on the syntax of constructs within structured text.

Description: The XIO instruction examines the data bit to see if it is cleared.

Arithmetic Status Flags: not affected

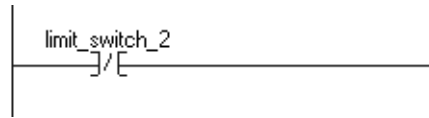
Fault Conditions: none

Execution:

Condition:	Relay Ladder Action:
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	
postscan	The rung-condition-out is set to false.

Example 1: If *limit_switch_2* is cleared, this enables the next instruction (the rung-condition-out is true).

Relay Ladder

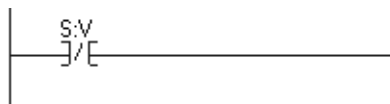


Structured Text

```
IF NOT limit_switch_2 THEN
    <statement>;
END_IF;
```

Example 2: If *S:V* is cleared (indicates that no overflow has occurred), this enables the next instruction (the rung-condition-out is true).

Relay Ladder



Structured Text

```
IF NOT S:V THEN
    <statement>;
END_IF;
```

Output Energize (OTE)

The OTE instruction sets or clears the data bit.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
data bit	BOOL	tag	bit to be set or cleared



Structured Text

Structured text does not have an OTE instruction, but you can achieve the same results using a non-retentive assignment.

```
data_bit [:=] BOOL_expression;
```

See Appendix C for information on the syntax of assignments and expressions within structured text.

Description: When the OTE instruction is enabled, the controller sets the data bit. When the OTE instruction is disabled, the controller clears the data bit.

Arithmetic Status Flags: not affected

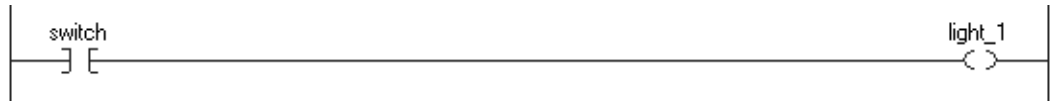
Fault Conditions: none

Execution:

Condition:	Relay Ladder Action:
prescan	The data bit is cleared. The rung-condition-out is set to false.
rung-condition-in is false	The data bit is cleared. The rung-condition-out is set to false.
rung-condition-in is true	The data bit is set. The rung-condition-out is set to true.
postscan	The data bit is cleared. The rung-condition-out is set to false.

Example: When *switch* is set, the OTE instruction sets (turns on) *light_1*. When *switch* is cleared, the OTE instruction clears (turns off) *light_1*.

Relay Ladder



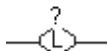
Structured Text

```
light_1 [:=] switch;
```

Output Latch (OTL)

The OTL instruction sets (latches) the data bit.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
data bit	BOOL	tag	bit to be set



Structured Text

Structured text does not have an OTL instruction, but you can achieve the same results using an IF...THEN construct and an assignment.

```
IF BOOL_expression THEN
    data_bit := 1;
END_IF;
```

See Appendix C for information on the syntax of constructs, expressions, and assignments within structured text.

Description: When enabled, the OTL instruction sets the data bit. The data bit remains set until it is cleared, typically by an OTU instruction. When disabled, the OTL instruction does not change the status of the data bit.

Arithmetic Status Flags: not affected

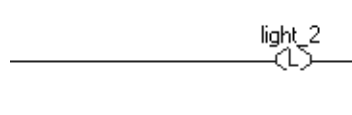
Fault Conditions: none

Execution:

Condition:	Relay Ladder Action:
prescan	The data bit is not modified. The rung-condition-out is set to false.
rung-condition-in is false	The data bit is not modified. The rung-condition-out is set to false.
rung-condition-in is true	The data bit is set. The rung-condition-out is set to true.
postscan	The data bit is not modified. The rung-condition-out is set to false.

Example: When enabled, the OTL instruction sets *light_2*. This bit remains set until it is cleared, typically by an OTU instruction.

Relay Ladder



Structured Text

```
IF BOOL_expression THEN  
    light_2 := 1;  
END_IF;
```

Output Unlatch (OTU)

The OTU instruction clears (unlatches) the data bit.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
data bit	BOOL	tag	bit to be cleared



Structured Text

Structured text does not have an OTU instruction, but you can achieve the same results using an IF...THEN construct and an assignment.

```
IF BOOL_expression THEN
    data_bit := 0;
END_IF;
```

See Appendix C for information on the syntax of constructs, expressions, and assignments within structured text.

Description: When enabled, the OTU instruction clears the data bit. When disabled, the OTU instruction does not change the status of the data bit.

Arithmetic Status Flags: not affected

Fault Conditions: none

Execution:

Condition:	Relay Ladder Action:
prescan	The data bit is not modified. The rung-condition-out is set to false.
rung-condition-in is false	The data bit is not modified. The rung-condition-out is set to false.
rung-condition-in is true	The data bit is cleared. The rung-condition-out is set to true.
postscan	The data bit is not modified. The rung-condition-out is set to false.

Example: When enabled, the OTU instruction clears *light_2*.

Relay Ladder



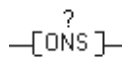
Structured Text

```
IF BOOL_expression THEN  
    light_2 := 0;  
END_IF;
```

One Shot (ONS)

The ONS instruction enables or disables the remainder of the rung, depending on the status of the storage bit.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
storage bit	BOOL	tag	internal storage bit stores the rung-condition-in from the last time the instruction was executed



Structured Text

Structured text does not have an ONS instruction, but you can achieve the same results using an IF...THEN construct.

```
IF BOOL_expression AND NOT storage_bit THEN
    <statement>;
END_IF;
storage_bit := BOOL_expression;
```

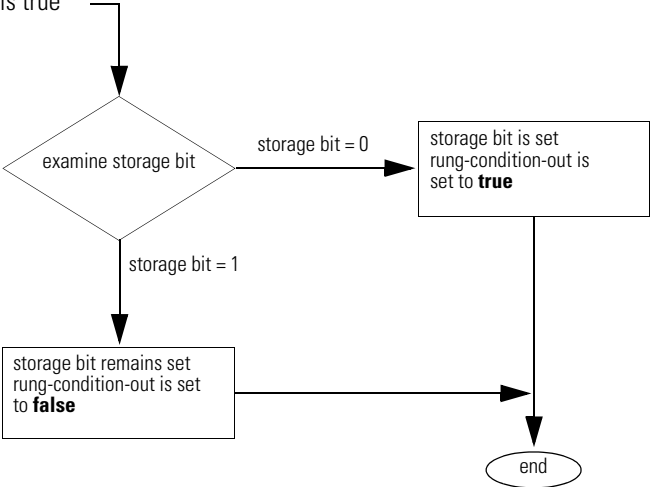
See Appendix C for information on the syntax of constructs, expressions, and expressions within structured text.

Description: When enabled and the storage bit is cleared, the ONS instruction enables the remainder of the rung. When disabled or when the storage bit is set, the ONS instruction disables the remainder of the rung.

Arithmetic Status Flags: not affected

Fault Conditions: none

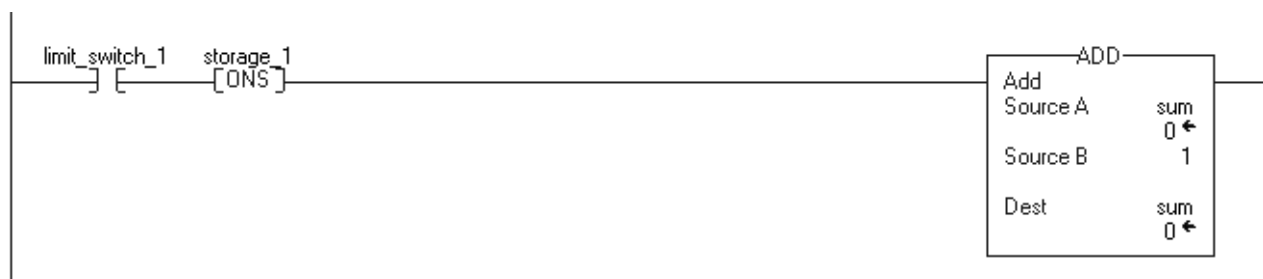
Execution:

Condition:	Relay Ladder Action:
prescan	The storage bit is set to prevent an invalid trigger during the first scan. The rung-condition-out is set to false.
rung-condition-in is false	The storage bit is cleared. The rung-condition-out is set to false.
rung-condition-in is true	 <pre>graph TD; Start([]) --> Decision{examine storage bit}; Decision -- "storage bit = 0" --> Action1["storage bit is set rung-condition-out is set to true"]; Decision -- "storage bit = 1" --> Action2["storage bit remains set rung-condition-out is set to false"]; Action1 --> End([end]); Action2 --> End;</pre>
postscan	The storage bit is cleared. The rung-condition-out is set to false.

Example: You typically precede the ONS instruction with an input instruction because you scan the ONS instruction when it is enabled and when it is disabled for it to operate correctly. Once the ONS instruction is enabled, the rung-condition-in must go clear or the storage bit must be cleared for the ONS instruction to be enabled again.

On any scan for which *limit_switch_1* is cleared or *storage_1* is set, this rung has no affect. On any scan for which *limit_switch_1* is set and *storage_1* is cleared, the ONS instruction sets *storage_1* and the ADD instruction increments *sum* by 1. As long as *limit_switch_1* stays set, *sum* stays the same value. The *limit_switch_1* must go from cleared to set again for *sum* to be incremented again.

Relay Ladder



Structured Text

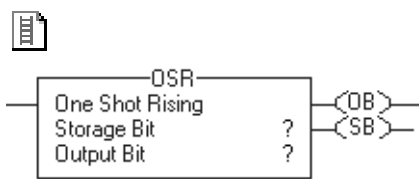
```
IF limit_switch_1 AND NOT storage_1 THEN
    sum := sum + 1;
END_IF;
storage_1 := limit_switch_1;
```

One Shot Rising (OSR)

The OSR instruction sets or clears the output bit, depending on the status of the storage bit.

This instruction is available in structured text and function block as OSRI, see page 1-19.

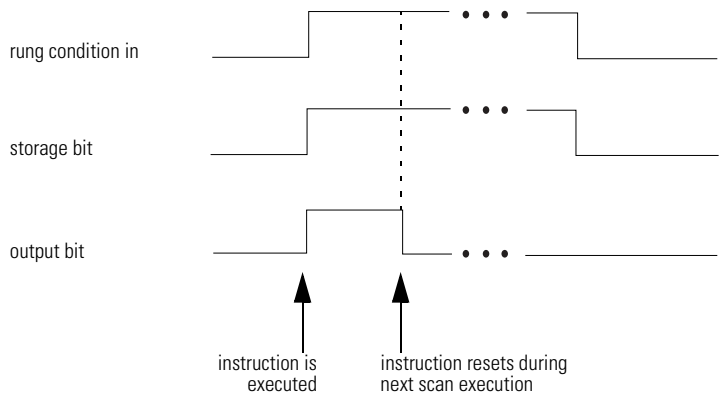
Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
storage bit	BOOL	tag	internal storage bit stores the rung-condition-in from the last time the instruction was executed
output bit	BOOL	tag	bit to be set

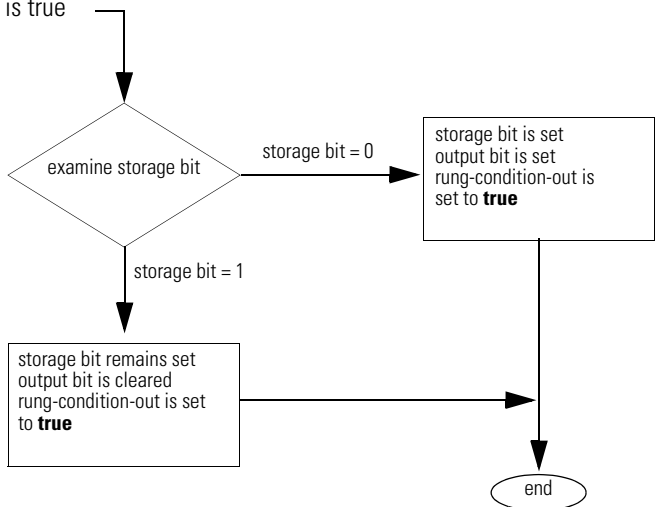
Description: When enabled and the storage bit is cleared, the OSR instruction sets the output bit. When enabled and the storage bit is set or when disabled, the OSR instruction clears the output bit



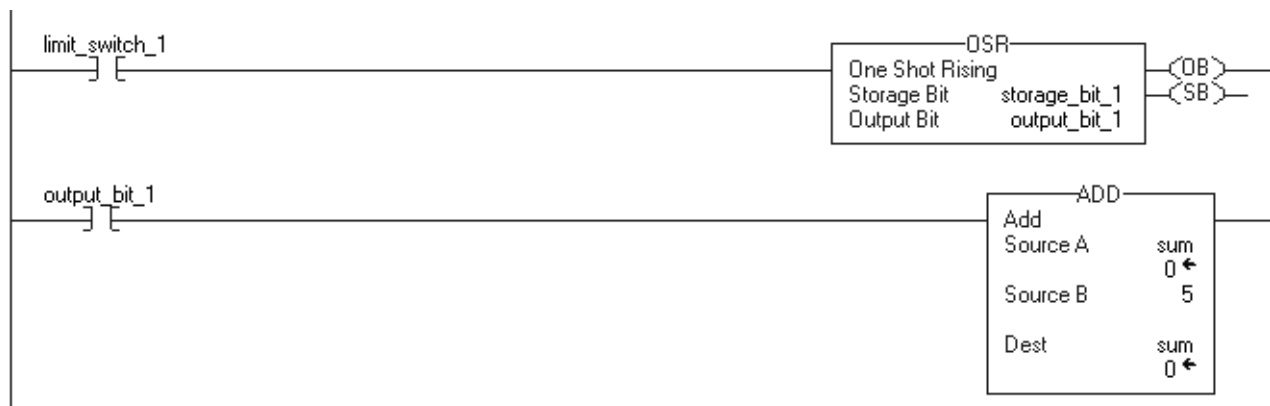
Arithmetic Status Flags: not affected

Fault Conditions: none

Execution:

Condition:	Relay Ladder Action:
prescan	The storage bit is set to prevent an invalid trigger during the first scan. The output bit is cleared. The rung-condition-out is set to false.
rung-condition-in is false	The storage bit is cleared. The output bit is not modified. The rung-condition-out is set to false.
rung-condition-in is true	 <pre> graph TD Start(()) --> Exam{examine storage bit} Exam -- "storage bit = 0" --> Box1[storage bit is set output bit is set rung-condition-out is set to true] Exam -- "storage bit = 1" --> Box2[storage bit remains set output bit is cleared rung-condition-out is set to true] Box1 --> End((end)) Box2 --> End </pre>
postscan	The storage bit is cleared. The output bit is not modified. The rung-condition-out is set to false.

Example: Each time *limit_switch_1* goes from cleared to set, the OSR instruction sets *output_bit_1* and the ADD instruction increments *sum* by 5. As long as *limit_switch_1* stays set, *sum* stays the same value. The *limit_switch_1* must go from cleared to set again for *sum* to be incremented again. You can use *output_bit_1* on multiple rungs to trigger other operations

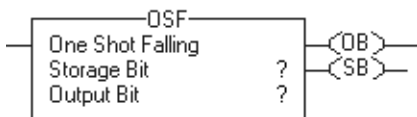


One Shot Falling (OSF)

The OSF instruction sets or clears the output bit depending on the status of the storage bit.

This instruction is available in structured text and function block as OSFI, see page 1-22.

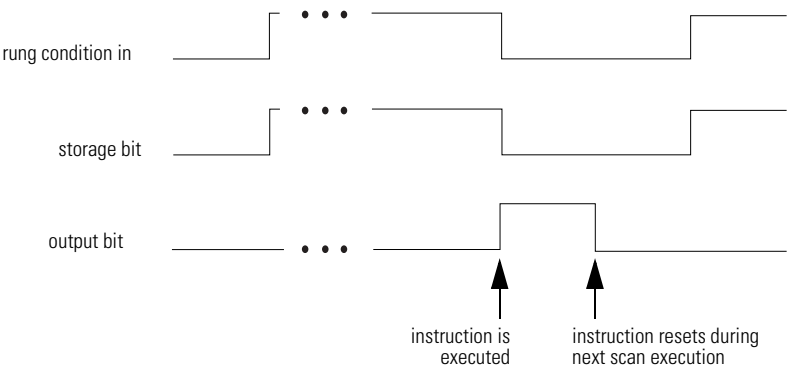
Operands:



Relay Ladder Operands

Operand:	Type:	Format:	Description:
storage bit	BOOL	tag	internal storage bit stores the rung-condition-in from the last time the instruction was executed
output bit	BOOL	tag	bit to be set

Description: When disabled and the storage bit is set, the OSF instruction sets the output bit. When disabled and the storage bit is cleared, or when enabled, the OSF instruction clears the output bit.



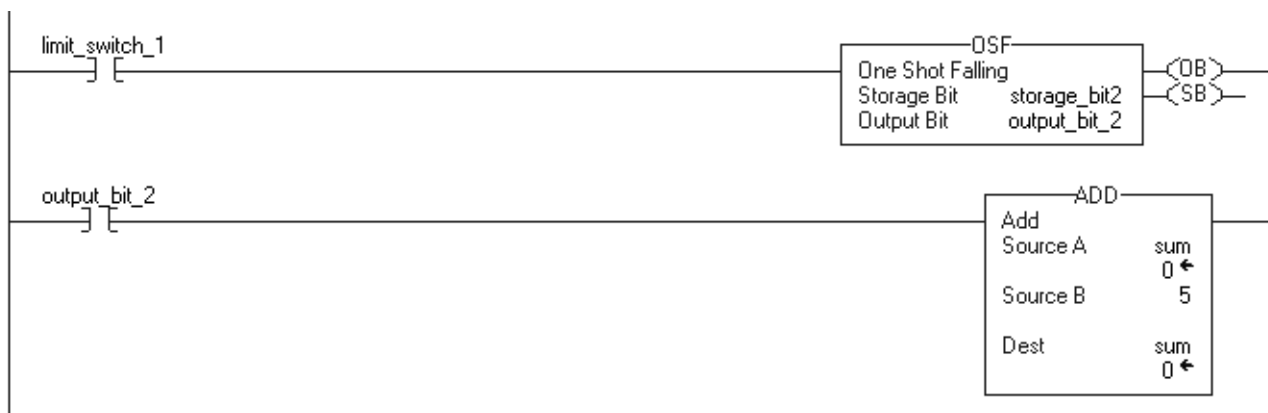
Arithmetic Status Flags: not affected

Fault Conditions: none

Execution:

Condition:	Relay Ladder Action:
prescan	The storage bit is cleared to prevent an invalid trigger during the first scan. The output bit is cleared. The rung-condition-out is set to false.
<pre> graph TD Start([rung-condition-in is false]) --> Exam{examine storage bit} Exam -- "storage bit = 0" --> Box1["storage bit remains cleared output bit is cleared rung-condition-out is set to false"] Exam -- "storage bit = 1" --> Box2["storage bit is cleared output bit is set rung-condition-out is set to false"] Box1 --> End([end]) Box2 --> End </pre>	
rung-condition-in is true	The storage bit is set. The output bit is cleared. The rung-condition-out is set to true.
postscan	See rung-condition-in is false above.

Example: Each time *limit_switch_1* goes from set to cleared, the OSF instruction sets *output_bit_2* and the ADD instruction increments *sum* by 5. As long as *limit_switch_1* stays cleared, *sum* stays the same value. The *limit_switch_1* must go from set to cleared again for *sum* to be incremented again. You can use *output_bit_2* on multiple rungs to trigger other operations.



One Shot Rising with Input (OSRI)

The OSRI instruction sets the output bit for one execution cycle when the input bit toggles from cleared to set.

This instruction is available in relay ladder as OSR, see page 1-15.

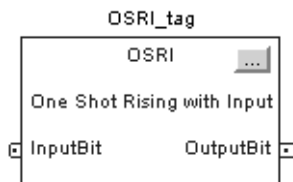
Operands:



OSRI (OSRI_tag) ;

Structured Text

Operand:	Type:	Format:	Description:
OSRI tag	FBD_ONESHOT	structure	OSRI structure



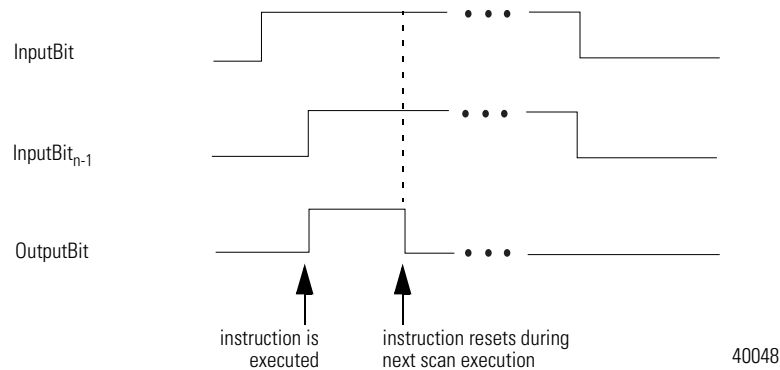
Function Block

Operand:	Type:	Format:	Description:
OSRI tag	FBD_ONESHOT	structure	OSRI structure

FBD_ONESHOT Structure

Input Parameter:	Data Type:	Description:
EnableIn	BOOL	Function Block: If cleared, the instruction does not execute and outputs are not updated. If set, the instruction executes. Default is set. Structured Text: No effect. The instruction executes.
InputBit	BOOL	Input bit. This is equivalent to rung condition for the relay ladder OSR instruction. Default is cleared.
Output Parameter:	Data Type:	Description:
EnableOut	BOOL	The instruction produced a valid result.
OutputBit	BOOL	Output bit

Description: When InputBit is set and InputBit_{n-1} is cleared, the OSRI instruction sets OutputBit. When InputBit_{n-1} is set or when InputBit is cleared, the OSRI instruction clears OutputBit.



Arithmetic Status Flags: not affected

Fault Conditions: none

Execution:

Condition:	Function Block Action:	Structured Text Action:
prescan	No action taken.	No action taken.
instruction first scan	InputBit _{n-1} is set.	InputBit _{n-1} is set.
instruction first run	InputBit _{n-1} is set.	InputBit _{n-1} is set.
EnableIn is cleared	EnableOut is cleared, the instruction does nothing, and the outputs are not updated.	na
EnableIn is set	On a cleared to set transition of InputBit, the instruction sets InputBit _{n-1} . The instruction executes. EnableOut is set.	On a cleared to set transition of InputBit, the instruction sets InputBit _{n-1} . EnableIn is always set. The instruction executes.
postscan	No action taken.	No action taken.

Example: When *limit_switch1* goes from cleared to set, the OSRI instruction sets OutputBit for one scan.

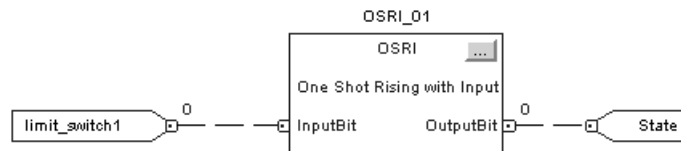
Structured Text

```
OSRI_01.InputBit := limit_switch1;
```

```
OSRI(OSRI_01);
```

```
State := OSRI_01.OutputBit;
```

Function Block



One Shot Falling with Input (OSFI)

The OSFI instruction sets the OutputBit for one execution cycle when the InputBit toggles from set to cleared.

This instruction is available in relay ladder as OSF, see page 1-17.

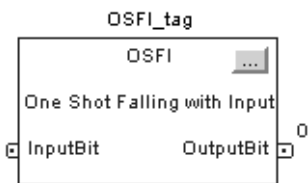
Operands:



OSFI (OSFI_tag) ;

Structured Text

Operand:	Type:	Format:	Description:
OSFI tag	FBD_ONESHOT	structure	OSFI structure



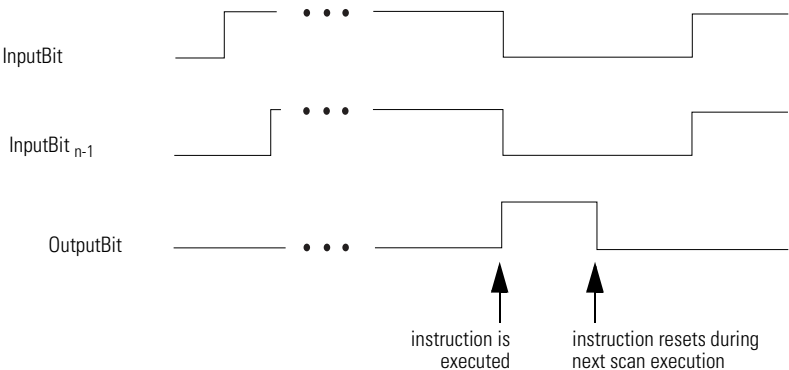
Function Block

Operand:	Type:	Format:	Description:
OSFI tag	FBD_ONESHOT	structure	OSFI structure

FBD_ONESHOT Structure

Input Parameter:	Data Type:	Description:
EnableIn	BOOL	Function Block: If cleared, the instruction does not execute and outputs are not updated. If set, the instruction executes. Default is set. Structured Text: No effect. The instruction executes.
InputBit	BOOL	Input bit. This is equivalent to rung condition for the relay ladder OSF instruction Default is cleared.
Output Parameter:	Data Type:	Description:
EnableOut	BOOL	The instruction produced a valid result.
OutputBit	BOOL	Output bit

Description: When the InputBit is cleared and the InputBit_{n-1} is set, the OSFI instruction sets the OutputBit. When InputBit_{n-1} is cleared or when InputBit is set, the OSFI instruction clears the OutputBit.



40047

Arithmetic Status Flags: not affected

Fault Conditions: none

Execution:

Condition:	Function Block Action:	Structured Text Action:
prescan	No action taken.	No action taken.
instruction first scan	InputBit _{n-1} is cleared.	InputBit _{n-1} is cleared.
instruction first run	InputBit _{n-1} is cleared.	InputBit _{n-1} is cleared.
EnableIn is cleared	EnableOut is cleared, the instruction does nothing, and the outputs are not updated.	na
EnableIn is set	On a cleared to set transition of InputBit, the instruction clears InputBit _{n-1} . The instruction executes. EnableOut is set.	On a cleared to set transition of InputBit, the instruction clears InputBit _{n-1} . EnableIn is always set. The instruction executes.
postscan	No action taken.	No action taken.

Example: When *limit_switch1* goes from set to cleared, the OSFI instruction sets OutputBit for one scan.

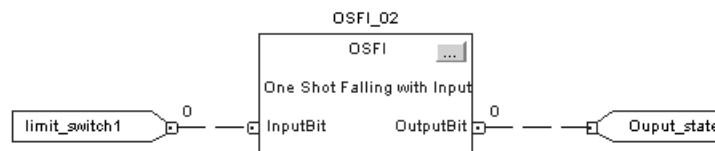
Structured Text

```
OSFI_01.InputBit := limit_switch1;
```

```
OSFI(OSFI_01);
```

```
Output_state := OSFI_01.OutputBit;
```

Function Block



Timer and Counter Instructions

(TON, TOF, RTO, TONR, TOFR, RTOR, CTU, CTD, CTUD, RES)

Introduction

Timers and counters control operations based on time or the number of events.

If you want to:	Use this instruction:	Available in these languages:	See page:
time how long a timer is enabled	TON	relay ladder	2-2
time how long a timer is disabled	TOF	relay ladder	2-6
accumulate time	RTO	relay ladder	2-10
time how long a timer is enabled with built-in reset in function block	TONR	structured text function block	2-14
time how long a timer is disabled with built-in reset in function block	TOFR	structure text function block	2-17
accumulate time with built-in reset in function block	RTOR	structured text function block	2-20
count up	CTU	relay ladder	2-24
count down	CTD	relay ladder	2-28
count up and count down in function block	CTUD	structured text function block	2-32
reset a timer or counter	RES	relay ladder	2-36

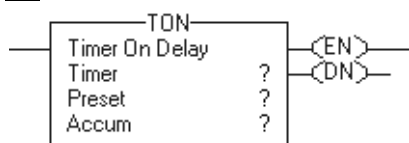
The time base for all timers is 1 msec.

Timer On Delay (TON)

The TON instruction is a non-retentive timer that accumulates time when the instruction is enabled (rung-condition-in is true).

This instruction is available in structured text and function block as TONR, see page 2-14.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Timer	TIMER	tag	timer structure
Preset	DINT	immediate	how long to delay (accumulate time)
Accum	DINT	immediate	total msec the timer has counted initial value is typically 0

TIMER Structure

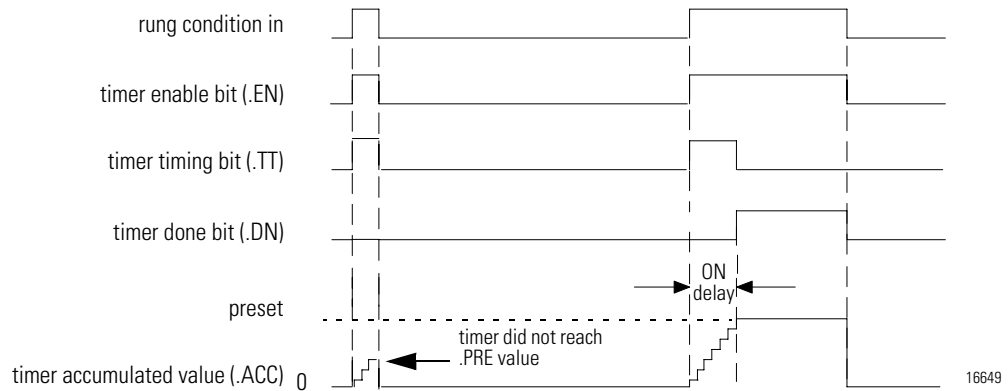
Mnemonic:	Data Type:	Description:
.EN	BOOL	The enable bit indicates that the TON instruction is enabled.
.TT	BOOL	The timing bit indicates that a timing operation is in process
.DN	BOOL	The done bit is set when $.ACC \geq .PRE$.
.PRE	DINT	The preset value specifies the value (1 msec units) which the accumulated value must reach before the instruction sets the .DN bit.
.ACC	DINT	The accumulated value specifies the number of milliseconds that have elapsed since the TON instruction was enabled.

Description: The TON instruction accumulates time until:

- the TON instruction is disabled
- the $.ACC \geq .PRE$

The time base is always 1 msec. For example, for a 2-second timer, enter 2000 for the .PRE value.

When the TON instruction is disabled, the .ACC value is cleared.

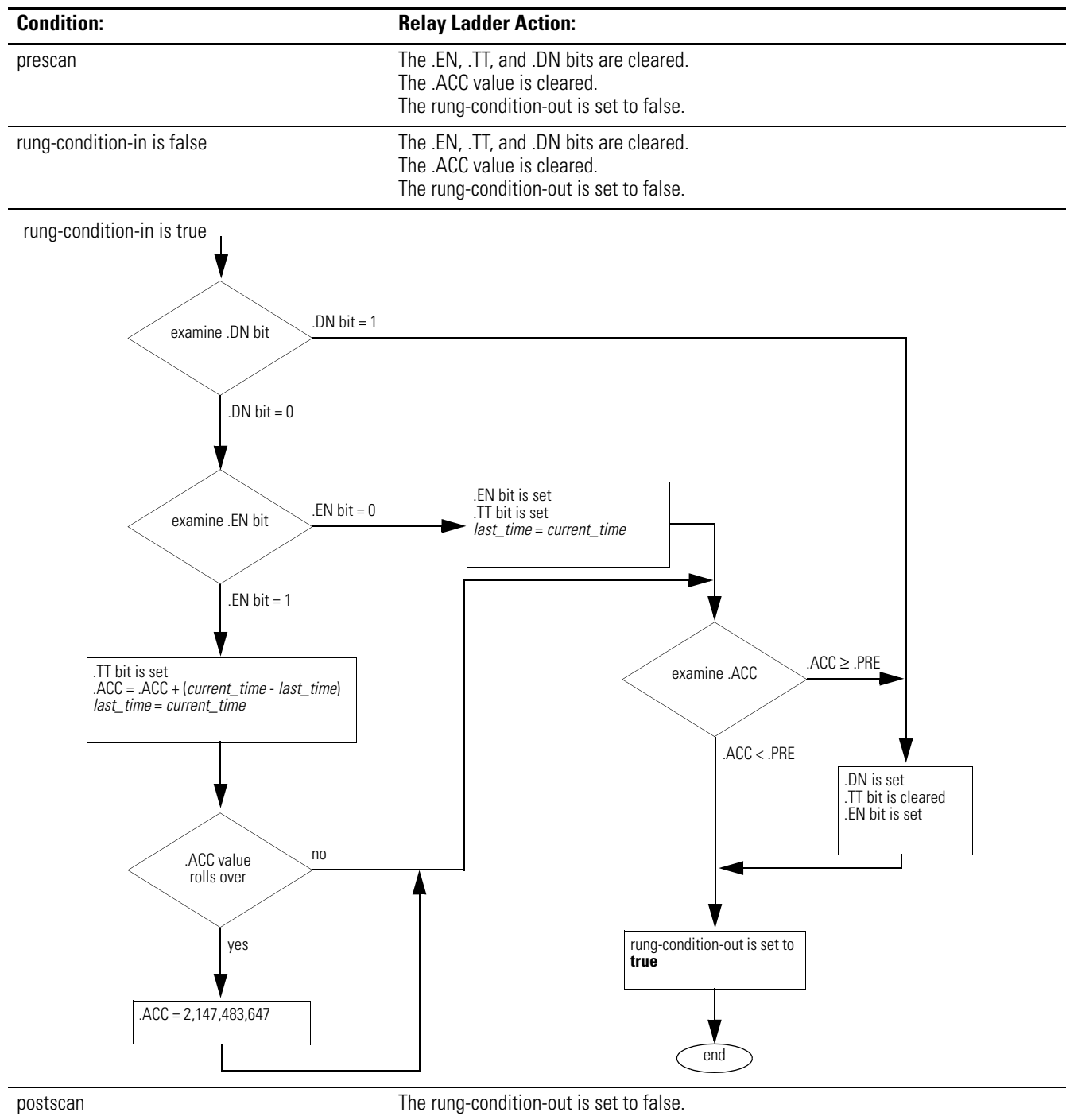


Arithmetic Status Flags: not affected

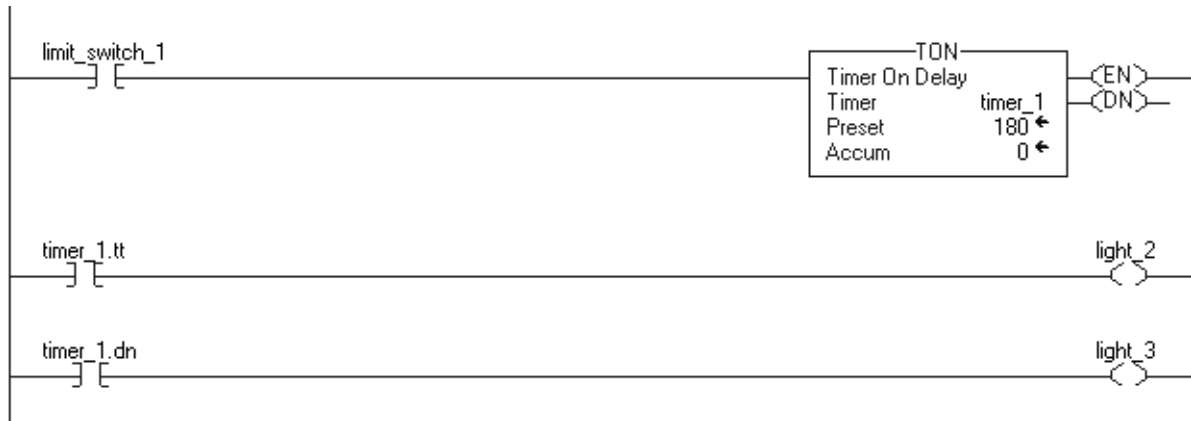
Fault Conditions:

A major fault will occur if:	Fault type:	Fault code:
.PRE < 0	4	34
.ACC < 0	4	34

Execution:



Example: When *limit_switch_1* is set, *light_2* is on for 180 msec (*timer_1* is timing). When *timer_1.acc* reaches 180, *light_2* goes off and *light_3* goes on. *Light_3* remains on until the TON instruction is disabled. If *limit_switch_1* is cleared while *timer_1* is timing, *light_2* goes off.

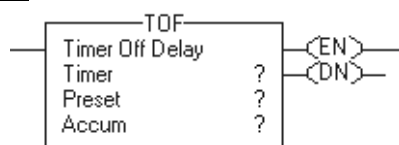


Timer Off Delay (TOF)

The TOF instruction is a non-retentive timer that accumulates time when the instruction is enabled (rung-condition-in is false).

This instruction is available in structured text and function block as TOFR, see page 2-17.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Timer	TIMER	tag	timer structure
Preset	DINT	immediate	how long to delay (accumulate time)
Accum	DINT	immediate	total msec the timer has counted initial value is typically 0

TIMER Structure

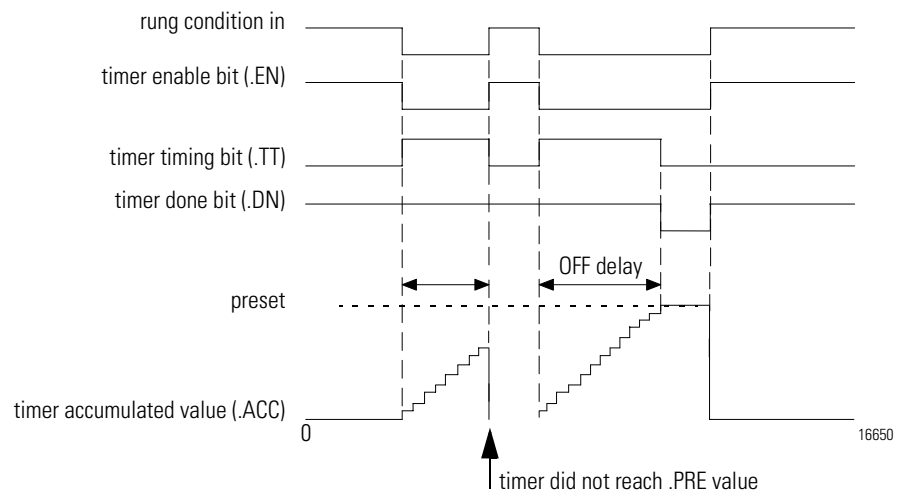
Mnemonic:	Data Type:	Description:
.EN	BOOL	The enable bit indicates that the TOF instruction is enabled.
.TT	BOOL	The timing bit indicates that a timing operation is in process
.DN	BOOL	The done bit is cleared when $.ACC \geq .PRE$.
.PRE	DINT	The preset value specifies the value (1 msec units) which the accumulated value must reach before the instruction clears the .DN bit.
.ACC	DINT	The accumulated value specifies the number of milliseconds that have elapsed since the TOF instruction was enabled.

Description: The TOF instruction accumulates time until:

- the TOF instruction is disabled
- the $.ACC \geq .PRE$

The time base is always 1 msec. For example, for a 2-second timer, enter 2000 for the .PRE value.

When the TOF instruction is disabled, the .ACC value is cleared.

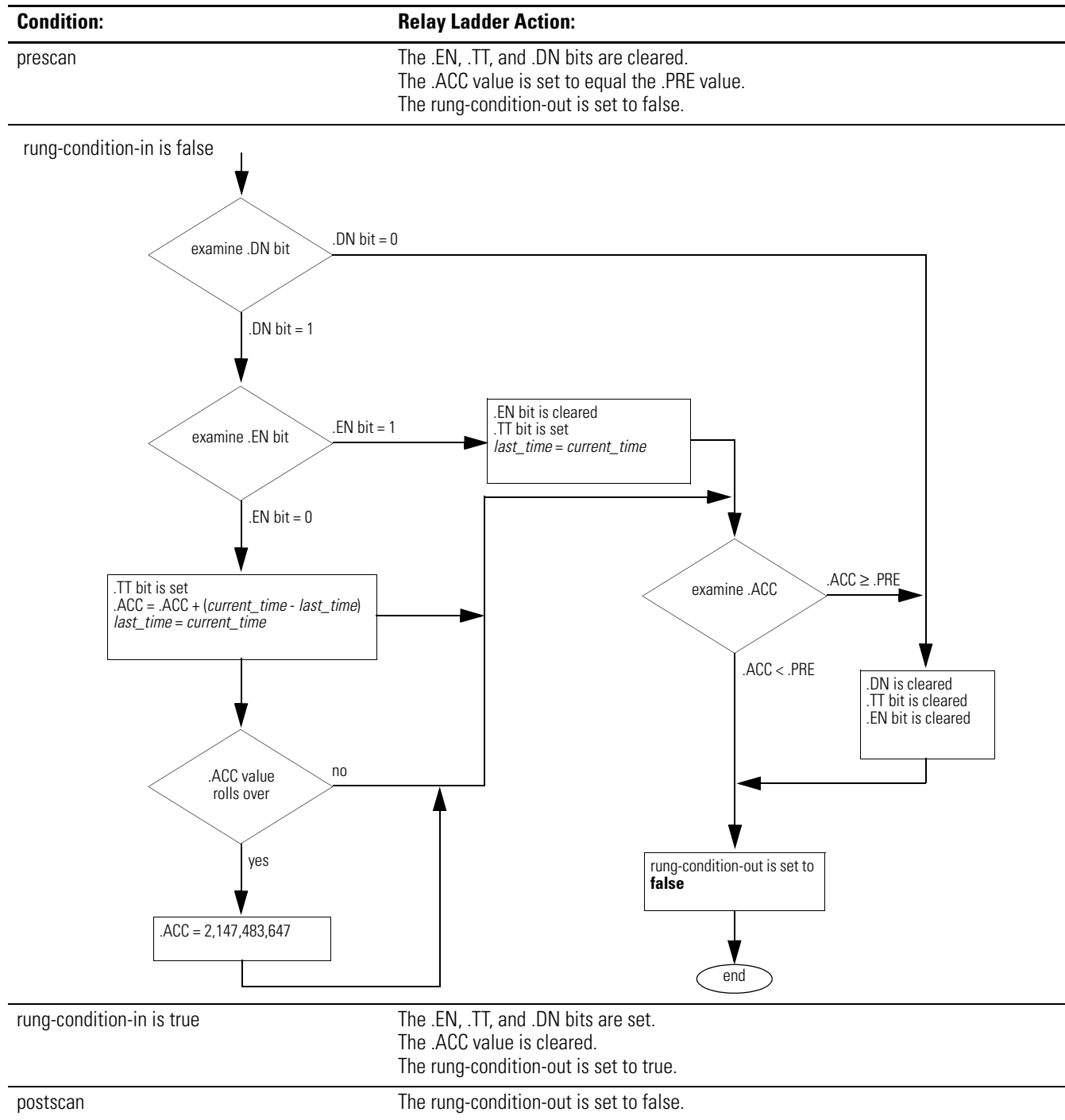


Arithmetic Status Flags: not affected

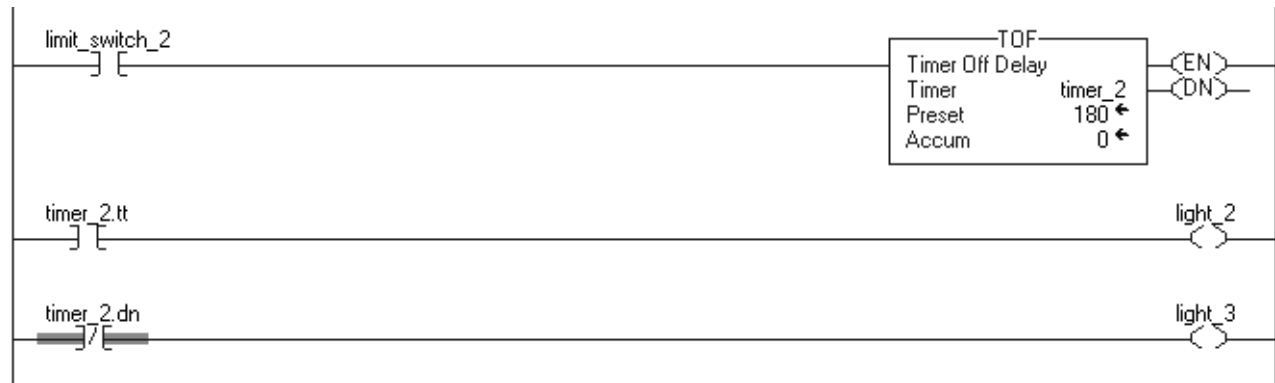
Fault Conditions:

A major fault will occur if:	Fault type:	Fault code:
.PRE < 0	4	34
.ACC < 0	4	34

Execution:



Example: When *limit_switch_2* is cleared, *light_2* is on for 180 msec (*timer_2* is timing). When *timer_2.acc* reaches 180, *light_2* goes off and *light_3* goes on. *Light_3* remains on until the TOF instruction is enabled. If *limit_switch_2* is set while *timer_2* is timing, *light_2* goes off.

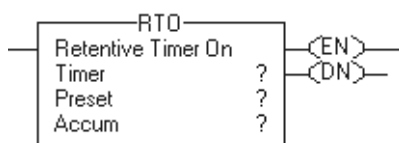


Retentive Timer On (RTO)

The RTO instruction is a retentive timer that accumulates time when the instruction is enabled.

This instruction is available in structured text and function block as RTOR, see page 2-20.

Operands:



Relay Ladder

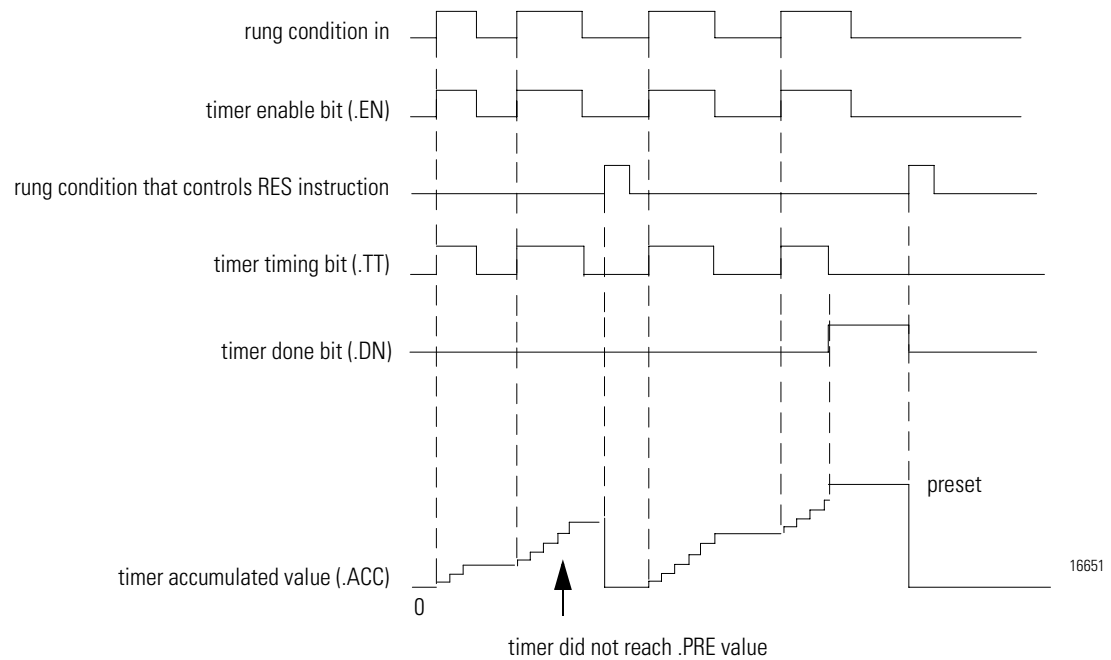
Operand:	Type:	Format:	Description:
Timer	TIMER	tag	timer structure
Preset	DINT	immediate	how long to delay (accumulate time)
Accum	DINT	immediate	number of msec the timer has counted initial value is typically 0

TIMER Structure

Mnemonic:	Data Type:	Description:
.EN	BOOL	The enable bit indicates that the RTO instruction is enabled.
.TT	BOOL	The timing bit indicates that a timing operation is in process
.DN	BOOL	The done bit indicates that $.ACC \geq .PRE$.
.PRE	DINT	The preset value specifies the value (1 msec units) which the accumulated value must reach before the instruction sets the .DN bit.
.ACC	DINT	The accumulated value specifies the number of milliseconds that have elapsed since the RTO instruction was enabled.

Description: The RTO instruction accumulates time until it is disabled. When the RTO instruction is disabled, it retains its .ACC value. You must clear the .ACC value, typically with a RES instruction referencing the same TIMER structure.

The time base is always 1 msec. For example, for a 2-second timer, enter 2000 for the .PRE value.

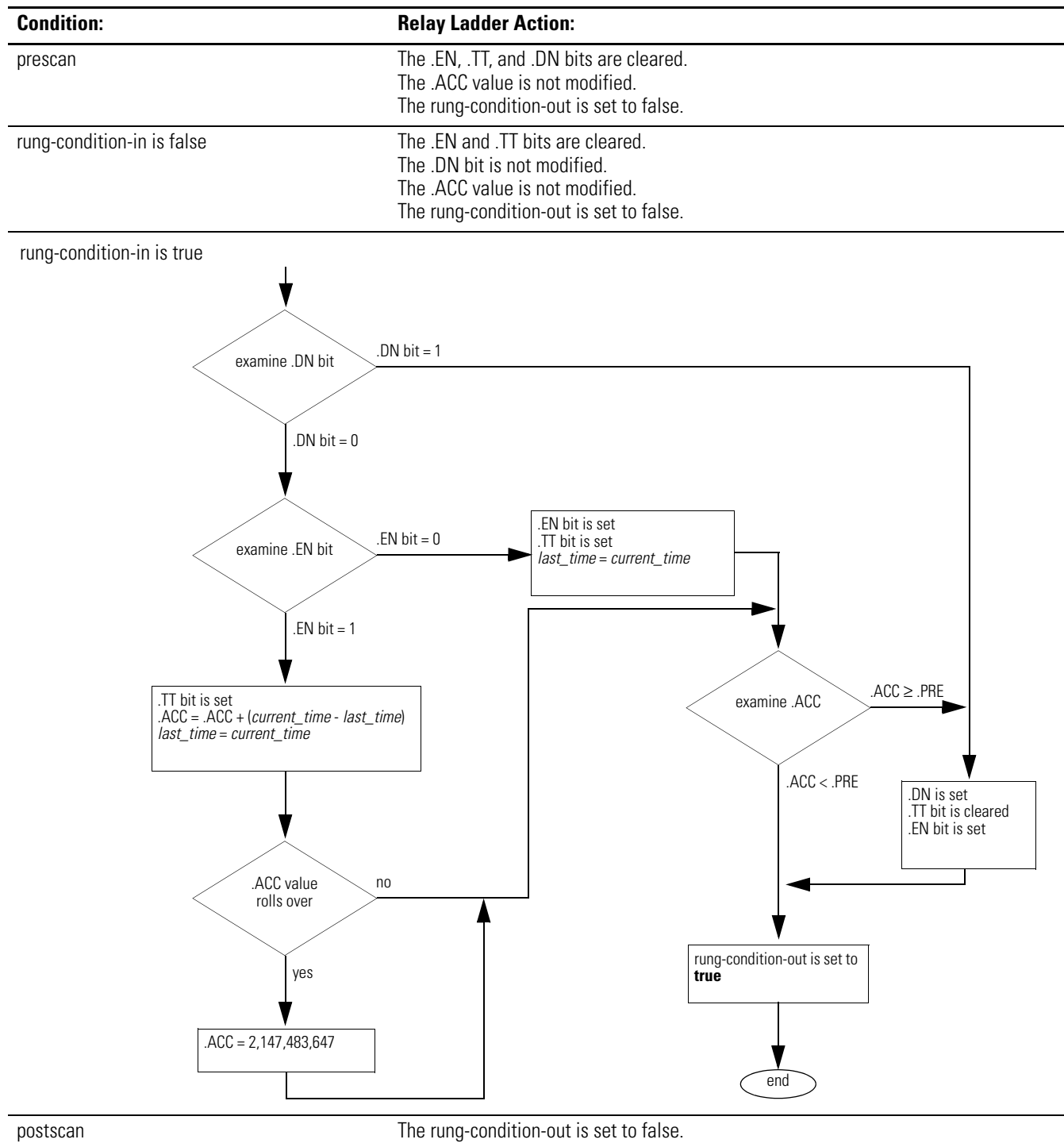


Arithmetic Status Flags: not affected

Fault Conditions:

A major fault will occur if:	Fault type:	Fault code:
.PRE < 0	4	34
.ACC < 0	4	34

Execution:



Example: When *limit_switch_1* is set, *light_1* is on for 180 msec (*timer_2* is timing). When *timer_3.acc* reaches 180, *light_1* goes off and *light_2* goes on. *Light_2* remains until *timer_3* is reset. If *limit_switch_2* is cleared while *timer_3* is timing, *light_1* remains on. When *limit_switch_2* is set, the RES instruction resets *timer_3* (clears status bits and .ACC value).



Timer On Delay with Reset (TONR)

The TONR instruction is a non-retentive timer that accumulates time when TimerEnable is set.

This instruction is available in relay ladder as two separate instructions: TON (see page 2-2) and RES (see page 2-36).

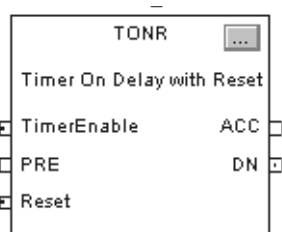
Operands:



TONR (TONR_tag) ;

Structured Text

Variable:	Type:	Format:	Description:
TONR tag	FBD_TIMER	structure	TONR structure



Function Block

Operand:	Type:	Format:	Description:
TONR tag	FBD_TIMER	structure	TONR structure

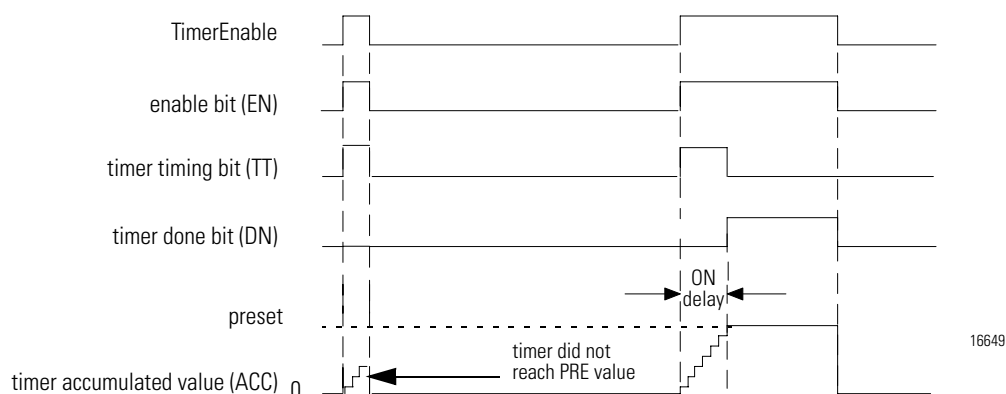
FBD_TIMER Structure

Input Parameter:	Data Type:	Description:
EnableIn	BOOL	Function Block: If cleared, the instruction does not execute and outputs are not updated. If set, the instruction executes. Default is set. Structured Text: No effect. The instruction executes.
TimerEnable	BOOL	If set, this enables the timer to run and accumulate time. Default is cleared.
PRE	DINT	Timer preset value. This is the value in 1msec units that ACC must reach before timing is finished. If invalid, the instruction sets the appropriate bit in Status and the timer does not execute. Valid = 0 to maximum positive integer
Reset	BOOL	Request to reset the timer. When set, the timer resets. Default is cleared.
Output Parameter:	Data Type:	Description:
EnableOut	BOOL	The instruction produced a valid result.
ACC	BOOL	Accumulated time in milliseconds.
EN	BOOL	Timer enabled output. Indicates the timer instruction is enabled.
TT	BOOL	Timer timing output. When set, a timing operation is in progress.
DN	BOOL	Timing done output. Indicates when the accumulated time is greater than or equal to the preset value.
Status	DINT	Status of the function block.
InstructFault (Status.0)	BOOL	The instruction detected one of the following execution errors. This is not a minor or major controller error. Check the remaining status bits to determine what occurred.
PresetInv (Status.1)	BOOL	The preset value is invalid.

Description: The TONR instruction accumulates time until the:

- TONR instruction is disabled
- $ACC \geq PRE$

The time base is always 1 msec. For example, for a 2-second timer, enter 2000 for the PRE value.



Set the Reset input parameter to reset the instruction. If TimerEnable is set when Reset is set, the TONR instruction begins timing again when Reset is cleared.

Arithmetic Status Flags: not affected

Fault Conditions: none

Execution:

Condition:	Function Block Action:	Structured Text Action:
prescan	No action taken.	No action taken.
instruction first scan	EN, TT and DN are cleared. ACC value is set to 0.	EN, TT and DN are cleared. ACC value is set to 0.
instruction first run	EN, TT and DN are cleared. ACC value is set to 0.	EN, TT and DN are cleared. ACC value is set to 0.
EnableIn is cleared	EnableOut is cleared, the instruction does nothing, and the outputs are not updated.	na
EnableIn is set	When EnableIn transitions from cleared to set, the instruction initializes as described for instruction first scan. The instruction executes. EnableOut is set.	EnableIn is always set. The instruction executes.
reset	When the Reset input parameter is set, the instruction clears EN, TT and DN and sets ACC = zero.	When the Reset input parameter is set, the instruction clears EN, TT and DN and sets ACC = zero.
postscan	No action taken.	No action taken.

Example: Each scan that *limit_switch1* is set, the TONR instruction increments the ACC value by elapsed time until the ACC value reaches the PRE value. When $ACC \geq PRE$, the DN parameter is set, and *timer_state* is set.

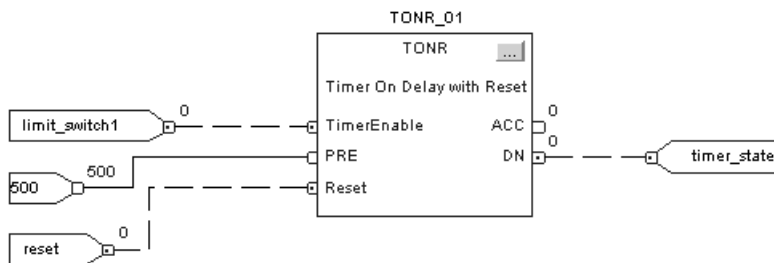
Structured Text

```
TONR_01.Preset := 500;
TONR_01.Reset  := reset;
TONR_01.TimerEnable := limit_switch1;

TONR(TONR_01);

timer_state := TONR_01.DN;
```

Function Block Example



Timer Off Delay with Reset (TOFR)

The TOFR instruction is a non-retentive timer that accumulates time when TimerEnable is cleared.

This instruction is available in relay ladder as two separate instructions: TOF (see page 2-6) and RES (see page 2-36).

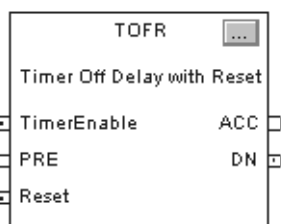
Operands:



TOFR (TOFR_tag) ;

Structured Text

Variable:	Type:	Format:	Description:
TOFR tag	FBD_TIMER	structure	TOFR structure



Function Block Operands

Operand:	Type:	Format:	Description:
TOFR tag	FBD_TIMER	structure	TOFR structure

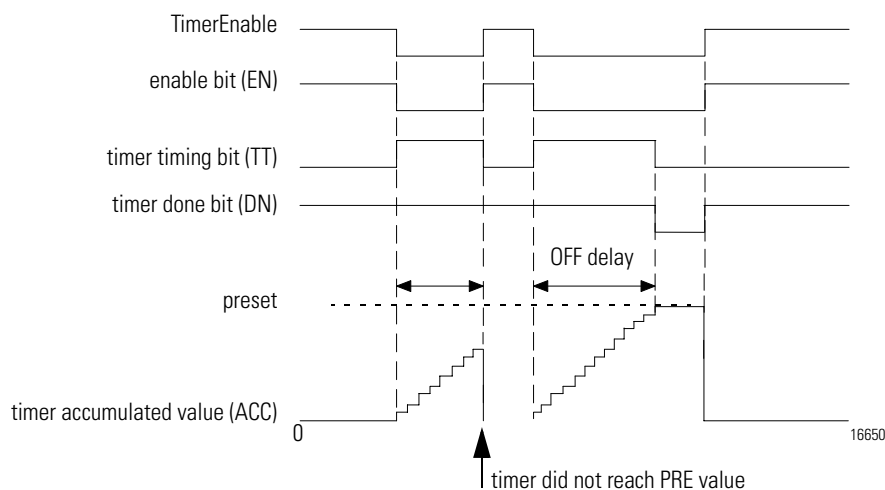
FBD_TIMER Structure

Input Parameter:	Data Type:	Description:
EnableIn	BOOL	Function Block: If cleared, the instruction does not execute and outputs are not updated. If set, the instruction executes. Default is set. Structured Text: No effect. The instruction executes.
TimerEnable	BOOL	If cleared, this enables the timer to run and accumulate time. Default is cleared.
PRE	DINT	Timer preset value. This is the value in 1msec units that ACC must reach before timing is finished. If invalid, the instructions sets the appropriate bit in Status and the timer does not execute. Valid = 0 to maximum positive integer
Reset	BOOL	Request to reset the timer. When set, the timer resets. Default is cleared.
Output Parameter:	Data Type:	Description:
EnableOut	BOOL	The instruction produced a valid result.
ACC	BOOL	Accumulated time in milliseconds.
EN	BOOL	Timer enabled output. Indicates the timer instruction is enabled.
TT	BOOL	Timer timing output. When set, a timing operation is in progress.
DN	BOOL	Timing done output. Indicates when accumulated time is greater than or equal to preset.
Status	DINT	Status of the function block.
InstructFault (Status.0)	BOOL	The instruction detected one of the following execution errors. This is not a minor or major controller error. Check the remaining status bits to determine what occurred.
PresetInv (Status.1)	BOOL	The preset value is invalid.

Description: The TOFR instruction accumulates time until the:

- TOFR instruction is disabled
- $ACC \geq PRE$

The time base is always 1 msec. For example, for a 2-second timer, enter 2000 for the PRE value.



Set the Reset input parameter to reset the instruction. If TimerEnable is cleared when Reset is set, the TOFR instruction does not begin timing again when Reset is cleared.

Arithmetic Status Flags: not affected

Fault Conditions: none

Execution:

Condition:	Function Block Action:	Structured Text Action:
prescan	No action taken.	No action taken.
instruction first scan	EN, TT and DN are cleared. ACC value is set to PRE.	EN, TT and DN are cleared. ACC value is set to PRE.
instruction first run	EN, TT and DN are cleared. ACC value is set to PRE.	EN, TT and DN are cleared. ACC value is set to PRE.
EnableIn is cleared	EnableOut is cleared, the instruction does nothing, and the outputs are not updated.	na
EnableIn is set	When EnableIn transitions from cleared to set, the instruction initializes as described for instruction first scan. The instruction executes. EnableOut is set.	EnableIn is always set. The instruction executes.
reset	When the Reset input parameter is set, the instruction clears EN, TT and DN and sets $ACC = PRE$. Note that this is different than using a RES instruction on a TOF instruction.	When the Reset input parameter is set, the instruction clears EN, TT and DN and sets $ACC = PRE$. Note that this is different than using a RES instruction on a TOF instruction.
postscan	No action taken.	No action taken.

Example: Each scan after *limit_switch1* is cleared, the TOFR instruction increments the ACC value by elapsed time until the ACC value reaches the PRE value. When $ACC \geq PRE$, the DN parameter is cleared, and *timer_state2* is set.

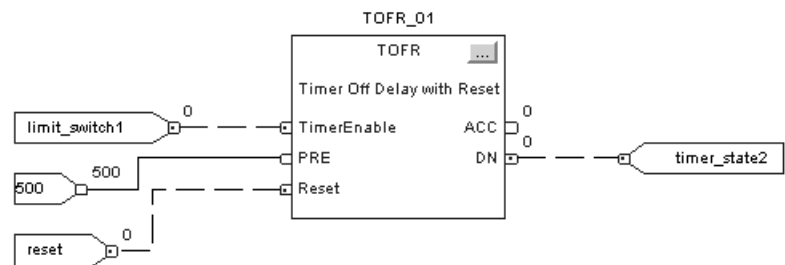
Structured Text

```
TOFR_01.Preset := 500
TOFR_01.Reset := reset;
TOFR_01.TimerEnable := limit_switch1;

TOFR(TOFR_01);

timer_state2 := TOFR_01.DN;
```

Function Block



Retentive Timer On with Reset (RTOR)

The RTOR instruction is a retentive timer that accumulates time when TimerEnable is set.

This instruction is available in relay ladder as two separate instructions: RTO (see page 2-10) and RES (see page 2-36).

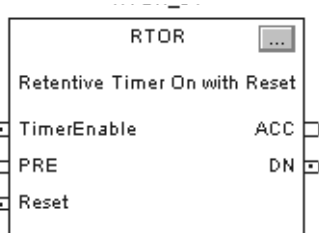
Operands:



RTOR (RTOR_tag) ;

Structured Text

Variable:	Type:	Format:	Description:
RTOR tag	FBD_TIMER	structure	RTOR structure



Function Block Operands

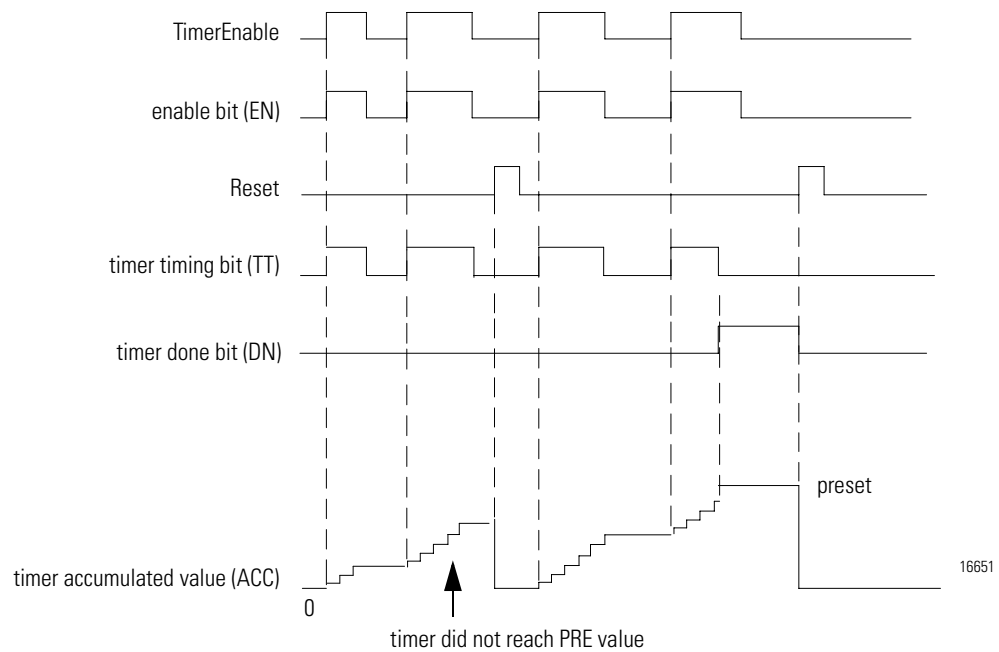
Operand:	Type:	Format:	Description:
RTOR tag	FBD_TIMER	structure	RTOR structure

FBD_TIMER Structure

Input Parameter:	Data Type:	Description:
EnableIn	BOOL	Function Block: If cleared, the instruction does not execute and outputs are not updated. If set, the instruction executes. Default is set. Structured Text: No effect. The instruction executes.
TimerEnable	BOOL	If set, this enables the timer to run and accumulate time. Default is cleared.
PRE	DINT	Timer preset value. This is the value in 1msec units that ACC must reach before timing is finished. If invalid, the instruction sets the appropriate bit in Status and the timer does not execute. Valid = 0 to maximum positive integer
Reset	BOOL	Request to reset the timer. When set, the timer resets.
Output Parameter:	Data Type:	Description:
EnableOut	BOOL	The instruction produced a valid result.
ACC	DINT	Accumulated time in milliseconds. This value is retained even while the TimerEnable input is cleared. This makes the behavior of this block different than the TONR block.
EN	BOOL	Timer enabled output. Indicates the timer instruction is enabled.
TT	BOOL	Timer timing output. When set, a timing operation is in progress.
DN	BOOL	Timing done output. Indicates when accumulated time is greater than or equal to preset.
Status	DINT	Status of the function block.
InstructFault (Status.0)	BOOL	The instruction detected one of the following execution errors. This is not a minor or major controller error. Check the remaining status bits to determine what occurred.
PresetInv (Status.1)	BOOL	The preset value is invalid.

Description: The RTOR instruction accumulates time until it is disabled. When the RTOR instruction is disabled, it retains its ACC value. You must clear the .ACC value using the Reset input.

The time base is always 1 msec. For example, for a 2-second timer, enter 2000 for the PRE value.



Set the Reset input parameter to reset the instruction. If TimerEnable is set when Reset is set, the RTOR instruction begins timing again when Reset is cleared.

Arithmetic Status Flags: not affected

Fault Conditions: none

Execution:

Condition:	Function Block Action:	Structured Text Action:
prescan	No action taken.	No action taken.
instruction first scan	EN, TT and DN are cleared ACC value is not modified	EN, TT and DN are cleared ACC value is not modified
instruction first run	EN, TT and DN are cleared ACC value is not modified	EN, TT and DN are cleared ACC value is not modified
EnableIn is cleared	EnableOut is cleared, the instruction does nothing, and the outputs are not updated.	na
EnableIn is set	Function Block: When EnableIn transitions from cleared to set, the instruction initializes as described for instruction first scan. The instruction executes. EnableOut is set.	EnableIn is always set. The instruction executes.
reset	When the Reset input parameter is set, the instruction clears EN, TT and DN and sets ACC = zero.	When the Reset input parameter is set, the instruction clears EN, TT and DN and sets ACC = zero.
postscan	No action taken.	No action taken.

Example: Each scan that *limit_switch1* is set, the RTOR instruction increments the ACC value by elapsed time until the ACC value reaches the PRE value. When $ACC \geq PRE$, the DN parameter is set, and *timer_state3* is set.

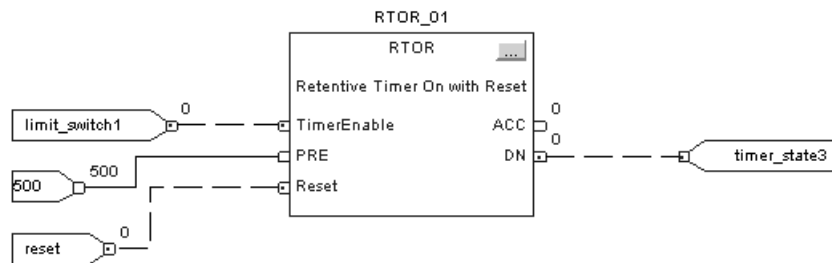
Structured Text

```
RTOR_01.Preset := 500
RTOR_01.Reset := reset;
RTOR_01.TimerEnable := limit_switch1;

RTOR(RTOR_01);

timer_state3 := RTOR_01.DN;
```

Function Block

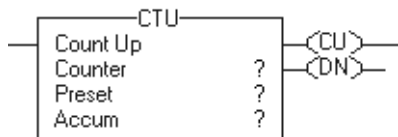


Count Up (CTU)

The CTU instruction counts upward.

This instruction is available in structured text and function block as CTUD, see page 2-32.

Operands:



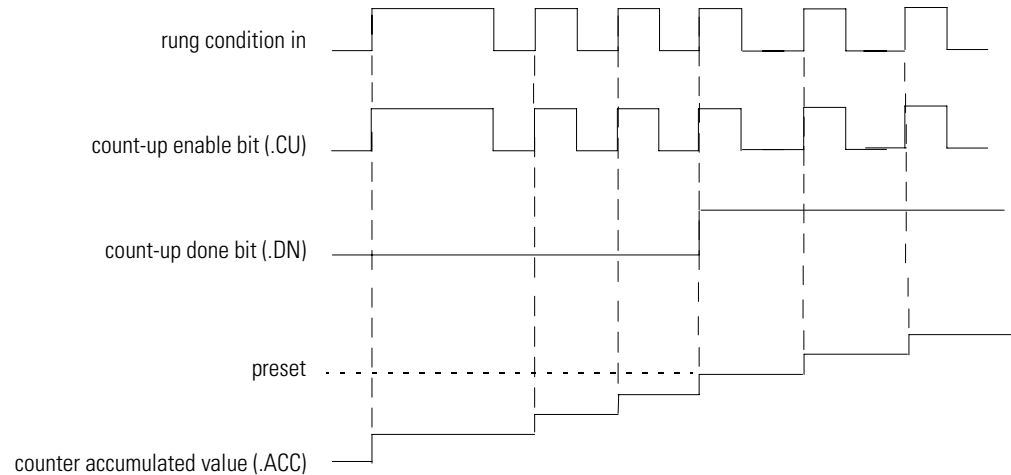
Relay Ladder

Operand:	Type:	Format:	Description:
Counter	COUNTER	tag	counter structure
Preset	DINT	immediate	how high to count
Accum	DINT	immediate	number of times the counter has counted initial value is typically 0

COUNTER Structure

Mnemonic:	Data Type:	Description:
.CU	BOOL	The count up enable bit indicates that the CTU instruction is enabled.
.DN	BOOL	The done bit indicates that $.ACC \geq .PRE$.
.OV	BOOL	The overflow bit indicates that the counter exceeded the upper limit of 2,147,483,647. The counter then rolls over to -2,147,483,648 and begins counting up again.
.UN	BOOL	The underflow bit indicates that the counter exceeded the lower limit of -2,147,483,648. The counter then rolls over to 2,147,483,647 and begins counting down again.
.PRE	DINT	The preset value specifies the value which the accumulated value must reach before the instruction sets the .DN bit.
.ACC	DINT	The accumulated value specifies the number of transitions the instruction has counted.

Description: When enabled and the .CU bit is cleared, the CTU instruction increments the counter by one. When enabled and the .CU bit is set, or when disabled, the CTU instruction retains its .ACC value.



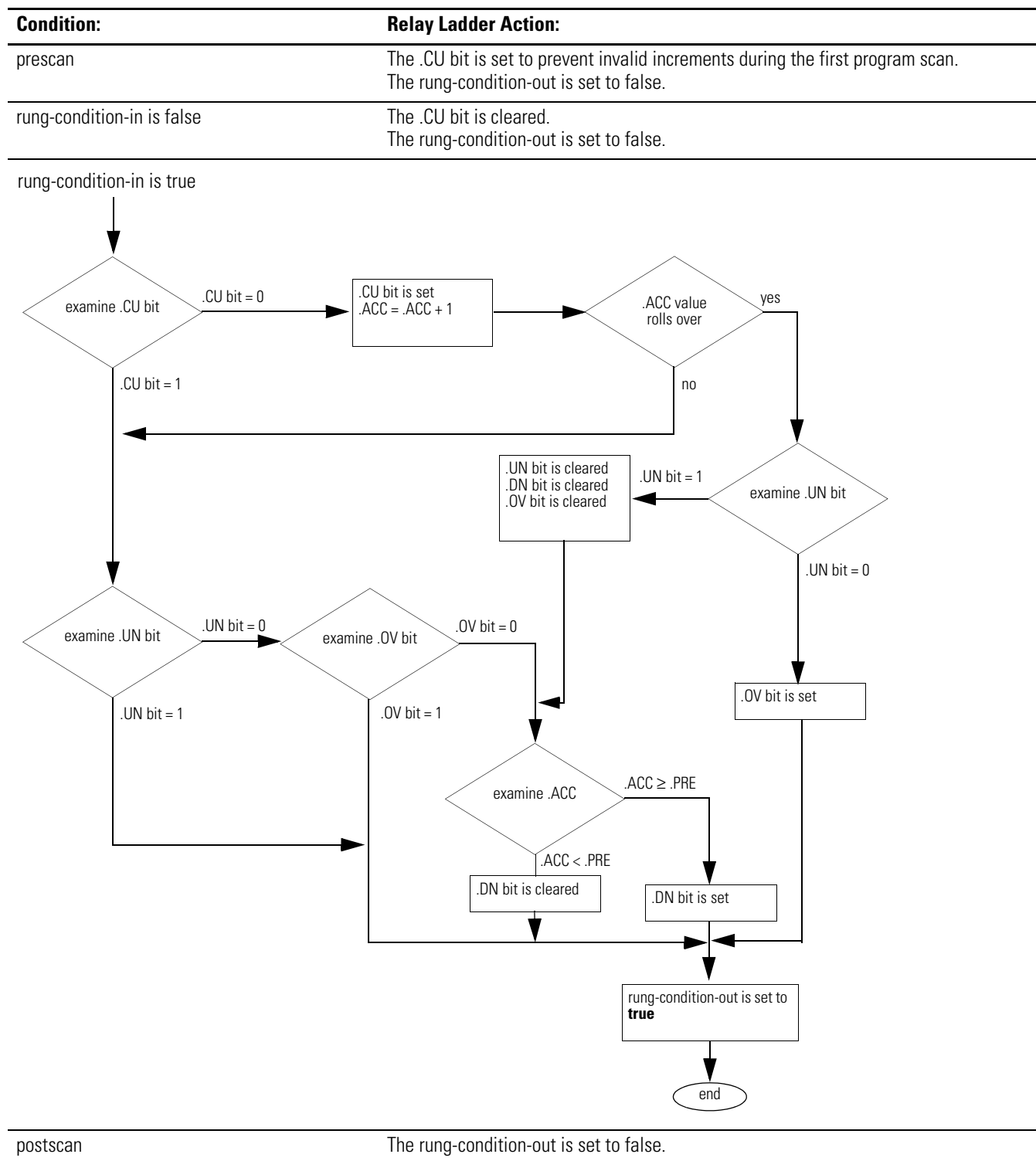
16636

The accumulated value continues incrementing, even after the .DN bit is set. To clear the accumulated value, use a RES instruction that references the counter structure or write 0 to the accumulated value.

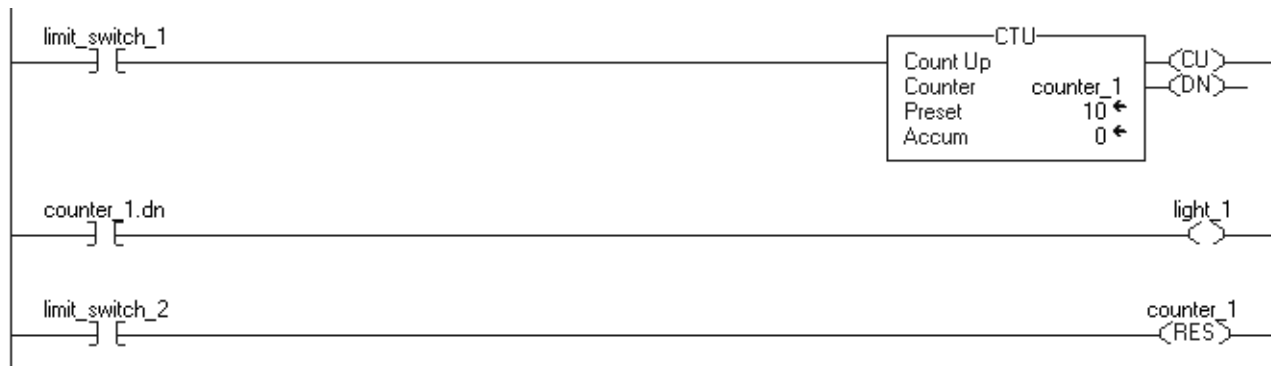
Arithmetic Status Flags: not affected

Fault Conditions: none

Execution:



Example: After *limit_switch_1* goes from disabled to enabled 10 times, the .DN bit is set and *light_1* turns on. If *limit_switch_1* continues to go from disabled to enabled, *counter_1* continues to increment its count and the .DN bit remains set. When *limit_switch_2* is enabled, the RES instruction resets *counter_1* (clears the status bits and the .ACC value) and *light_1* turns off.

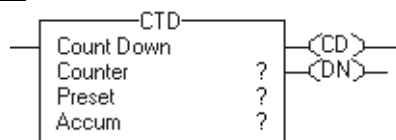


Count Down (CTD)

The CTD instruction counts downward.

This instruction is available in structured text and function block as CTUD, see page 2-32.

Operands:



Relay Ladder

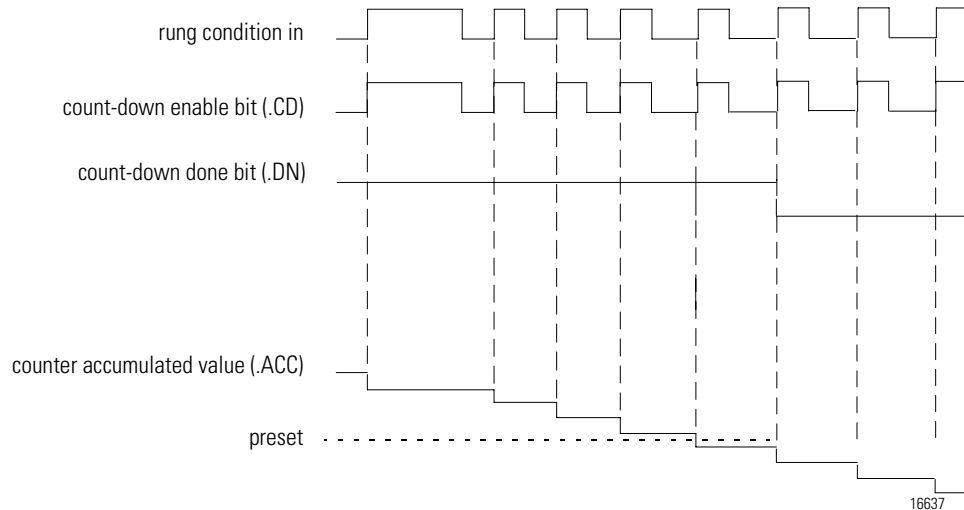
Operand:	Type:	Format:	Description:
Counter	COUNTER	tag	counter structure
Preset	DINT	immediate	how low to count
Accum	DINT	immediate	number of times the counter has counted initial value is typically 0

COUNTER Structure

Mnemonic:	Data Type:	Description:
.CD	BOOL	The count down enable bit indicates that the CTD instruction is enabled.
.DN	BOOL	The done bit indicates that $.ACC \geq .PRE$.
.OV	BOOL	The overflow bit indicates that the counter exceeded the upper limit of 2,147,483,647. The counter then rolls over to -2,147,483,648 and begins counting up again.
.UN	BOOL	The underflow bit indicates that the counter exceeded the lower limit of -2,147,483,648. The counter then rolls over to 2,147,483,647 and begins counting down again.
.PRE	DINT	The preset value specifies the value which the accumulated value must reach before the instruction sets the .DN bit.
.ACC	DINT	The accumulated value specifies the number of transitions the instruction has counted.

Description: The CTD instruction is typically used with a CTU instruction that references the same counter structure.

When enabled and the .CD bit is cleared, the CTD instruction decrements the counter by one. When enabled and the .CD bit is set, or when disabled, the CTD instruction retains its .ACC value.

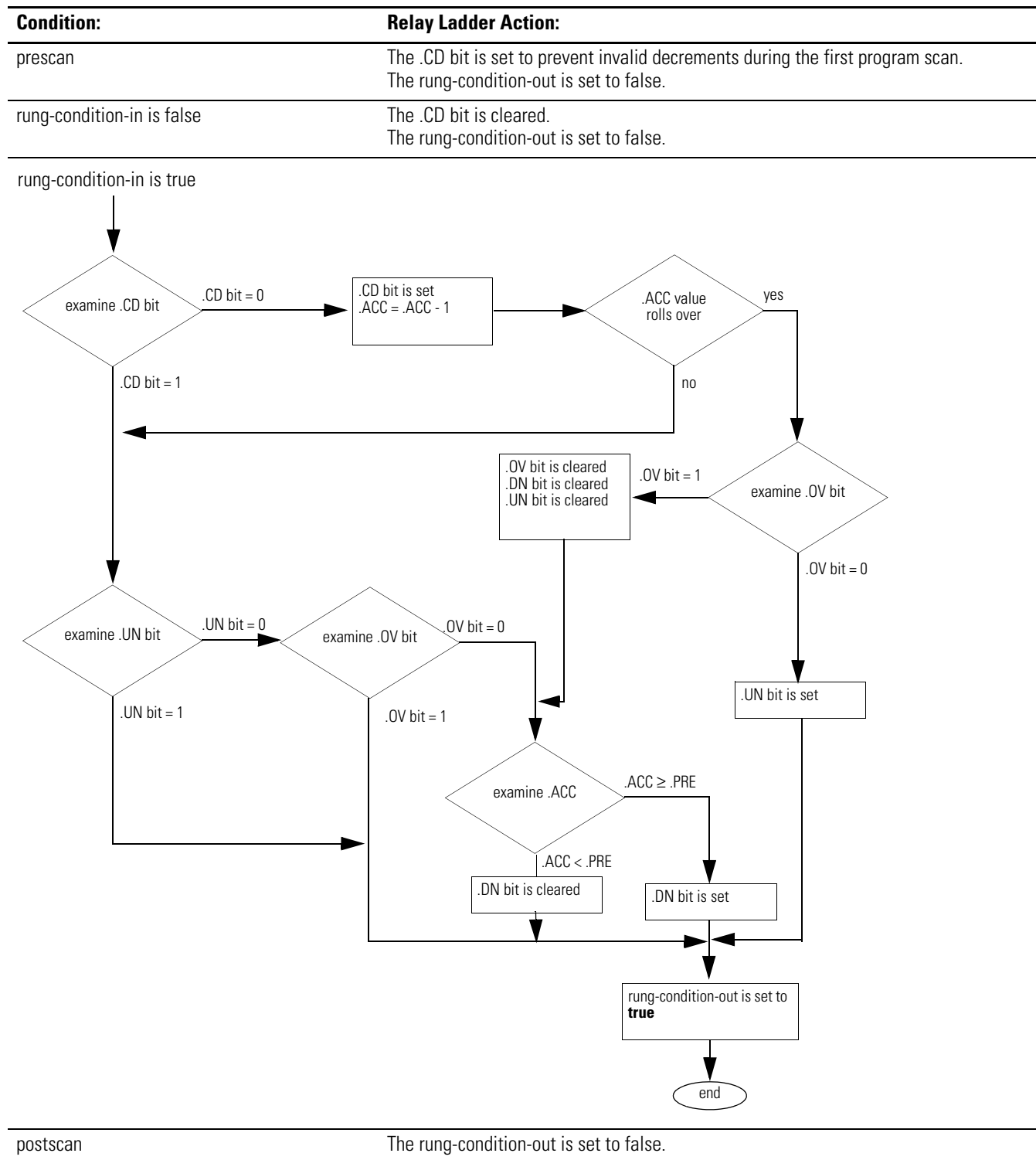


The accumulated value continues decrementing, even after the .DN bit is set. To clear the accumulated value, use a RES instruction that references the counter structure or write 0 to the accumulated value.

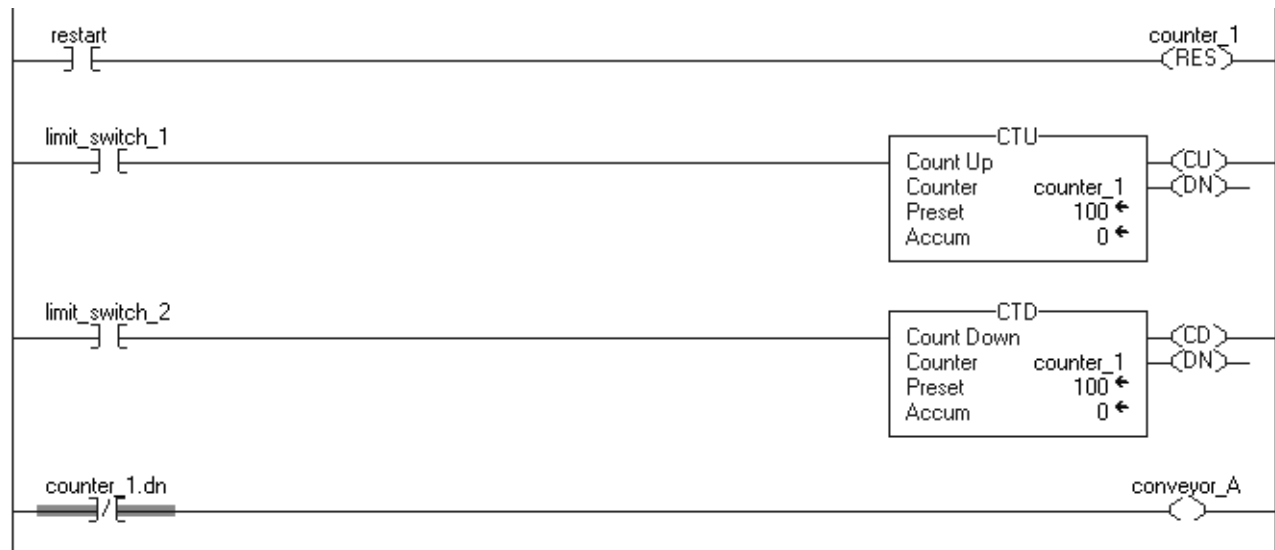
Arithmetic Status Flags: not affected

Fault Conditions: none

Execution:



Example: A conveyor brings parts into a buffer zone. Each time a part enters, *limit_switch_1* is enabled and *counter_1* increments by 1. Each time a part leaves, *limit_switch_2* is enabled and *counter_1* decrements by 1. If there are 100 parts in the buffer zone (*counter_1.dn* is set), *conveyor_a* turns on and stops the conveyor from bringing in any more parts until the buffer has room for more parts.



Count Up/Down (CTUD)

The CTUD instruction counts up by one when CUEnable transitions from clear to set. The instruction counts down by one when CDEnable transitions from clear to set.

This instruction is available in relay ladder as three separate instructions: CTU (see page 2-24), CTD (see page 2-28), and RES (see page 2-36).

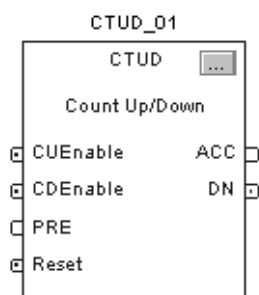
Operands:



CTUD (CTUD_tag) ;

Structured Text

Variable:	Type:	Format:	Description:
CTUD tag	FBD_COUNTER	structure	CTUD structure



Function Block

Operand:	Type:	Format:	Description:
CTUD tag	FBD_COUNTER	structure	CTUD structure

FBD_COUNTER Structure

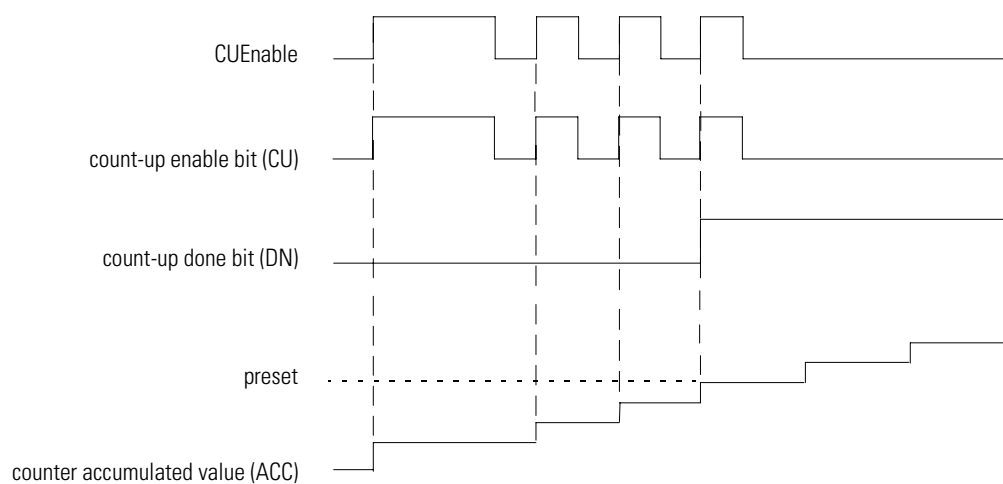
Input Parameter:	Data Type:	Description:
EnableIn	BOOL	Function Block: If cleared, the instruction does not execute and outputs are not updated. If set, the instruction executes. Default is set. Structured Text: No effect. The instruction executes.
CUEnable	BOOL	Enable up count. When input toggles from clear to set, accumulator counts up by one. Default is cleared.
CDEnable	BOOL	Enable down count. When input toggles from clear to set, accumulator counts down by one. Default is cleared.
PRE	DINT	Counter preset value. This is the value the accumulated value must reach before DN is set. Valid = any integer. Default is 0.
Reset	BOOL	Request to reset the timer. When set, the counter resets. Default is cleared.

Output Parameter:	Data Type:	Description:
EnableOut	BOOL	The instruction produced a valid result.
ACC	DINT	Accumulated value.
CU	BOOL	Count up enabled.
CD	BOOL	Count down enabled.
DN	BOOL	Counting done. Set when accumulated value is greater than or equal to preset.
OV	BOOL	Counter overflow. Indicates the counter exceeded the upper limit of 2,147,483,647. The counter then rolls over to -2,147,483,648 and begins counting down again.
UN	BOOL	Counter underflow. Indicates the counter exceeded the lower limit of -2,147,483,648. The counter then rolls over to 2,147,483,647 and begins counting down again.

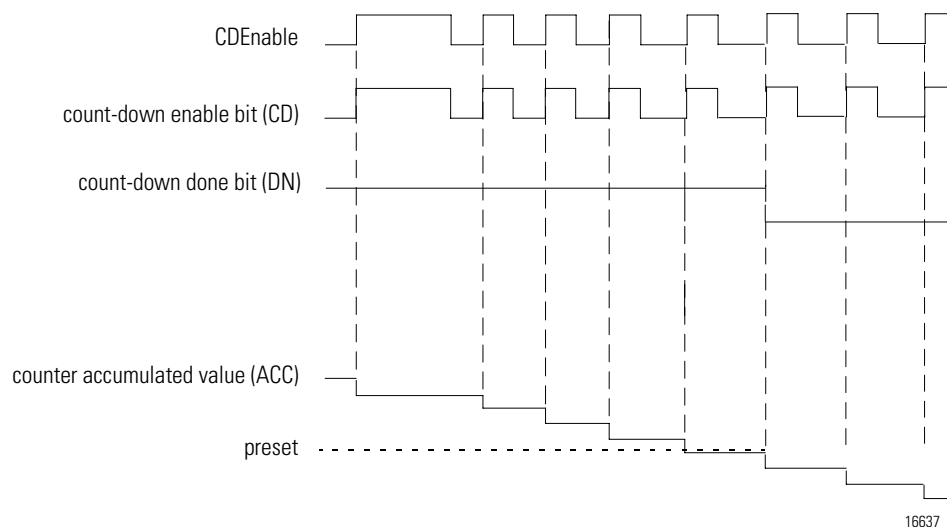
Description When enabled and CUEnable is set, the CTUD instructions increments the counter by one. When enabled and CDEnable is set, the CTUD instruction decrements the counter by one.

Both the CUEnable and CDEnable input parameters can both be toggled during the same scan. The instruction executes the count up prior to the count down.

Counting up



16636

Counting down

16637

When disabled, the CTUD instruction retains its accumulated value. Set the Reset input parameter of the FBD_COUNTER structure to reset the instruction.

Arithmetic Status Flags: not affected

Fault Conditions: none

Execution:

Condition:	Function Block Action:	Structured Text Action:
prescan	No initialization required.	No initialization required.
instruction first scan	CUEnable _{n-1} and CDEnable _{n-1} are set.	CUEnable _{n-1} and CDEnable _{n-1} are set.
instruction first run	CUEnable _{n-1} and CDEnable _{n-1} are set.	CUEnable _{n-1} and CDEnable _{n-1} are set.
EnableIn is cleared	EnableOut is cleared, the instruction does nothing, and the outputs are not updated.	na
EnableIn is set	The instruction sets CUEnable _{n-1} and CDenable _{n-1} . On a cleared to set transition of EnableIn: <ul style="list-style-type: none"> The instruction executes. EnableOut is set. 	The instruction sets CUEnable _{n-1} and CDenable _{n-1} . EnableIn is always set. The instruction executes.
reset	When set, the instruction clears CUEnable _{n-1} , CDenable _{n-1} , CU, CD, DN, OV, and UN and sets ACC = zero.	When set, the instruction clears CUEnable _{n-1} , CDenable _{n-1} , CU, CD, DN, OV, and UN and sets ACC = zero.
postscan	No action taken.	No action taken.

Example: When *limit_switch1* goes from cleared to set, CUEnable is set for one scan and the CTUD instruction increments the ACC value by 1. When $ACC \geq PRE$, the DN parameter is set, which enables the function block instruction following the CTUD instruction.

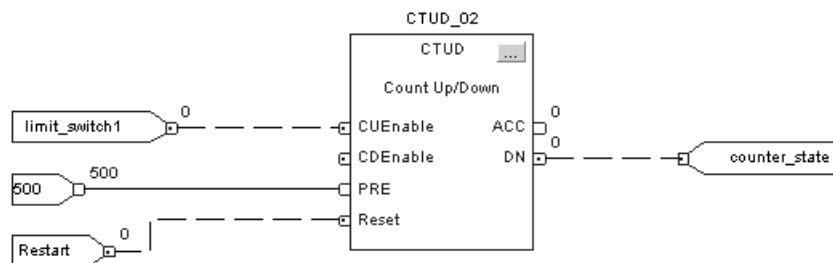
Structured Text

```
CTUD_01.Preset := 500;
CTUD_01.Reset := Restart;
CTUD_01.CUEnable := limit_switch1;

CTUD(CTUD_01);

counter_state := CTUD_01.DN;
```

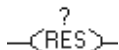
Function Block



Reset (RES)

The RES instruction resets a TIMER, COUNTER, or CONTROL structure.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
structure	TIMER CONTROL COUNTER	tag	structure to reset

Description: When enabled the RES instruction clears these elements:

When using a RES instruction for a:	The instruction clears:
TIMER	.ACC value control status bits
COUNTER	.ACC value control status bits
CONTROL	.POS value control status bits

ATTENTION



Because the RES instruction clears the .ACC value, .DN bit, and .TT bit, do not use the RES instruction to reset a TOF timer.

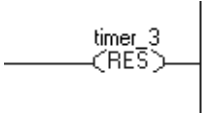
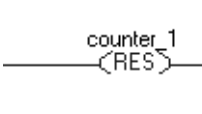
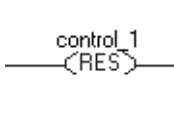
Arithmetic Status Flags: not affected

Fault Conditions: none

Execution:

Condition:	Relay Ladder Action:
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	The RES instruction resets the specified structure. The rung-condition-out is set to true.
postscan	The rung-condition-out is set to false.

Examples:

Example:	Description:
 <p>A horizontal line representing a normally open contact is connected to a vertical line. The contact is labeled 'timer_3' above it and '(RES)' below it.</p>	When enabled, reset <i>timer_3</i> .
 <p>A horizontal line representing a normally open contact is connected to a vertical line. The contact is labeled 'counter_1' above it and '(RES)' below it.</p>	When enabled, reset <i>counter_1</i> .
 <p>A horizontal line representing a normally open contact is connected to a vertical line. The contact is labeled 'control_1' above it and '(RES)' below it.</p>	When enabled, reset <i>control_1</i> .

Notes:

Input/Output Instructions

(MSG, GSV, SSV, IOT)

Introduction

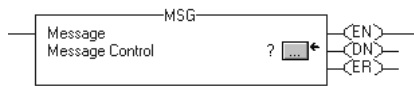
The input/output instructions read or write data to or from the controller or a block of data to or from another module on another network.

If you want to:	Use this instruction:	Available in these languages:	See page:
send data to or from another module	MSG	relay ladder structured text	3-2
get controller status information	GSV	relay ladder structured text	3-34
set controller status information	SSV	relay ladder structured text	3-34
<ul style="list-style-type: none">send output values to an I/O module or consuming controller at a specific point in your logictrigger an event task in another controller	IOT	relay ladder structured text	3-57

Message (MSG)

The MSG instruction asynchronously reads or writes a block of data to another module on a network.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Message control	MESSAGE	tag	message structure



```
MSG(MessageControl) ;
```

Structured Text

The operands are the same as those for the relay ladder MSG instruction.

MESSAGE Structure

Mnemonic:	Data Type:	Description:
.FLAGS	INT	The .FLAGS member provides access to the status members (bits) in one, 16-bit word.
		This bit: Is this member:
		2 .EW
		4 .ER
		5 .DN
		6 .ST
		7 .EN
		8 .TO
		9 .EN_CC
Important: Resetting any MSG status bits while a MSG is enabled can disrupt communications.		
.ERR	INT	If the .ER bit is set, the error code word identifies error codes for the MSG instruction.
.EXERR	INT	The extended error code word specifies additional error code information for some error codes.
.REQ_LEN	INT	The requested length specifies how many words the message instruction will attempt to transfer.
.DN_LEN	INT	The done length identifies how many words actually transferred.
.EW	BOOL	The enable waiting bit is set when the controller detects that a message request has entered the queue. The controller resets the.EW bit when the .ST bit is set.
.ER	BOOL	The error bit is set when the controller detects that a transfer failed. The .ER bit is reset the next time the rung-condition-in goes from false to true.
.DN	BOOL	The done bit is set when the last packet of the message is successfully transferred. The .DN bit is reset the next time the rung-condition-in goes from false to true.
.ST	BOOL	The start bit is set when the controller begins executing the MSG instruction. The .ST bit is reset when the .DN bit or the .ER bit is set.

Mnemonic:	Data Type:	Description:
.EN	BOOL	The enable bit is set when the rung-condition-in goes true and remains set until either the .DN bit or the .ER bit is set and the rung-condition-in is false. If the rung-condition-in goes false, but the .DN bit and the .ER bit are cleared, the .EN bit remains set.
.TO	BOOL	If you manually set the .TO bit, the controller stops processing the message and sets the .ER bit.
.EN_CC	BOOL	The enable cache bit determines how to manage the MSG connection. Refer to "Choose a cache option:" on page 3-31 Connections for MSG instructions going out the serial port are not cached, even if the .EN_CC bit is set.
.ERR_SRC	SINT	Used by RSLogix 5000 software to show the error path on the Message Configuration dialog box
.DestinationLink	INT	To change the Destination Link of a DH+ or CIP with Source ID message, set this member to the required value.
.DestinationNode	INT	To change the Destination Node of a DH+ or CIP with Source ID message, set this member to the required value.
.SourceLink	INT	To change the Source Link of a DH+ or CIP with Source ID message, set this member to the required value.
.Class	INT	To change the Class parameter of a CIP Generic message, set this member to the required value.
.Attribute	INT	To change the Attribute parameter of a CIP Generic message, set this member to the required value.
.Instance	DINT	To change the Instance parameter of a CIP Generic message, set this member to the required value.
.LocalIndex	DINT	If you use an asterisk [*] to designate the element number of the local array, the LocalIndex provides the element number. To change the element number, set this member to the required value.
		If the message: Then the local array is the:
		reads data Destination element
		writes data Source element
.Channel	SINT	To send the message out a different channel of the 1756-DHRIO module, set this member to the required value. Use either the ASCII character A or B.
.Rack	SINT	To change the rack number for a block transfer message, set this member to the required rack number (octal).
.Group	SINT	To change the group number for a block transfer message, set this member to the required group number (octal).
.Slot	SINT	To change the slot number for a block transfer message, set this member to the required slot number.
		If the message goes over this network: Then specify the slot number in:
		universal remote I/O octal
		ControlNet decimal (0-15)
.Path	STRING	To send the message to a different controller, set this member to the new path. <ul style="list-style-type: none">Enter the path as hexadecimal values.Omit commas [,] For example, for a path of 1, 0, 2, 42, 1, 3, enter \$01\$00\$02\$2A\$01\$03. To browse to a device and automatically create a portion or all of the new string, right-click a string tag and choose <i>Go to Message Path Editor</i> .

Mnemonic:	Data Type:	Description:						
.RemoteIndex	DINT	<p>If you use an asterisk [*] to designate the element number of the remote array, the RemoteIndex provides the element number. To change the element number, set this member to the required value.</p> <table><tr><th>If the message:</th><th>Then the remote array is the:</th></tr><tr><td>reads data</td><td>Source element</td></tr><tr><td>writes data</td><td>Destination element</td></tr></table>	If the message:	Then the remote array is the:	reads data	Source element	writes data	Destination element
If the message:	Then the remote array is the:							
reads data	Source element							
writes data	Destination element							
.RemoteElement	STRING	<p>To specify a different tag or address in the controller to which the message is sent, set this member to the required value. Enter the tag or address as ASCII characters.</p> <table><tr><th>If the message:</th><th>Then the remote array is the:</th></tr><tr><td>reads data</td><td>Source element</td></tr><tr><td>writes data</td><td>Destination element</td></tr></table>	If the message:	Then the remote array is the:	reads data	Source element	writes data	Destination element
If the message:	Then the remote array is the:							
reads data	Source element							
writes data	Destination element							
.UnconnectedTimeout	DINT	The time out for unconnected messages. The default value is 30 seconds.						
.ConnectionRate	DINT	<p>The ConnectionRate times the TimeoutMultiplier produces the time out for connected messages.</p> <ul style="list-style-type: none">• The default ConnectionRate is 7.5 seconds.• The default TimeoutMultiplier is 0 (which equates to a multiplication factor of 4).• The default time out for connected messages is 30 seconds (7.5 seconds x 4 = 30 seconds).• To change the time out, change the ConnectionRate and leave the TimeoutMultiplier at the default value.						
.TimeoutMultiplier	SINT							

ATTENTION

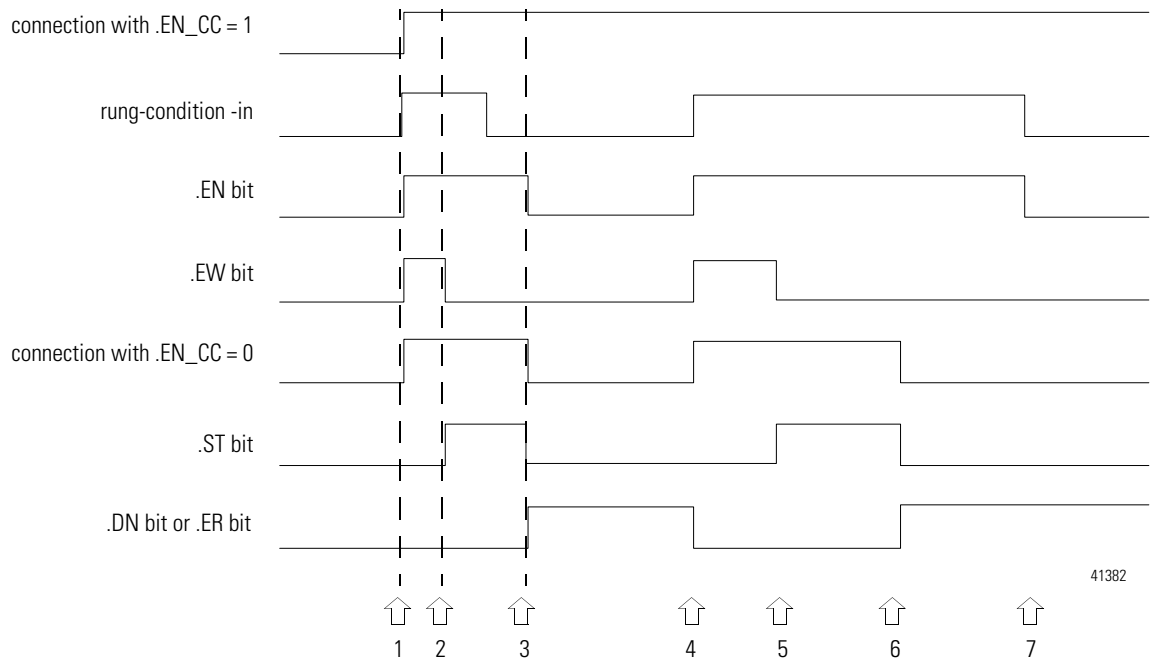
The controller processes the .ST, .EW, .DN and .ER bits asynchronously to the program scan. To examine these bits in ladder logic, copy the .FLAGS word to an INT tag and check the bits from there. Otherwise, timing problems might invalidate your application with possible damage to equipment and injury to personnel.

Description The MSG instruction transfers elements of data.

This is a transitional instruction:

- In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.
- In structured text, condition the instruction so that it only executes on a transition. See Appendix C.

The size of each element depends on the data types you specify and the type of message command you use.



Where:	Description:	Where:	Description:
1	rung-condition-in is true .EN is set .EW is set connection is opened*	5	message is sent .ST is set .EW is cleared
2	message is sent .ST is set .EW is cleared	6	message is done or errored rung-condition-in is still true .DN or .ER is set .ST is cleared connection is closed (if .EN_CC = 0)
3	message is done or errored rung-condition-in is false .DN or .ER is set .ST is cleared connection is closed (if .EN_CC = 0) .EN is cleared (rung-condition-in is false)	7	rung-condition-in goes false and .DN or .ER is set .EN is cleared
4	rung-condition-in is true .DN or .ER was previously set .EN is set .EW is set connection is opened* .DN or .ER is cleared		

Execution:

Condition:	Relay Ladder Action:	Structured Text Action:
prescan	The rung-condition-out is set to false.	No action taken.

rung-condition-in is false
(does not apply to structured text)

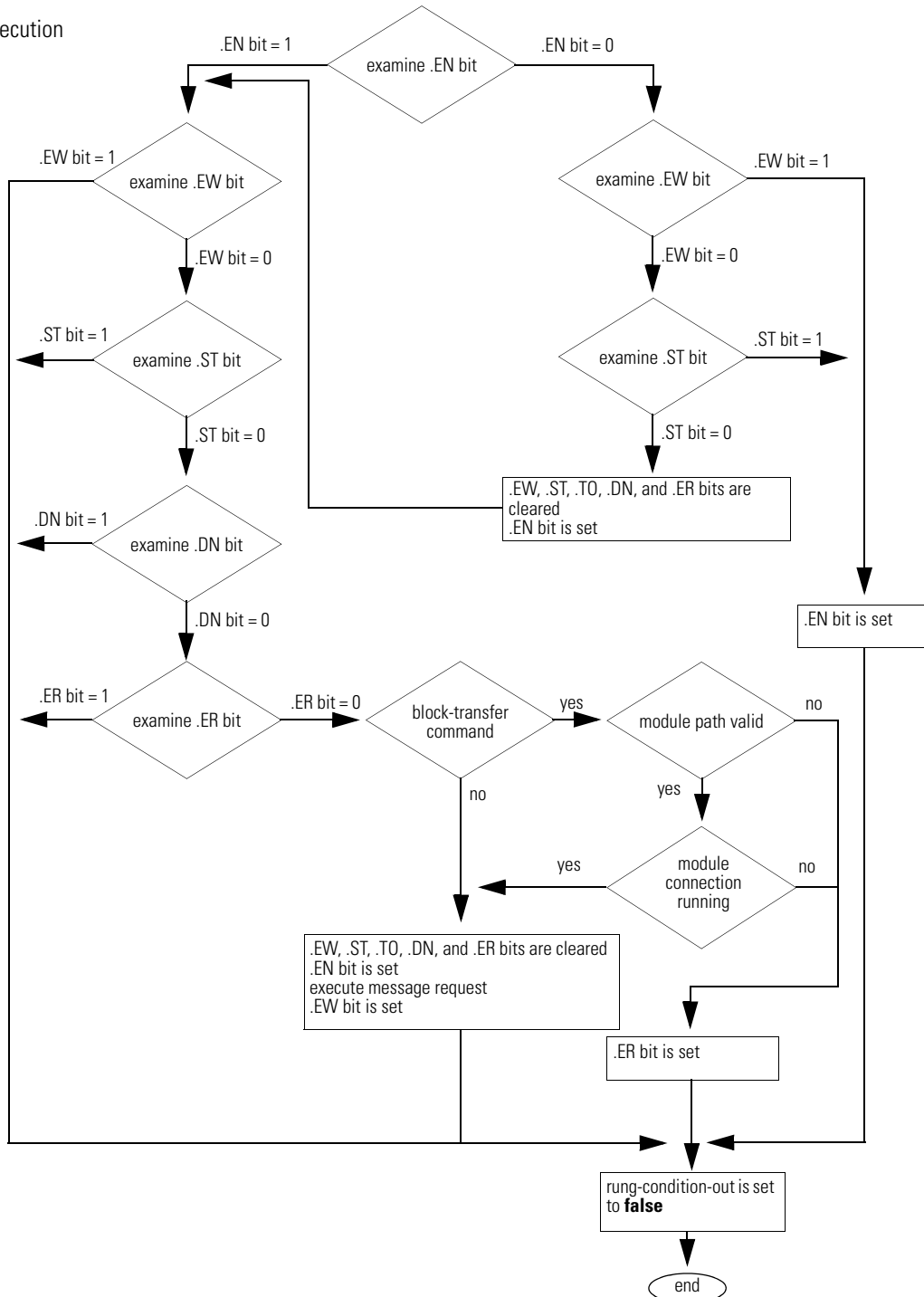
```

graph TD
    Start([start]) --> Prescan[Condition: prescan]
    Prescan --> ENBit{examine .EN bit}
    ENBit -- ".EN bit = 1" --> EWBit{examine .EW bit}
    ENBit -- ".EN bit = 0" --> DNBit1{examine .DN bit}
    EWBit -- ".EW bit = 1" --> STBit{examine .ST bit}
    EWBit -- ".EW bit = 0" --> DNBit1
    STBit -- ".ST bit = 1" --> DNBit1
    STBit -- ".ST bit = 0" --> DNBit2{examine .DN bit}
    DNBit1 -- ".DN bit = 1" --> ERBit1{examine .ER bit}
    DNBit1 -- ".DN bit = 0" --> DNBit2
    DNBit2 -- ".DN bit = 1" --> ENCleared[.EN bit is cleared]
    DNBit2 -- ".DN bit = 0" --> ERBit2{examine .ER bit}
    ERBit1 -- ".ER bit = 1" --> BTransfer{block-transfer command}
    ERBit1 -- ".ER bit = 0" --> BTransfer
    ERBit2 -- ".ER bit = 1" --> ENCleared
    ERBit2 -- ".ER bit = 0" --> BTransfer
    BTransfer -- "yes" --> PathValid{module path valid}
    BTransfer -- "no" --> ExecuteMsg[execute message request]
    PathValid -- "yes" --> ConnRunning{module connection running}
    PathValid -- "no" --> ExecuteMsg
    ConnRunning -- "yes" --> ERSet[.ER bit is set]
    ConnRunning -- "no" --> ExecuteMsg
    ExecuteMsg --> EWSet[.EW bit is set]
    ENCleared --> Junction(( ))
    ERSet --> Junction
    EWSet --> Junction
    Junction --> RungOutFalse[rung-condition-out is set to false]
    RungOutFalse --> End([end])
  
```

rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	na
---------------------------	---	----

Condition:	Relay Ladder Action:	Structured Text Action
EnableIn is set	na	EnableIn is always set. The instruction executes.

instruction execution



postscan

The rung-condition-out is set to false.

No action taken.

Arithmetic Status Flags: not affected

Fault Conditions: none

MSG Error Codes

The error codes depend on the type of MSG instruction.

Error Codes

RSLogix 5000 software does not always display the full description.

Error code (hex):	Description:	Display in software:
0001	Connection failure (see extended error codes)	same as description
0002	Insufficient resource	same as description
0003	Invalid value	same as description
0004	IOI syntax error (see extended error codes)	same as description
0005	Destination unknown, class unsupported, instance undefined or structure element undefined (see extended error codes)	same as description
0006	Insufficient packet space	same as description
0007	Connection lost	same as description
0008	Service unsupported	same as description
0009	Error in data segment or invalid attribute value	same as description
000A	Attribute list error	same as description
000B	State already exists	same as description
000C	Object model conflict	same as description
000D	Object already exists	same as description
000E	Attribute not settable	same as description
000F	Permission denied	same as description
0010	Device state conflict	same as description
0011	Reply will not fit	same as description
0012	Fragment primitive	same as description
0013	Insufficient command data	same as description
0014	Attribute not supported	same as description
0015	Too much data	same as description
001A	Bridge request too large	same as description
001B	Bridge response too large	same as description
001C	Attribute list shortage	same as description

Error code (hex):	Description:	Display in software:
001D	Invalid attribute list	same as description
001E	Embedded service error	same as description
001F	Connection related failure (see extended error codes)	same as description
0022	Invalid reply received	same as description
0025	Key segment error	same as description
0026	Invalid IOI error	same as description
0027	Unexpected attribute in list	same as description
0028	DeviceNet error - invalid member ID	same as description
0029	DeviceNet error - member not settable	same as description
00D1	Module not in run state	unknown error
00FB	Message port not supported	unknown error
00FC	Message unsupported data type	unknown error
00FD	Message uninitialized	unknown error
00FE	Message timeout	unknown error
00FF	General error (see extended error codes)	unknown error

Extended Error Codes

RSLogix 5000 software does not display any text for the extended error codes.

These are the extended error codes for error code **0001**.

Extended error code (hex):	Description:	Extended error code (hex):	Description:
0100	Connection in use	0203	Connection timeout
0103	Transport not supported	0204	Unconnected message timeout
0106	Ownership conflict	0205	Unconnected send parameter error
0107	Connection not found	0206	Message too large
0108	Invalid connection type	0301	No buffer memory
0109	Invalid connection size	0302	Bandwidth not available
0110	Module not configured	0303	No screeners available
0111	EPR not supported	0305	Signature match
0114	Wrong module	0311	Port not available
0115	Wrong device type	0312	Link address not available
0116	Wrong revision	0315	Invalid segment type
0118	Invalid configuration format	0317	Connection not scheduled
011A	Application out of connections		

These are the extended error codes for error code **001F**.

Extended error code (hex):	Description:
0203	Connection timeout

These are the extended error codes for error code **0004** and **0005**.

Extended error code (hex):	Description:
0000	extended status out of memory
0001	extended status out of instances

These are the extended error codes for error code **00FF**.

Extended error code (hex):	Description:	Extended error code (hex):	Description:
2001	Excessive IOI	2108	Controller in upload or download mode
2002	Bad parameter value	2109	Attempt to change number of array dimensions
2018	Semaphore reject	210A	Invalid symbol name
201B	Size too small	210B	Symbol does not exist
201C	Invalid size	210E	Search failed
2100	Privilege failure	210F	Task cannot start
2101	Invalid keyswitch position	2110	Unable to write
2102	Password invalid	2111	Unable to read
2103	No password issued	2112	Shared routine not editable
2104	Address out of range	2113	Controller in faulted mode
2105	Address and how many out of range	2114	Run mode inhibited
2106	Data in use		
2107	Type is invalid or not supported		

PLC and SLC Error Codes (.ERR)

Logix firmware revision 10.x and later provides new error codes for errors that are associated with PLC and SLC message types (PCCC messages).

- This change lets RSLogix 5000 software display a more meaningful description for many of the errors. Previously the software did not give a description for any of the errors associated with the 00F0 error code.
- The change also makes the error codes more consistent with errors returned by other controllers, such as PLC-5 controllers.

The following table shows the change in the error codes from R9.x and earlier to R10.x and later. As a result of the change, the .ERR member returns a unique value for each PCCC error. The .EXERR is no longer required for these errors.

Table 3.1 PLC and SLC Error Codes (hex)

R9.x and earlier		R10.x and later		Description:
.ERR	.EXERR	.ERR	.EXERR	
0010		1000		Illegal command or format from local processor
0020		2000		Communication module not working
0030		3000		Remote node is missing, disconnected, or shut down
0040		4000		Processor connected but faulted (hardware)
0050		5000		Wrong station number
0060		6000		Requested function is not available
0070		7000		Processor is in Program mode
0080		8000		Processor's compatibility file does not exist
0090		9000		Remote node cannot buffer command
00B0		B000		Processor is downloading so it is not accessible
00F0	0001	F001		Processor incorrectly converted the address
00F0	0002	F002		Incomplete address
00F0	0003	F003		Incorrect address
00F0	0004	F004		Illegal address format - symbol not found
00F0	0005	F005		Illegal address format - symbol has 0 or greater than the maximum number of characters supported by the device
00F0	0006	F006		Address file does not exist in target processor
00F0	0007	F007		Destination file is too small for the number of words requested
00F0	0008	F008		Cannot complete request Situation changed during multipacket operation

Table 3.1 PLC and SLC Error Codes (hex) (Continued)

R9.x and earlier		R10.x and later		Description:
.ERR	.EXERR	.ERR	.EXERR	
00F0	0009	F009		Data or file is too large Memory unavailable
00F0	000A	F00A		Target processor cannot put requested information in packets
00F0	000B	F00B		Privilege error; access denied
00F0	000C	F00C		Requested function is not available
00F0	000D	F00D		Request is redundant
00F0	000E	F00E		Command cannot be executed
00F0	000F	F00F		Overflow; histogram overflow
00F0	0010	F010		No access
00F0	0011	F011		Data type requested does not match data available
00F0	0012	F012		Incorrect command parameters
00F0	0013	F013		Address reference exists to deleted area
00F0	0014	F014		Command execution failure for unknown reason PLC-3 histogram overflow
00F0	0015	F015		Data conversion error
00F0	0016	F016		The scanner is not available to communicate with a 1771 rack adapter
00F0	0017	F017		The adapter is no available to communicate with the module
00F0	0018	F018		The 1771 module response was not valid
00F0	0019	F019		Duplicate label
00F0	001A	F01A		File owner active - the file is being used
00F0	001B	F01B		Program owner active - someone is downloading or editing online
00F0	001C	F01C		Disk file is write protected or otherwise not accessible (offline only)
00F0	001D	F01D		Disk file is being used by another application Update not performed (offline only)

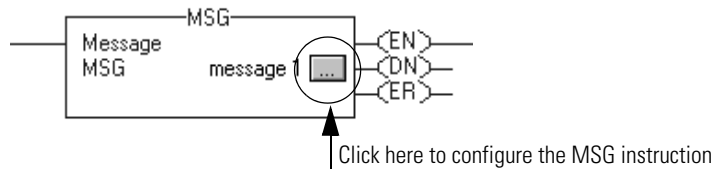
Block-Transfer Error Codes

These are the Logix5000 block-transfer specific error codes.

Error code (hex):	Description:	Display in software:
00D0	The scanner did not receive a block-transfer response from the block-transfer module within 3.5 seconds of the request	unknown error
00D1	The checksum from the read response did not match the checksum of the data stream	unknown error
00D2	The scanner requested either a read or write but the block-transfer module responded with the opposite	unknown error
00D3	The scanner requested a length and the block-transfer module responded with a different length	unknown error
00D6	The scanner received a response from the block-transfer module indicating the write request failed	unknown error
00EA	The scanner was not configured to communicate with the rack that would contain this block-transfer module	unknown error
00EB	The logical slot specified is not available for the given rack size	unknown error
00EC	There is currently a block-transfer request in progress and a response is required before another request can begin	unknown error
00ED	The size of the block-transfer request is not consistent with valid block-transfer size requests	unknown error
00EE	The type of block-transfer request is not consistent with the expected BT_READ or BT_WRITE	unknown error
00EF	The scanner was unable to find an available slot in the block-transfer table to accommodate the block-transfer request	unknown error
00F0	The scanner received a request to reset the remote I/O channels while there were outstanding block-transfers	unknown error
00F3	Queues for remote block-transfers are full	unknown error
00F5	No communication channels are configured for the requested rack or slot	unknown error
00F6	No communication channels are configured for remote I/O	unknown error
00F7	The block-transfer timeout, set in the instruction, timed out before completion	unknown error
00F8	Error in block-transfer protocol - unsolicited block-transfer	unknown error
00F9	Block-transfer data was lost due to a bad communication channel	unknown error
00FA	The block-transfer module requested a different length than the associated block-transfer instruction	unknown error
00FB	The checksum of the block-transfer read data was wrong	unknown error
00FC	There was an invalid transfer of block-transfer write data between the adapter and the block-transfer module	unknown error
00FD	The size of the block-transfer plus the size of the index in the block-transfer data table was greater than the size of the block-transfer data table file	unknown error

Specify the Configuration Details

After you enter the MSG instruction and specify the MESSAGE structure, use the Message Configuration dialog box to specify the details of the message.



The details you configure depend on the message type you select.

42976

If the target device is a:	Select one of these message types:	See page:
Logix5000 controller	CIP Data Table Read	3-16
	CIP Data Table Write	
I/O module that you configure using RSLogix 5000 software	Module Reconfigure	3-17
	CIP Generic	3-18
PLC-5 controller	PLC5 Typed Read	3-19
	PLC5 Typed Write	
	PLC5 Word Range Read	
	PLC5 Word Range Write	
SLC controller MicroLogix controller	SLC Typed Read	3-20
	SLC Typed Write	
Block-transfer module	Block-Transfer Read	3-21
	Block-Transfer Write	
PLC-3 processor	PLC3 typed read	3-22
	PLC3 typed write	
	PLC3 word range read	
	PLC3 word range write	
PLC-2 processor	PLC2 unprotected read	3-23
	PLC2 unprotected write	

You must specify this configuration information:

For this property:	Specify:
Source Element	<ul style="list-style-type: none"> If you select a read message type, the Source Element is the address of the data you want to read in the target device. Use the addressing syntax of the target device. If you select a write message type, the Source Tag is the first element of the tag that you want to send to the target device.
Number of Elements	The number of elements you read/write depends on the type of data you are using. An element refers to one "chunk" of related data. For example, tag <i>timer1</i> is one element that consists of one timer control structure.
Destination Element	<ul style="list-style-type: none"> If you select a read message type, the Destination Element is the first element of the tag in the Logix5000 controller where you want to store the data you read from the target device. If you select a write message type, the Destination Element is the address of the location in the target device where you want to write the data.

Specifying CIP Data Table Read and Write messages

The CIP Data Table Read and Write message types transfer data between Logix5000 controllers.

Select this command:	If you want to:
CIP Data Table Read	read data from another controller. The Source and Destination types must match.
CIP Data Table Write	write data to another controller. The Source and Destination types must match.

Reconfigure an I/O module

Use the Module Reconfigure message to send new configuration information to an I/O module. During the reconfiguration:

- Input modules continue to send input data to the controller.
- Output modules continue to controller their output devices.

A Module Reconfigure message requires this configuration properties:

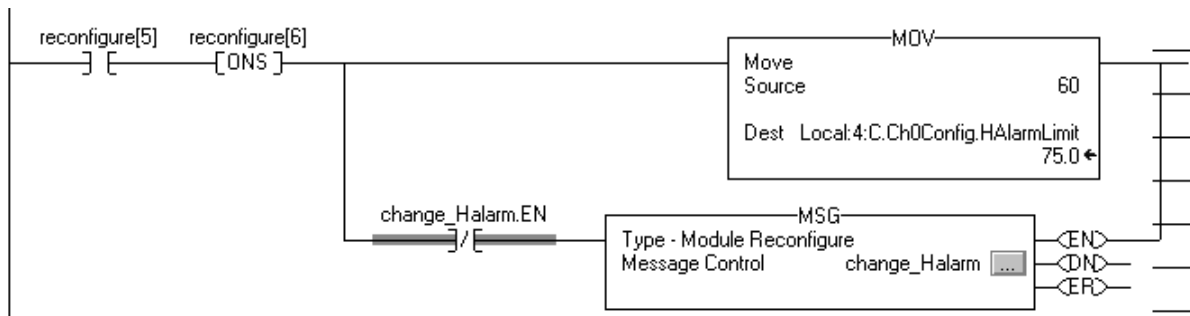
In this property:	Select:
Message Type	Module Reconfigure

Example: To reconfigure an I/O module:

1. Set the required member of the configuration tag of the module to the new value.
2. Send a Module Reconfigure message to the module.

When *reconfigure[5]* is set, set the high alarm to 60 for the local module in slot 4. The Module Reconfigure message then sends the new alarm value to the module. The one shot instruction prevents the rung from sending multiple messages to the module while the *reconfigure[5]* is on.

Relay Ladder



Structured Text

```
IF reconfigure[5] AND NOT reconfigure[6] THEN
    Local:4:C.Ch0Config.HAlarmLimit := 60;
    IF NOT change_Halarm.EN THEN
        MSG(change_Halarm);
    END_IF;
END_IF;
reconfigure[6] := reconfigure[5];
```

Specify CIP Generic messages

A CIP Generic message performs a specific action on an I/O module.

If you want to	In this property:	Type or select:
Perform a pulse test on a digital output module	Message Type	CIP Generic
	Service Type	Pulse Test
	Source	<i>tag_name</i> of type INT [5]
		This array contains:
	<i>tag_name</i> [0]	bit mask of points to test (test only one point at a time)
	<i>tag_name</i> [1]	reserved, leave 0
	<i>tag_name</i> [2]	pulse width (hundreds of μ secs, usually 20)
	<i>tag_name</i> [3]	zero cross delay for ControlLogix I/O (hundreds of μ secs, usually 40)
	<i>tag_name</i> [4]	verify delay
	Destination	leave blank
Reset electronic fuses on a digital output module	Message Type	CIP Generic
	Service Type	Reset Electronic Fuse
	Source	<i>tag_name</i> of type DINT This tag represents a bit mask of the points to reset fuses on.
	Destination	leave blank
Reset latched diagnostics on a digital input module	Message Type	CIP Generic
	Service Type	Reset Latched Diagnostics (I)
	Source	<i>tag_name</i> of type DINT This tag represents a bit mask of the points to reset diagnostics on.
Reset latched diagnostics on a digital output module	Message Type	CIP Generic
	Service Type	Reset Latched Diagnostics (O)
	Source	<i>tag_name</i> of type DINT This tag represents a bit mask of the points to reset diagnostics on.
Unlatch the alarm of an analog input module	Message Type	CIP Generic
	Service Type	Select which alarm that you want to unlatch: <ul style="list-style-type: none"> • Unlatch All Alarms (I) • Unlatch Analog High Alarm (I) • Unlatch Analog High High Alarm (I) • Unlatch Analog Low Alarm (I) • Unlatch Analog Low Low Alarm (I) • Unlatch Rate Alarm (I)
	Instance	Channel of the alarm that you want to unlatch

If you want to	In this property:	Type or select:
Unlatch the alarm of an analog output module	Message Type	CIP Generic
	Service Type	Select which alarm that you want to unlatch: <ul style="list-style-type: none"> • Unlatch All Alarms (0) • Unlatch High Alarm (0) • Unlatch Low Alarm (0) • Unlatch Ramp Alarm (0)
	Instance	Channel of the alarm that you want to unlatch

Specifying PLC-5 messages

Use the PLC-5 message types to communicate with PLC-5 controllers.

Select this command:	If you want to:
PLC5 Typed Read	Read 16-bit integer, floating-point, or string type data and maintain data integrity. See Table 3.2 on page 3-19.
PLC5 Typed Write	Write 16-bit integer, floating-point, or string type data and maintain data integrity. See Table 3.2 on page 3-19.
PLC5 Word Range Read	Read a contiguous range of 16-bit words in PLC-5 memory regardless of data type. This command starts at the address specified as the Source Element and reads sequentially the number of 16-bit words requested. The data from the Source Element is stored, starting at the address specified as the Destination Tag.
PLC5 Word Range Write	Write a contiguous range of 16-bit words from Logix5000 memory regardless of data type to PLC-5 memory. This command starts at the address specified as the Source Tag and reads sequentially the number of 16-bit words requested. The data from the Source Tag is stored, starting at the address specified as the Destination Element in the PLC-5 processor.

The following table shows the data types to use with PLC5 Typed Read and PLC5 Typed Write messages.

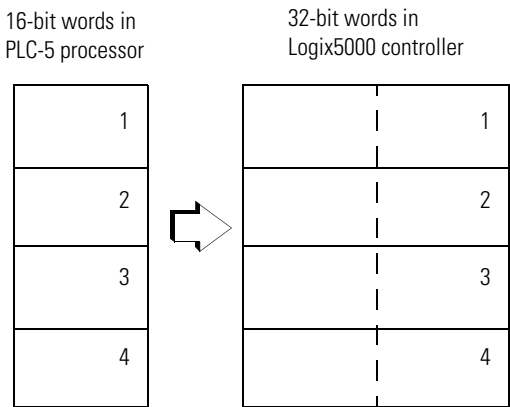
Table 3.2 Data types for PLC5 Typed Read and Typed Write messages

For this PLC-5 data type:	Use this Logix5000 data type:
B	INT
F	REAL
N	INT DINT (Only write DINT values to a PLC-5 controller if the value is $\geq -32,768$ and $\leq 32,767$.)
S	INT
ST	STRING

The Typed Read and Typed Write commands also work with SLC 5/03 processors (OS303 and above), SLC 5/04 processors (OS402 and above), and SLC 5/05 processors.

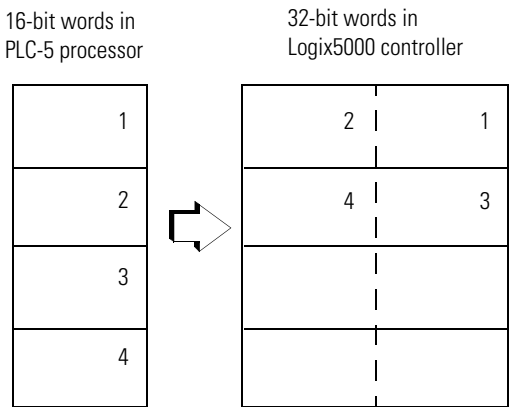
The following diagrams show how the typed and word-range commands differ. The example uses read commands from a PLC-5 processor to a Logix5000 controller.

Typed read command



The typed commands maintain data structure and value.

Word-range read command



The word-range commands fill the destination tag contiguously. Data structure and value change depending on the destination data type.

Specifying SLC messages

Use the SLC message types to communicate with SLC and MicroLogix controllers. The following table shows which data types that the instruction lets you access. The table also shows the corresponding Logix5000 data type.

For this SLC or MicroLogix data type:	Use this Logix5000 data type:
F	REAL
L (MicroLogix 1200 and 1500 controllers)	DINT
N	INT

Specify block-transfer messages

The block-transfer message types are used to communicate with block-transfer modules over a Universal Remote I/O network.

If you want to:	Select this command:
read data from a block-transfer module. This message type replaces the BTR instruction.	Block-Transfer Read
write data to a block-transfer module. This message type replaces the BTW instruction.	Block-Transfer Write

To configure a block-transfer message, follow these guidelines:

- The source (for BTW) and destination (for BTR) tags must be large enough to accept the requested data, except for MESSAGE, AXIS, and MODULE structures.
- Specify how many 16-bit integers (INT) to send or receive. You can specify from 0 to 64 integers.

If you want the:	Then specify:
Block-transfer module to determine how many 16-bit integers to send (BTR).	0 for the number of elements
Controller to send 64 integers (BTW).	

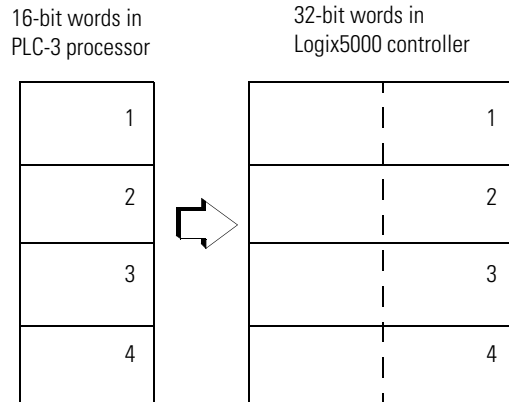
Specifying PLC-3 messages

The PLC-3 message types are designed for PLC-3 processors.

Select this command:	If you want to:
PLC3 Typed Read	<p>read integer or REAL type data.</p> <p>For integers, this command reads 16-bit integers from the PLC-3 processor and stores them in SINT, INT, or DINT data arrays in the Logix5000 controller and maintains data integrity.</p> <p>This command also reads floating-point data from the PLC-3 and stores it in a REAL data type tag in the Logix5000 controller.</p>
PLC3 Typed Write	<p>write integer or REAL type data.</p> <p>This command writes SINT or INT data, to the PLC-3 integer file and maintains data integrity. You can write DINT data as long as it fits within an INT data type ($-32,768 \leq \text{data} \leq 32,767$).</p> <p>This command also writes REAL type data from the Logix5000 controller to a PLC-3 floating-point file.</p>
PLC3 Word Range Read	<p>read a contiguous range of 16-bit words in PLC-3 memory regardless of data type.</p> <p>This command starts at the address specified as the Source Element and reads sequentially the number of 16-bit words requested.</p> <p>The data from the Source Element is stored, starting at the address specified as the Destination Tag.</p>
PLC3 Word Range Write	<p>write a contiguous range of 16-bit words from Logix5000 memory regardless of data type to PLC-3 memory.</p> <p>This command starts at the address specified as the Source Tag and reads sequentially the number of 16-bit words requested.</p> <p>The data from the Source Tag is stored, starting at the address specified as the Destination Element in the PLC-3 processor.</p>

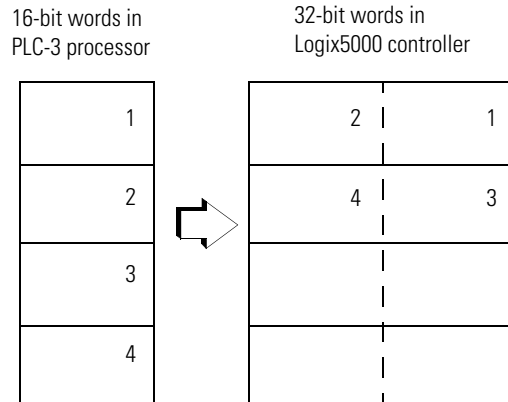
The following diagrams show how the typed and word-range commands differ. The example uses read commands from a PLC-3 processor to a Logix5000 controller.

Typed read command



The typed commands maintain data structure and value.

Word-range read command



The word-range commands fill the destination tag contiguously. Data structure and value change depending on the destination data type.

Specifying PLC-2 messages

The PLC-2 message types are designed for PLC-2 processors.

Select this command:	If you want to:
PLC2 Unprotected Read	read 16-bit words from any area of the PLC-2 data table or the PLC-2 compatibility file of another processor.
PLC2 Unprotected Write	write 16-bit words to any area of the PLC-2 data table or the PLC-2 compatibility file of another processor.

The message transfer uses 16-bit words, so make sure the Logix5000 tag appropriately stores the transferred data (typically as an INT array).

MSG Configuration Examples

The following examples show source and destination tags and elements for different controller combinations.

For MSG instructions originating from a Logix5000 controller and writing to another controller:

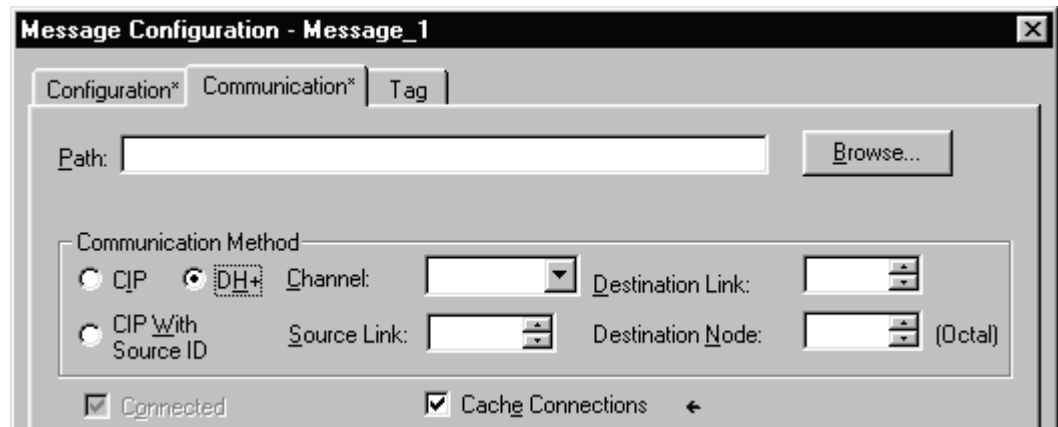
Message Path:	Example Source and Destination:	
Logix5000 → Logix5000	source tag	<i>array_1[0]</i>
	destination tag	<i>array_2[0]</i>
	You can use an alias tag for the source tag (in originating Logix5000 controller). You cannot use an alias for the destination tag. The destination must be a base tag.	
Logix5000 → PLC-5 Logix5000 → SLC	source tag	<i>array_1[0]</i>
	destination element	<i>N7:10</i>
	You can use an alias tag for the source tag (in originating Logix5000 controller).	
Logix5000 → PLC-2	source tag	<i>array_1[0]</i>
	destination element	<i>010</i>

For MSG instructions originating from a Logix5000 controller and reading from another controller:

Message Path:	Example Source and Destination:	
Logix5000 → Logix5000	source tag	<i>array_1[0]</i>
	destination tag	<i>array_2[0]</i>
	You cannot use an alias tag for the source tag. The source must be a base tag. You can use an alias tag for the destination tag (in originating Logix5000 controller).	
Logix5000 → PLC-5 Logix5000 → SLC	source element	<i>N7:10</i>
	destination tag	<i>array_1[0]</i>
	You can use an alias tag for the destination tag (in originating Logix5000 controller).	
Logix5000 → PLC-2	source element	<i>010</i>
	destination tag	<i>array_1[0]</i>

Specify the Communication Details

When you configure a MSG instruction, you specify these details on the Communication tab.

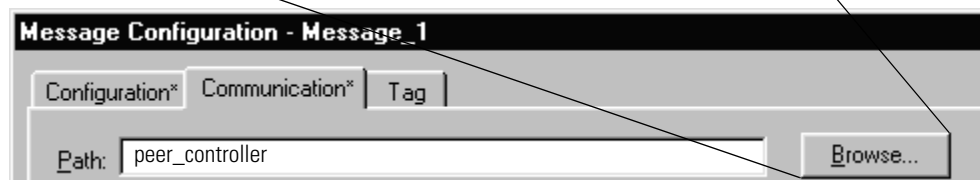
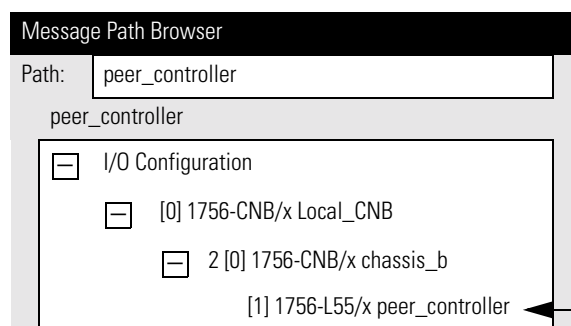


43008

Specify a path

The path describes the route that the message takes to get to the destination.

- If you add the local communication module, remote communication module, and the destination controller or device to the I/O configuration of the controller, the *Browse* button lets you select the destination.



- Some remote communication modules or devices are unavailable to the I/O configuration of the controller. For those situations, complete the path as follows:

1. Use the *Browse* button to select the local communication module.

2. In the *Path* text box, type the port from which the message exits the module.

3. Type the address of the next module along the path to the destination.

4. Type additional port and address combinations, if needed.

local_module, port, address, port, address

Where:	For this:	Is:
<i>port</i>	backplane from any 1756 controller or module	1
	DF1 port from a Logix5000 controller	2
	ControlNet port from a 1756-CNB module	
	Ethernet port from a 1756-ENBx or -ENET module	
	DH+ port over channel A from a 1756-DHRIO module	
	DH+ port over channel B from a 1756-DHRIO module	3
<i>address</i>	ControlLogix backplane	slot number
	DF1 network	station address (0-254)
	ControlNet network	node number (1-99 decimal)
	DH+ network	8# followed by the node number (1-77 octal) For example, to specify the octal node address of 37, type 8#37.
	EtherNet/IP network	You can specify a module on an EtherNet/IP network using any of these formats: IP address (e.g., 130.130.130.5) IP address:Port (e.g., 130.130.130.5:24) DNS name (e.g., tanks) DNS name:Port (e.g., tanks:24)

- For block transfer messages, add the following modules to the I/O configuration of the controller:

For block-transfers over this network: Add these modules to the I/O configuration:

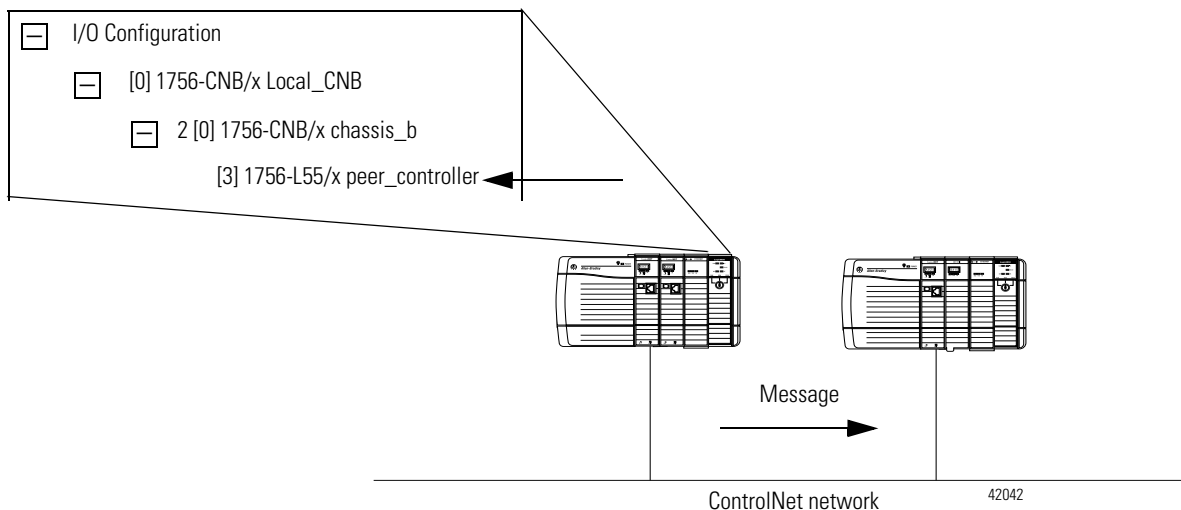
ControlNet	<ul style="list-style-type: none"> • local communication module (e.g., 1756-CNB module) • remote adapter module (e.g., 1771-ACN module)
universal remote I/O	<ul style="list-style-type: none"> • local communication module (e.g., 1756-DHRIO module) • one remote adapter module (e.g., 1771-ASB module) for each rack, or portion of a rack, in the chassis • block-transfer module (optional)

The following pages show example paths:

- over ControlNet, page 3-27
- over EtherNet/IP, page 3-28
- for a DH+ message, page 3-29

EXAMPLE

Specify a path over ControlNet

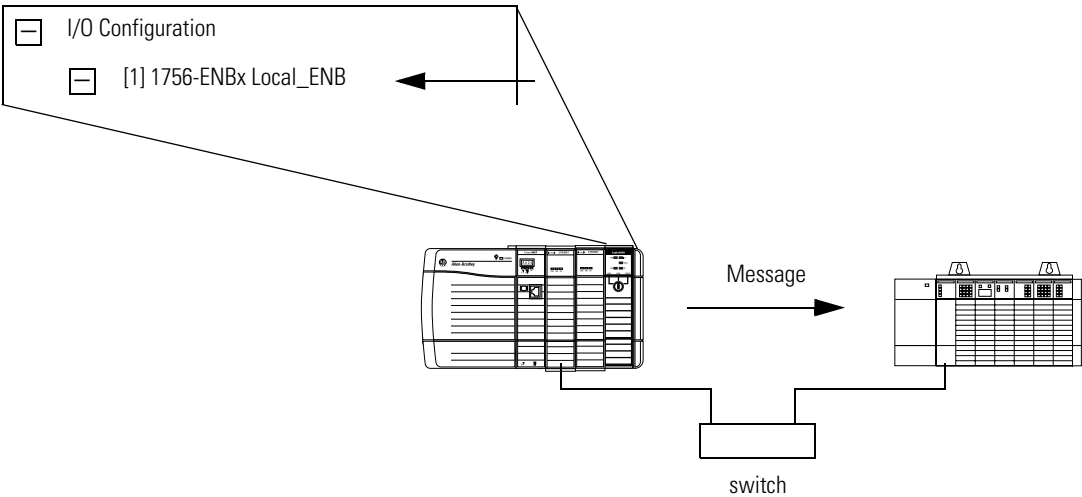


Path: peer_controller

where:

peer_controller is the name of the controller that receives the message.

EXAMPLE Specify a path over EtherNet/IP

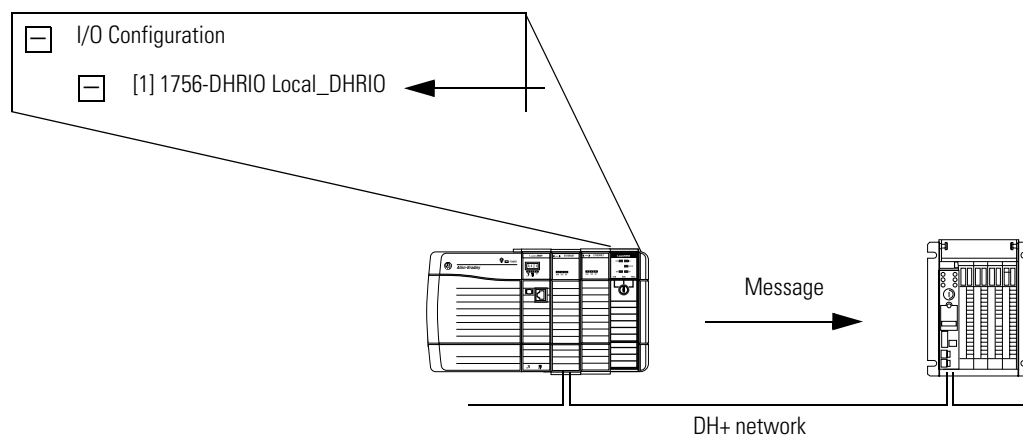


Path: Local_ENB,2,127.127.127.12

Where:	Is the:
Local_ENB	name of the 1756-ENBx module in the local chassis
2	Ethernet port of the 1756-ENBx module in the local chassis
127.127.127.12	IP address of the SLC 5/05 controller

EXAMPLE

Specify a path over DH+



Path: Local_DHRIO

where:

Local_DHRIO is the name in the 1756-DHRIO module in the same chassis as the controller that sends the message.

Specify a communication method or module address:

Use the following table to select a communication method or module address for the message.

If the destination device is a:	Then select:	And specify:	
Logix5000 controller	CIP	no other specifications required	
PLC-5 controller over an EtherNet/IP network			
PLC-5 controller over a ControlNet network			
SLC 5/05 controller			
PLC-5 controller over a DH+ network	DH+	Channel:	Channel A or B of the 1756-DHRIO module that is connected to the DH+ network
SLC controller over a DH+ network		Source Link:	Link ID assigned to the backplane of the controller in the routing table of the 1756-DHRIO module. (The source node in the routing table is automatically the slot number of the controller.)
PLC-3 processor		Destination Link	Link ID of the remote DH+ link where the target device resides
PLC-2 processor		Destination Node:	Station address of the target device, in octal
		If there is only one DH+ link and you did not use the RSLinx software to configure the DH/RIO module for remote links, specify 0 for both the Source Link and the Destination Link.	
Application on a workstation that is receiving an unsolicited message routed over an EtherNet/IP or ControlNet network through RSLinx	CIP with Source ID (This lets the application receive data from a controller.)	Source Link:	Remote ID of the topic in RSLinx software
		Destination Link:	Virtual Link ID set up in RSLinx (0-65535)
		Destination Node:	Destination ID (0-77 octal) provided by the application to RSLinx. For a DDE topic in RSLinx, use 77.
		The slot number of the ControlLogix controller is used as the Source Node.	
block transfer module over a universal remote I/O network	RIO	Channel:	Channel A or B of the 1756-DHRIO module that is connected to the RIO network
		Rack	Rack number (octal) of the module
		Group	Group number of the module
		Slot	Slot number that the module is in
block transfer module over a ControlNet network	ControlNet	Slot	Slot number that the module is in

Choose a cache option:

Depending on how you configure a MSG instruction, it may use a connection to send or receive data.

This type of message:	And this communication method:	Uses a connection:
CIP data table read or write	→	✓
PLC2, PLC3, PLC5, or SLC (all types)	CIP	
	CIP with Source ID	
	DH+	✓
CIP generic	→	your option ⁽¹⁾
block-transfer read or write	→	✓

⁽¹⁾ You can connect CIP generic messages. But for most applications we recommend you leave CIP generic messages unconnected.

If a MSG instruction uses a connection, you have the option to leave the connection open (cache) or close the connection when the message is done transmitting.

If you:	Then:
Cache the connection	The connection stays open after the MSG instruction is done. This optimizes execution time. Opening a connection each time the message executes increases execution time.
Do not cache the connection	The connection closes after the MSG instruction is done. This frees up that connection for other uses.

The controller has the following limits on the number of connections that you can cache:

If you have this software and firmware revision:	Then you can cache:
11.x or earlier	<ul style="list-style-type: none"> • block transfer messages for up to 16 connections • other types of messages for up to 16 connections
12.x or later	up to 32 connections

If several messages go to the same device, the messages may be able to share a connection.

If the MSG instructions are to:	And they are:	Then:
different devices	—————▶	Each MSG instruction uses 1 connection.
same device	enabled at the same time	Each MSG instruction uses 1 connection.
	NOT enabled at the same time	The MSG instructions share the connection. (I.e., Together they count as 1 connection.)

EXAMPLE

Share a Connection

If the controller alternates between sending a block-transfer read message and a block-transfer write message to the same module, then together both messages count as 1 connection. Caching both messages counts as 1 on the cache list.

Guidelines

As you plan and program your MSG instructions, follow these guidelines:

Guideline:	Details:
1. For each MSG instruction, create a control tag.	Each MSG instruction requires its own control tag. <ul style="list-style-type: none"> • Data type = MESSAGE • Scope = controller • The tag <i>cannot</i> be part of an array or a user-defined data type.
2. Keep the source and/or destination data at the controller scope.	A MSG instruction can access only tags that are in the Controller Tags folder (controller scope).
3. If your MSG is to a device that uses 16-bit integers, use a buffer of INTs in the MSG and DINTs throughout the project.	<p>If your message is to a device that uses 16-bit integers, such as a PLC-5® or SLC 500™ controller, and it transfers integers (not REALs), use a buffer of INTs in the message and DINTs throughout the project.</p> <p>This increases the efficiency of your project because Logix controllers execute more efficiently and use less memory when working with 32-bit integers (DINTs).</p> <p>To convert between INTs and DINTs, see <i>Logix5000 Controllers Common Procedures</i>, publication 1756-PM001.</p>
4. Cache the connected MSGs that execute most frequently.	<p>Cache the connection for those MSG instructions that execute most frequently, up to the maximum number permissible for your controller revision.</p> <p>This optimizes execution time because the controller does not have to open a connection each time the message executes.</p>
5. If you want to enable more than 16 MSGs at one time, use some type of management strategy.	<p>If you enable more than 16 MSGs at one time, some MSG instructions may experience delays in entering the queue. To guarantee the execution of each message, use one of these options:</p> <ul style="list-style-type: none"> • Enable each message in sequence. • Enable the messages in groups. • Program a message to communicate with multiple devices. For more information, see <i>Logix5000 Controllers Common Procedures</i>, publication 1756-PM001. • Program logic to coordinate the execution of messages. For more information, see <i>Logix5000 Controllers Common Procedures</i>, publication 1756-PM001.
6. Keep the number of unconnected and uncached MSGs less than the number of unconnected buffers.	<p>The controller can have 10 - 40 unconnected buffers. The default number is 10.</p> <ul style="list-style-type: none"> • If all the unconnected buffers are in use when an instruction leaves the message queue, the instruction errors and does not transfer the data. • You can increase the number of unconnected buffers (40 max.), but continue to follow guideline 5. • To increase the number of unconnected buffers, see <i>Logix5000 Controllers Common Procedures</i>, publication 1756-PM001.

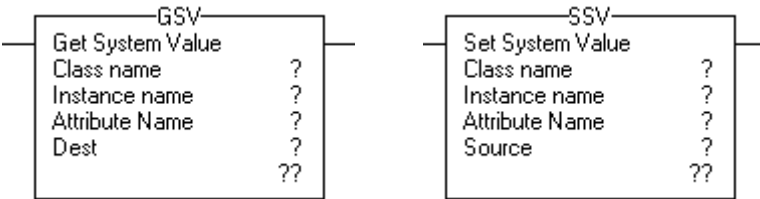
Get System Value (GSV) and Set System Value (SSV)

The GSV/SSV instructions get and set controller system data that is stored in objects.

Operands:



Relay Ladder



Operand:	Type:	Format:	Description:
Class name		name	name of object
Instance name		name	name of specific object, when object requires name
Attribute Name		name	attribute of object data type depends on the attribute you select
Destination (GSV)	SINT INT DINT REAL	tag	destination for attribute data
Source (SSV)	SINT INT DINT REAL	tag	tag that contains data you want to copy to the attribute



Structured Text

```
GSV (ClassName, InstanceName, AttributeName, Dest) ;  
SSV (ClassName, InstanceName, AttributeName, Source) ;
```

The operands for are the same as those for the relay ladder GSV and SSV instructions.

Description: The GSV/SSV instructions get and set controller system data that is stored in objects. The controller stores system data in objects. There is no status file, as in the PLC-5 processor.

When enabled, the GSV instruction retrieves the specified information and places it in the destination. When enabled, the SSV instruction sets the specified attribute with data from the source.

When you enter a GSV/SSV instruction, the programming software displays the valid object classes, object names, and attribute names for each instruction. For the GSV instruction, you can get values for all the available attributes. For the SSV instruction, the software displays only those attributes are allowed to set (SSV).

ATTENTION

Use the GSV/SSV instructions carefully. Making changes to objects can cause unexpected controller operation or injury to personnel.

If the size of the Source or Destination is too small, the instruction does not execute and a minor fault is logged. The following section, *GSV/SSV Objects*, defines each object's attributes and their associated data types. For example, the MajorFaultRecord attribute of the Program object requires a DINT[11] data type.

Arithmetic Status Flags: not affected

Fault Conditions:

A minor fault will occur if:	Fault type:	Fault code:
invalid object address	4	5
specified an object that does not support GSV/SSV	4	6
invalid attribute	4	6
did not supply enough information for an SSV instruction	4	6
the GSV destination was not large enough to hold the requested data	4	7

Execution:

Condition:	Relay Ladder Action:	Structured Text Action:
prescan	The rung-condition-out is set to false.	No action taken
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	na
EnableIn is set	na	EnableIn is always set. The instruction executes.
instruction executes	Get or set the specified value.	Get or set the specified value.
postscan	The rung-condition-out is set to false.	No action taken.

GSV/SSV Objects

When you enter a GSV/SSV instruction, you specify the object and its attribute that you want to access. In some cases, there will be more than one instance of the same type of object, so you might also have to specify the object name. For example, there can be several tasks in your application. Each task has its own TASK object that you access by the task name.

ATTENTION



For the GSV instruction, only the specified size of data is copied to the destination. For example, if the attribute is specified as a SINT and the destination is a DINT, only the lower 8 bits of the DINT destination are updated, leaving the remaining 24 bits unchanged.

You can access these objects:

For information about this object:	See this page or publication:
AXIS	<i>ControlLogix Motion Module Setup and Configuration Manual</i> , publication 1756-UM006
CONTROLLER	3-37
CONTROLLERDEVICE	3-37
CST	3-39
DF1	3-40
FAULTLOG	3-43
MESSAGE	3-44
MODULE	3-46
MOTIONGROUP	3-47
PROGRAM	3-48
ROUTINE	3-49
SERIALPORT	3-49
TASK	3-51
WALLCLOCKTIME	3-53

Accessing the CONTROLLER object

The CONTROLLER object provides status information about a controller's execution.

Attribute:	Data Type:	Instruction:	Description:
TimeSlice	INT	GSV SSV	Percentage of available CPU that is assigned to communications. Valid values are 10-90. This value cannot be changed when the controller keyswitch is in the run position.

Accessing the CONTROLLERDEVICE object

The CONTROLLERDEVICE object identifies the physical hardware of the controller.

Attribute:	Data Type:	Instruction:	Description:																												
DeviceName	SINT[33]	GSV	ASCII string that identifies the catalog number of the controller and memory board. The first byte contains a count of the number of ASCII characters returned in the array string.																												
ProductCode	INT	GSV	Identifies the type of controller <table><tr><th>Logix controller:</th><th>Product code:</th></tr><tr><td>CompactLogix5320</td><td>43</td></tr><tr><td>CompactLogix5330</td><td>44</td></tr><tr><td>CompactLogix5335E</td><td>65</td></tr><tr><td>ControlLogix5550</td><td>3</td></tr><tr><td>ControlLogix5553</td><td>50</td></tr><tr><td>ControlLogix5555</td><td>51</td></tr><tr><td>ControlLogix5561</td><td>54</td></tr><tr><td>ControlLogix5562</td><td>55</td></tr><tr><td>ControlLogix5563</td><td>56</td></tr><tr><td>DriveLogix5720</td><td>48</td></tr><tr><td>FlexLogix5433</td><td>41</td></tr><tr><td>FlexLogix5434</td><td>42</td></tr><tr><td>SoftLogix5860</td><td>15</td></tr></table>	Logix controller:	Product code:	CompactLogix5320	43	CompactLogix5330	44	CompactLogix5335E	65	ControlLogix5550	3	ControlLogix5553	50	ControlLogix5555	51	ControlLogix5561	54	ControlLogix5562	55	ControlLogix5563	56	DriveLogix5720	48	FlexLogix5433	41	FlexLogix5434	42	SoftLogix5860	15
Logix controller:	Product code:																														
CompactLogix5320	43																														
CompactLogix5330	44																														
CompactLogix5335E	65																														
ControlLogix5550	3																														
ControlLogix5553	50																														
ControlLogix5555	51																														
ControlLogix5561	54																														
ControlLogix5562	55																														
ControlLogix5563	56																														
DriveLogix5720	48																														
FlexLogix5433	41																														
FlexLogix5434	42																														
SoftLogix5860	15																														
ProductRev	INT	GSV	Identifies the current product revision. Display should be hexadecimal. The low byte contains the major revision; the high byte contains the minor revision.																												

Attribute:	Data Type:	Instruction:	Description:																																										
SerialNumber	DINT	GSV	Serial number of the device. The serial number is assigned when the device is built.																																										
Status	INT	GSV	Bits identify status: Bits 3-0 are reserved Device Status Bits <table><tr><th>Bits 7-4:</th><th>Meaning:</th></tr><tr><td>0000</td><td>reserved</td></tr><tr><td>0001</td><td>flash update in progress</td></tr><tr><td>0010</td><td>reserved</td></tr><tr><td>0011</td><td>reserved</td></tr><tr><td>0100</td><td>flash is bad</td></tr><tr><td>0101</td><td>faulted</td></tr><tr><td>0110</td><td>run</td></tr><tr><td>0111</td><td>program</td></tr></table> Fault Status Bits <table><tr><th>Bits 11-8:</th><th>Meaning:</th></tr><tr><td>0001</td><td>recoverable minor fault</td></tr><tr><td>0010</td><td>unrecoverable minor fault</td></tr><tr><td>0100</td><td>recoverable major fault</td></tr><tr><td>1000</td><td>unrecoverable major fault</td></tr></table> Logix5000 Specific Status Bits <table><tr><th>Bits 13-12:</th><th>Meaning:</th></tr><tr><td>01</td><td>keyswitch in run</td></tr><tr><td>10</td><td>keyswitch in program</td></tr><tr><td>11</td><td>keyswitch in remote</td></tr></table> <table><tr><th>Bits 15-14</th><th>Meaning</th></tr><tr><td>01</td><td>controller is changing modes</td></tr><tr><td>10</td><td>debug mode if controller is in run mode</td></tr></table>	Bits 7-4:	Meaning:	0000	reserved	0001	flash update in progress	0010	reserved	0011	reserved	0100	flash is bad	0101	faulted	0110	run	0111	program	Bits 11-8:	Meaning:	0001	recoverable minor fault	0010	unrecoverable minor fault	0100	recoverable major fault	1000	unrecoverable major fault	Bits 13-12:	Meaning:	01	keyswitch in run	10	keyswitch in program	11	keyswitch in remote	Bits 15-14	Meaning	01	controller is changing modes	10	debug mode if controller is in run mode
Bits 7-4:	Meaning:																																												
0000	reserved																																												
0001	flash update in progress																																												
0010	reserved																																												
0011	reserved																																												
0100	flash is bad																																												
0101	faulted																																												
0110	run																																												
0111	program																																												
Bits 11-8:	Meaning:																																												
0001	recoverable minor fault																																												
0010	unrecoverable minor fault																																												
0100	recoverable major fault																																												
1000	unrecoverable major fault																																												
Bits 13-12:	Meaning:																																												
01	keyswitch in run																																												
10	keyswitch in program																																												
11	keyswitch in remote																																												
Bits 15-14	Meaning																																												
01	controller is changing modes																																												
10	debug mode if controller is in run mode																																												
Type	INT	GSV	Identifies the device as a controller. Controller = 14																																										
Vendor	INT	GSV	Identifies the vendor of the device. Allen-Bradley = 0001																																										

Accessing the CST object

The CST (coordinated system time) object provides coordinated system time for the devices in one chassis.

Attribute:	Data Type:	Instruction:	Description:	
CurrentStatus	INT	GSV	Current status of the coordinated system time. Bits identify:	
			Bit:	Meaning
			0	timer hardware faulted: the device's internal timer hardware is in a faulted state
			1	ramping enabled: the current value of the timer's lower 16+ bits ramp up to the requested value, rather than snap to the lower value. These bits are manipulated by the network specific tick synchronization method.
			2	system time master: the CST object is a master time source in the ControlLogix system
			3	synchronized: the CST object's 64-bit CurrentValue is synchronized by a master CST object via a system time update
			4	local network master: the CST object is the local network master time source
			5	in relay mode: the CST object is acting in a time relay mode
			6	duplicate master detected: a duplicate local network time master has been detected. This bit is always 0 for time-dependent nodes.
			7	unused
			8-9	00 = time dependent node 01 = time master node 10 = time relay node 11 = unused
CurrentValue	DINT[2]	GSV	10-15 unused	
			Current value of the timer. DINT[0] contains the lower 32; DINT[1] contains the upper 32 bits. The timer source is adjusted to match the value supplied in update services and from local communication network synchronization. The adjustment is either a ramping to the requested value or an immediate setting to the request value, as reported in the CurrentStatus attribute.	

Accessing the DF1 object

The DF1 object provides an interface to the DF1 communication driver that you can configure for the serial port.

Attribute:	Data Type:	Instruction:	Description:
ACKTimeout	DINT	GSV	The amount of time to wait for an acknowledgment to a message transmission (point-to-point and master only). Valid value 0-32,767. Delay in counts of 20 msec periods. Default is 50 (1 second).
DiagnosticCounters	INT[19]	GSV	Array of diagnostic counters for the DF1 communication driver.
word offset	DF1 point-to-point	DF1 slave	master
0	signature (0x0043)	signature (0x0042)	signature (0x0044)
1	modem bits	modem bits	modem bits
2	packets sent	packets sent	packets sent
3	packets received	packets received	packets received
4	undelivered packets	undelivered packets	undelivered packets
5	unused	messages retried	messages retried
6	NAKs received	NAKs received	unused
7	ENQs received	poll packets received	unused
8	bad packets NAKed	bad packets not ACKed	bad packets not ACKed
9	no memory sent NAK	no memory not ACKed	unused
10	duplicate packets received	duplicate packets received	duplicate packets received
11	bad characters received	unused	unused
12	DCD recoveries count	DCD recoveries count	DCD recoveries count
13	lost modem count	lost modem count	lost modem count
14	unused	unused	priority scan time maximum
15	unused	unused	priority scan time last
16	unused	unused	normal scan time maximum
17	unused	unused	normal scan time last
18	ENQs sent	unused	unused
DuplicateDetection	SINT	GSV	Enables duplicate message detection. Value: 0 non zero Meaning: duplicate message detection disabled duplicate message detection disabled
EmbeddedResponseEnable	SINT	GSV	Enables embedded response functionality (point-to-point only). Value: 0 1 Meaning: initiated only after one is received (default) enabled unconditionally
ENQTransmitLimit	SINT	GSV	The number of inquiries (ENQs) to send after an ACK timeout (point-to-point only). Valid value 0-127. Default setting is 3.
EOTSuppression	SINT	GSV	Enable suppressing EOT transmissions in response to poll packets (slave only). Value: 0 non zero Meaning: EOT suppression disabled (disabled) EOT suppression enabled
ErrorDetection	SINT	GSV	Specifies the error-detection scheme. Value: 0 1 Meaning: BCC (default) CRC

Attribute:	Data Type:	Instruction:	Description:
MasterMessageTransmit	SINT	GSV	<p>Current value of the master message transmission (master only).</p> <p>Value: 0 1</p> <p>Meaning: between station polls in poll sequence (in place of master's station number)</p> <p>Default is 0.</p>
NAKReceiveLimit	SINT	GSV	<p>The number of NAKs received in response to a message before stopping transmission (point-to-point communication only). Valid value 0-127. Default is 3.</p>
NormalPollGroupSize	INT	GSV	<p>Number of stations to poll in the normal poll node array after polling all the stations in the priority poll node array (master only). Valid value 0-255. Default is 0.</p>
PollingMode	SINT	GSV	<p>Current polling mode (master only).</p> <p>Value: 0 1 2 3</p> <p>Meaning: message-based, but don't allow slaves to initiate messages message-based, but allow slaves to initiate messages (default) standard, single-message transfer per node scan standard, multiple-message transfer per node scan</p> <p>Default setting is 1.</p>
ReplyMessageWait	DINT	GSV	<p>The time (acting as a master) to wait after receiving an ACK before polling the slave for a response (master only). Valid value 0-65,535. Delay in counts of 20 msec periods. The default is 5 periods (100 msec).</p>
StationAddress	INT	GSV	<p>Current station address of the serial port. Valid value 0-254. Default is 0.</p>
SlavePollTimeout	DINT	GSV	<p>The amount of time in msec that the slave waits for the master to poll before the slave declares that it is unable to transmit because the master is inactive (slave only). Valid value 0-32,767. Delay in counts of 20 msec periods. The default is 3000 periods (1 minute).</p>
TransmitRetries	SINT	GSV	<p>Number of times to resend a message without getting an acknowledgment (master and slave only). Valid value 0-127. Default is 3.</p>
PendingACKTimeout	DINT	SSV	Pending value for the ACKTimeout attribute.
PendingDuplicateDetection	SINT	SSV	Pending value for the DuplicateDetection attribute.
PendingEmbeddedResponse Enable	SINT	SSV	Pending value for the EmbeddedResponse attribute.
PendingENQTransmitLimit	SINT	SSV	Pending value for the ENQTransmitLimit attribute.
PendingEOTSuppression	SINT	SSV	Pending value for the EOTSuppression attribute.
PendingErrorDetection	SINT	SSV	Pending value for the ErrorDetection attribute.
PendingNormalPollGroupSize	INT	SSV	Pending value for the NormalPollGroupSize attribute.
PendingMasterMessage Transmit	SINT	SSV	Pending value for the MasterMessageTransmit attribute.
PendingNAKReceiveLimit	SINT	SSV	Pending value for the NAKReceiveLimit attribute.
PendingPollingMode	SINT	SSV	Pending value for the PollingMode attribute.

Attribute:	Data Type:	Instruction:	Description:
PendingReplyMessageWait	DINT	SSV	Pending value for the ReplyMessageWait attribute.
PendingStationAddress	INT	SSV	Pending value for the StationAddress attribute.
PendingSlavePollTimeout	DINT	SSV	Pending value for the SlavePollTimeout attribute.
PendingTransmitRetries	SINT	SSV	Pending value for the TransmitRetries attribute.

To apply values for any of the DF1 pending attributes:

1. Use an SSV instruction to set the value for the pending attribute.

You can set as many pending attributes as you want, using an SSV instruction for each pending attribute.

2. Use a MSG instruction to apply the value. The MSG instruction applies every pending attribute you set. Configure the MSG instruction as:

MSG Configuration Tab:	Field:	Value:
Configuration	Message Type	CIP Generic
	Service Code	0d hex
	Object Type	a2
	Object ID	1
	Object Attribute	leave blank
	Source	leave blank
	Number of Elements	0
	Destination	leave blank
Communication	Path	communication path to self (1,s where s = slot number of controller)

Accessing the FAULTLOG object

The FAULTLOG object provides fault information about the controller.

Attribute:	Data Type:	Instruction:	Description:																		
MajorEvents	INT	GSV SSV	How many major faults have occurred since the last time this counter was reset.																		
MinorEvents	INT	GSV SSV	How many minor faults have occurred since the last time this counter was reset.																		
MajorFaultBits	DINT	GSV SSV	Individual bits indicate the reason for the current major fault. <table><tr><th>Bit:</th><th>Meaning:</th></tr><tr><td>1</td><td>power loss</td></tr><tr><td>3</td><td>I/O</td></tr><tr><td>4</td><td>instruction execution (program)</td></tr><tr><td>5</td><td>fault handler</td></tr><tr><td>6</td><td>watchdog</td></tr><tr><td>7</td><td>stack</td></tr><tr><td>8</td><td>mode change</td></tr><tr><td>11</td><td>motion</td></tr></table>	Bit:	Meaning:	1	power loss	3	I/O	4	instruction execution (program)	5	fault handler	6	watchdog	7	stack	8	mode change	11	motion
Bit:	Meaning:																				
1	power loss																				
3	I/O																				
4	instruction execution (program)																				
5	fault handler																				
6	watchdog																				
7	stack																				
8	mode change																				
11	motion																				
MinorFaultBits	DINT	GSV SSV	Individual bits indicate the reason for the current minor fault. <table><tr><th>Bit:</th><th>Meaning:</th></tr><tr><td>4</td><td>instruction execution (program)</td></tr><tr><td>6</td><td>watchdog</td></tr><tr><td>9</td><td>serial port</td></tr><tr><td>10</td><td>battery</td></tr></table>	Bit:	Meaning:	4	instruction execution (program)	6	watchdog	9	serial port	10	battery								
Bit:	Meaning:																				
4	instruction execution (program)																				
6	watchdog																				
9	serial port																				
10	battery																				

Accessing the MESSAGE object

You can access the MESSAGE object through the GSV/SSV instructions. Specify the message tag name to determine which MESSAGE object you want. The MESSAGE object provides an interface to setup and trigger peer-to-peer communications. This object replaces the MG data type of the PLC-5 processor.

Attribute:	Data Type:	Instruction:	Description:
ConnectionPath	SINT[130]	GSV SSV	Data to setup the connection path. The first two bytes (low byte and high byte) are the length in bytes of the connection path.
ConnectionRate	DINT	GSV SSV	Requested packet rate of the connection.
MessageType	SINT	GSV SSV	Specifies the type of message. Value: 0 Meaning: not initialized
Port	SINT	GSV SSV	Indicates which port the message should be sent on. Value: 1 2 Meaning: backplane serial port
TimeoutMultiplier	SINT	GSV SSV	Determines when a connection should be considered timed out and closed. Value: 0 1 2 Meaning: connection will timeout in 4 times the update rate (default) connection will timeout in 8 times the update rate connection will timeout in 16 times the update rate
UnconnectedTimeout	DINT	GSV SSV	Timeout period in microseconds for all unconnected messages. The default is 30,000,000 microseconds (30 seconds).

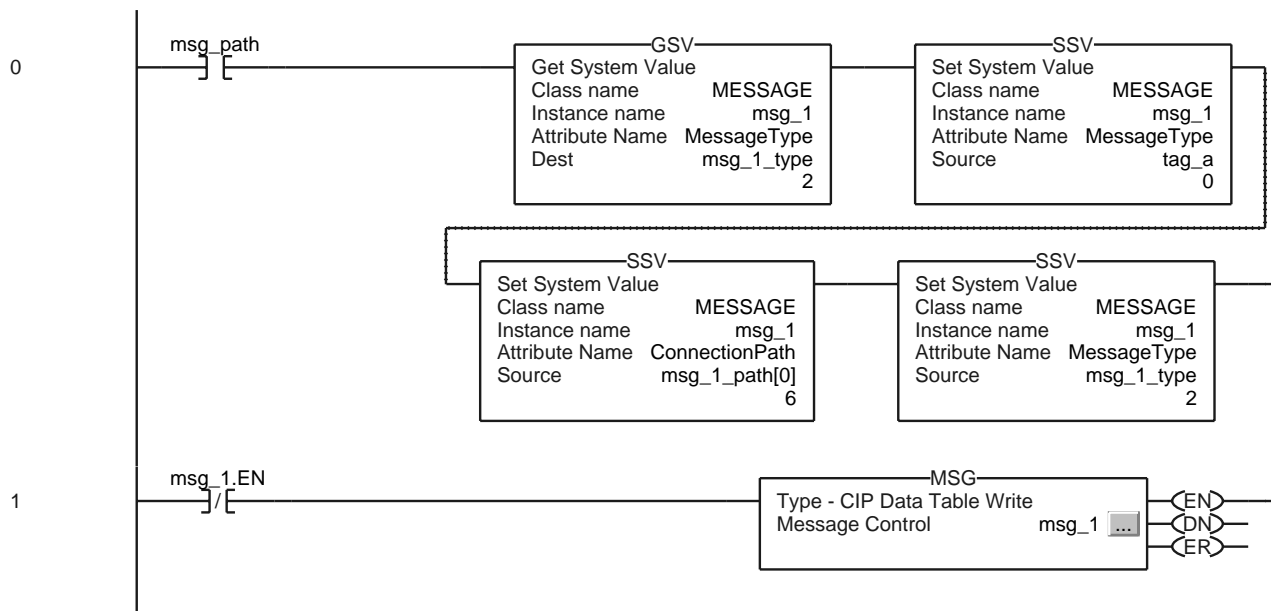
To change a MESSAGE attribute, follow these steps:

1. Use a GSV instruction to get the MessageType attribute and save it in a tag.
2. Use a SSV instruction to set the MessageType to 0.
3. Use a SSV instruction to set the MESSAGE attribute that you want to change.
4. Use a SSV instruction to set the MessageType attribute back to the original value you obtained in step 1.

Example: The following example changes the ConnectionPath attribute, so that the message goes to a different controller. When *msg_path* is on, sets the path of the *msg_1* message to the value of *msg_1_path*. This send the message to a different controller.

Where:	Is:
<i>msg_1</i>	message whose attribute you want to change
<i>msg_1_type</i>	tag that stores the value of the MessageType attribute
<i>tag_a</i>	tag that stores a 0.
<i>msg_1_path</i>	array tag that stores the new connection path for the message

Relay Ladder



Structured Text

```

IF msg_path THEN
    GSV (MESSAGE, msg_1, MessageType, msg_1_type) ;
    SSV (MESSAGE, msg_1, MessageType, tag_a) ;
    SSV (MESSAGE, msg_1, ConnectionPath, msg_1_path[0]) ;
    SSV (MESSAGE, msg_1, MessageType, msg_1_type) ;
END_IF;

IF NOT msg_1.EN THEN
    MSG (msg_1) ;
END_IF;

```

Accessing the MODULE object

The MODULE object provides status information about a module. To select a particular MODULE object, set the Object Name operand of the GSV/SSV instruction to the module name. The specified module must be present in the I/O Configuration section of the controller organizer and must have a device name.

Attribute:	Data Type:	Instruction:	Description:	
EntryStatus	INT	GSV	Specifies the current state of the specified map entry. The lower 12 bits should be masked when performing a comparison operation. Only bits 12-15 are valid.	
			Value: 16#0000 16#1000	Meaning: Standby: the controller is powering up. Faulted: any of the MODULE object's connections to the associated module fail. This value should not be used to determine if the module failed because the MODULE object leaves this state periodically when trying to reconnect to the module. Instead, test for Running state (16#4000). Check for FaultCode not equal to 0 to determine if a module is faulted. When Faulted, the FaultCode and FaultInfo attributes are valid until the fault condition is corrected.
			16#2000	Validating: the MODULE object is verifying MODULE object integrity prior to establishing connections to the module.
			16#3000	Connecting: the MODULE object is initiating connections to the module.
			16#4000	Running: all connections to the module are established and data is successfully transferring.
			16#5000	Shutting down: the MODULE object is in the process of shutting down all connections to the module.
			16#6000	Inhibited: the MODULE object is inhibited (the inhibit bit in the Mode attribute is set).
			16#7000	Waiting: the parent MODULE object upon which this MODULE object depends is not running.
			FaultCode	INT
FaultInfo	DINT	GSV	Provides specific information about the MODULE object fault code.	
ForceStatus	INT	GSV	Specifies the status of forces. Bit: 0 1 2-15 Meaning: forces installed (1=yes, 0=no) forces enabled (1=yes, 0=no) not used	

Attribute:	Data Type:	Instruction:	Description:										
Instance	DINT	GSV	Provides the instance number of this MODULE object.										
LEDStatus	INT	GSV	<div>Specifies the current state of the I/O LED on the front of the controller.</div> <table><tr><th>Value:</th><th>Meaning:</th></tr><tr><td>0</td><td>LED off: No MODULE objects are configured for the controller (there are no modules in the I/O Configuration section of the controller organizer).</td></tr><tr><td>1</td><td>Flashing red: None of the MODULE objects are Running.</td></tr><tr><td>2</td><td>Flashing green: At least one MODULE object is not Running.</td></tr><tr><td>3</td><td>Solid green: All the Module objects are Running.</td></tr></table> <div>Note: You do not enter an object name with this attribute because this attribute applies to the entire collection of modules.</div>	Value:	Meaning:	0	LED off: No MODULE objects are configured for the controller (there are no modules in the I/O Configuration section of the controller organizer).	1	Flashing red: None of the MODULE objects are Running.	2	Flashing green: At least one MODULE object is not Running.	3	Solid green: All the Module objects are Running.
Value:	Meaning:												
0	LED off: No MODULE objects are configured for the controller (there are no modules in the I/O Configuration section of the controller organizer).												
1	Flashing red: None of the MODULE objects are Running.												
2	Flashing green: At least one MODULE object is not Running.												
3	Solid green: All the Module objects are Running.												
Mode	INT	GSV SSV	<div>Specifies the current mode of the MODULE object.</div> <table><tr><th>Bit:</th><th>Meaning:</th></tr><tr><td>0</td><td>If set, causes a major fault to be generated if any of the MODULE object connections fault while the controller is in Run mode.</td></tr><tr><td>2</td><td>If set, causes the MODULE object to enter Inhibited state after shutting down all the connections to the module.</td></tr></table>	Bit:	Meaning:	0	If set, causes a major fault to be generated if any of the MODULE object connections fault while the controller is in Run mode.	2	If set, causes the MODULE object to enter Inhibited state after shutting down all the connections to the module.				
Bit:	Meaning:												
0	If set, causes a major fault to be generated if any of the MODULE object connections fault while the controller is in Run mode.												
2	If set, causes the MODULE object to enter Inhibited state after shutting down all the connections to the module.												

Accessing the MOTIONGROUP object

The MOTIONGROUP object provides status information about a group of axes for the servo module. Specify the motion-group tag name to determine which MOTIONGROUP object you want.

Attribute:	Data Type:	Instruction:	Description:
Instance	DINT	GSV	Provides the instance number of this MOTION_GROUP object.

Accessing the PROGRAM object

The PROGRAM object provides status information about a program. Specify the program name to determine which PROGRAM object you want.

Attribute:	Data Type:	Instruction:	Description:
DisableFlag	SINT	GSV SSV	Controls this program's execution. Value: 0 1 Meaning: execution enabled execution disabled
Instance	DINT	GSV	Provides the instance number of this PROGRAM object.
LastScanTime	DINT	GSV SSV	Time it took to execute this program the last time it was executed. Time is in microseconds.
MajorFaultRecord	DINT[11]	GSV SSV	Records major faults for this program We recommend that you create a user-defined structure to simplify access to the MajorFaultRecord attribute:
Name: TimeLow TimeHigh Type Code Info	Data Type: DINT DINT INT INT DINT[8]	Style: Decimal Decimal Decimal Decimal Hexadecimal	Description: lower 32 bits of fault timestamp value upper 32 bits of fault timestamp value fault type (program, I/O, etc.) unique code for the fault (depends on fault type) fault specific information (depends on fault type and code)
MaxScanTime	DINT	GSV SSV	Maximum recorded execution time for this program. Time is in microseconds.
MinorFaultRecord	DINT[11]	GSV SSV	Records minor faults for this program We recommend that you create a user-defined structure to simplify access to the MinorFaultRecord attribute:
Name: TimeLow TimeHigh Type Code Info	Data Type: DINT DINT INT INT DINT[8]	Style: Decimal Decimal Decimal Decimal Hexadecimal	Description: lower 32 bits of fault timestamp value upper 32 bits of fault timestamp value fault type (program, I/O, etc.) unique code for the fault (depends on fault type) fault specific information (depends on fault type and code)
SFCRestart	INT	GSV SSV	unused - reserved for future use

Accessing the ROUTINE object

The ROUTINE object provides status information about a routine. Specify the routine name to determine which ROUTINE object you want.

Attribute:	Data Type:	Instruction:	Description:
Instance	DINT	GSV	Provides the instance number of this ROUTINE object. Valid values are 0-65,535.

Accessing the SERIALPORT object

The SERIALPORT object provides an interface to the serial communication port.

Attribute:	Data Type:	Instruction:	Description:
BaudRate	DINT	GSV	Specifies the baud rate. Valid values are 110, 300, 600, 1200, 2400, 4800, 9600, and 19200 (default).
DataBits	SINT	GSV	Specifies the number of bits of data per character. Value: 7 8 Meaning: 7 data bits (ASCII only) 8 data bits (default)
Parity	SINT	GSV	Specifies the parity. Value: 0 1 2 Meaning: no parity (no default) odd parity (ASCII only) even parity
RTSOffDelay	INT	GSV	Amount of time to delay turning off the RTS line after the last character has been transmitted. Valid value 0-32,767. Delay in counts of 20 msec periods. The default is 0 msec.
RTSSendDelay	INT	GSV	Amount of time to delay transmitting the first character of a message after turning on the RTS line. Valid value 0-32,767. Delay in counts of 20 msec periods. The default is 0 msec.
StopBits	SINT	GSV	Specifies the number of stop bits. Value: 1 2 Meaning: 1 stop bit (default) 2 stop bits (ASCII only)
PendingBaudRate	DINT	SSV	Pending value for the BaudRate attribute.
PendingDataBits	SINT	SSV	Pending value for the DataBits attribute.
PendingParity	SINT	SSV	Pending value for the Parity attribute.

Attribute:	Data Type:	Instruction:	Description:
PendingRTSOffDelay	INT	SSV	Pending value for the RTSOffDelay attribute.
PendingRTSSendDelay	INT	SSV	Pending value for the RTSSendDelay attribute.
PendingStopBits	SINT	SSV	Pending value for the StopBits attribute.

To apply values for any of the SERIALPORT pending attributes:

1. Use an SSV instruction to set the value for the pending attribute.

You can set as many pending attributes as you want, using an SSV instruction for each pending attribute.

2. Use a MSG instruction to apply the value. The MSG instruction applies every pending attribute you set. Configure the MSG instructions as:

MSG Configuration Tab:	Field:	Value:
Configuration	Message Type	CIP Generic
	Service Code	0d hex
	Object Type	6f hex
	Object ID	1
	Object Attribute	leave blank
	Source	leave blank
	Number of Elements	0
	Destination	leave blank
Communication	Path	communication path to self (1,s where s = slot number of controller)

Accessing the TASK object

The TASK object provides status information about a task. Specify the task name to determine which TASK object you want.

Attribute:	Data Type:	Instruction:	Description:
DisableUpdateOutputs	DINT	GSV SSV	Enables or disables the processing of outputs at the end of a task
			To: enable the processing of outputs at the end of the task
			Set the attribute to: 0
EnableTimeOut	DINT	GSV SSV	Enables or disables the timeout function of an event task.
			To: disable the timeout function
			Set the attribute to: 1 (or any non-zero value)
InhibitTask	DINT	GSV SSV	Prevents the task from executing. If a task is inhibited, the controller still prescans the task when the controller transitions from program to run or test mode.
			To: enable the task
			Set the attribute to: 0 (default)
Instance	DINT	GSV	Provides the instance number of this TASK object. Valid values are 0-31.
			To: inhibit (disable) the task
			Set the attribute to: 1 (or any non-zero value)
LastScanTime	DINT	GSV SSV	Time it took to execute this task the last time it was executed. Time is in microseconds.
MaxInterval	DINT[2]	GSV SSV	The maximum time interval between successive executions of the task. DINT[0] contains the lower 32 bits of the value; DINT[1] contains the upper 32 bits of the value. A value of 0 indicates 1 or less executions of the task.
MaxScanTime	DINT	GSV SSV	Maximum recorded execution time for this program. Time is in microseconds.
MinInterval	DINT[2]	GSV SSV	The minimum time interval between successive executions of the task. DINT[0] contains the lower 32 bits of the value; DINT[1] contains the upper 32 bits of the value. A value of 0 indicates 1 or less executions of the task.
OverlapCount	DINT	GSV SSV	Number of times that the task was triggered while it was still executing. Valid for an event or a periodic task.
			To clear the count, set the attribute to 0.
Priority	INT	GSV SSV	Relative priority of this task as compared to the other tasks. Valid values 0-15.

Attribute:	Data Type:	Instruction:	Description:	
Rate	DINT	GSV SSV	If the task type is:	Then the Rate attribute specifies the:
			periodic	Period for the task. Time is in microseconds.
			event	The timeout value for the task. Time is in microseconds.
StartTime	DINT[2]	GSV SSV	Value of WALLCLOCKTIME when the last execution of the task was started. DINT[0] contains the lower 32 bits of the value; DINT[1] contains the upper 32 bits of the value.	
Status	DINT	GSV SSV	Provides status information about the task. Once the controller sets one of these bits, you must manually clear the bit.	
			To determine if:	Examine this bit:
			An EVNT instruction triggered the task (event task only).	0
			A timeout triggered the task (event task only).	1
Watchdog	DINT	GSV SSV	An overlap occurred for this task.	2
			Time limit for execution of all programs associated with this task. Time is in microseconds.	
			If you enter 0, these values are assigned:	
			Time:	Task Type:
			0.5 sec	periodic or event
			5.0 sec	continuous

Accessing the WALLCLOCKTIME object

The WALLCLOCKTIME object provides a timestamp the controller can use for scheduling.

Attribute:	Data Type:	Instruction:	Description:
CSTOffset	DINT[2]	GSV SSV	Positive offset from the CurrentValue of the CST object (coordinated system time, see page 3-39). DINT[0] contains the lower 32 bits of the value; DINT[1] contains the upper 32 bits of the value. Value in μ secs. The default is 0.
CurrentValue	DINT[2]	GSV SSV	Current value of the wall clock time. DINT[0] contains the lower 32 bits of the value; DINT[1] contains the upper 32 bits of the value. The value is the number of microseconds that have elapsed since 0000 hrs 1 January 1972. The CST and WALLCLOCKTIME objects are mathematically related in the controller. For example, if you add the CST CurrentValue and the WALLCLOCKTIME CSTOffset, the result is the WALLCLOCKTIME CurrentValue.
DateTime	DINT[7]	GSV SSV	The date and time in a readable format. DINT[0] year DINT[1] integer representation of month (1-12) DINT[2] integer representation of day (1-31) DINT[3] hour (0-23) DINT[4] minute (0-59) DINT[5] seconds (0-59) DINT[6] microseconds (0-999,999)

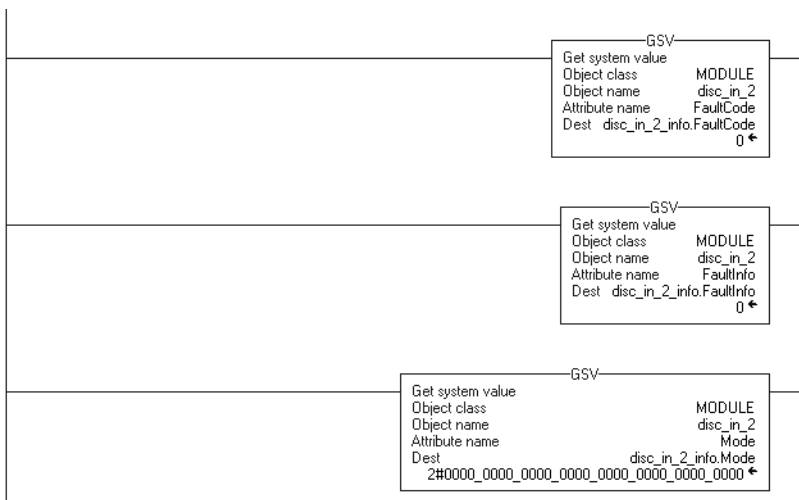
GSV/SSV Programming Example

Getting fault information

The following examples use GSV instructions to get fault information.

Example 1: This example gets fault information from the I/O module *disc_in_2* and places the data in a user-defined structure *disc_in_2_info*.

Relay Ladder



Structured Text

```

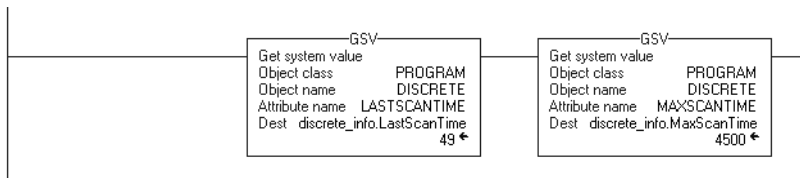
GSV(MODULE, disc_in_2, FaultCode, disc_in_2_info.FaultCode);

GSV(MODULE, disc_in_2, FaultInfo, disc_in_2_info.FaultInfo);

GSV(MODULE, disc_in_2, Mode, disc_in_2_info.Mode);
  
```

Example 2: This example gets status information about program *discrete* and places the data in a user-defined structure *discrete_info*.

Relay Ladder



Structured Text

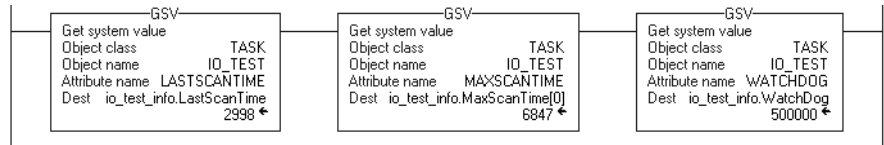
```

GSV(PROGRAM, DISCRETE, LASTSCANTIME,
    discrete_info.LastScanTime);

GSV(PROGRAM, DISCRETE, MAXSCANTIME, discrete_info.MaxScanTime);
  
```


Example 3: This example gets status information about task *IO_test* and places the data in a user-defined structure *io_test_info*.

Relay Ladder



Structured Text

```
GSV(TASK, IO_TEST, LASTSCANTIME, io_test_info.LastScanTime);

GSV(TASK, IO_TEST, MAXSCANTIME, io_test_info.MaxScanTime);

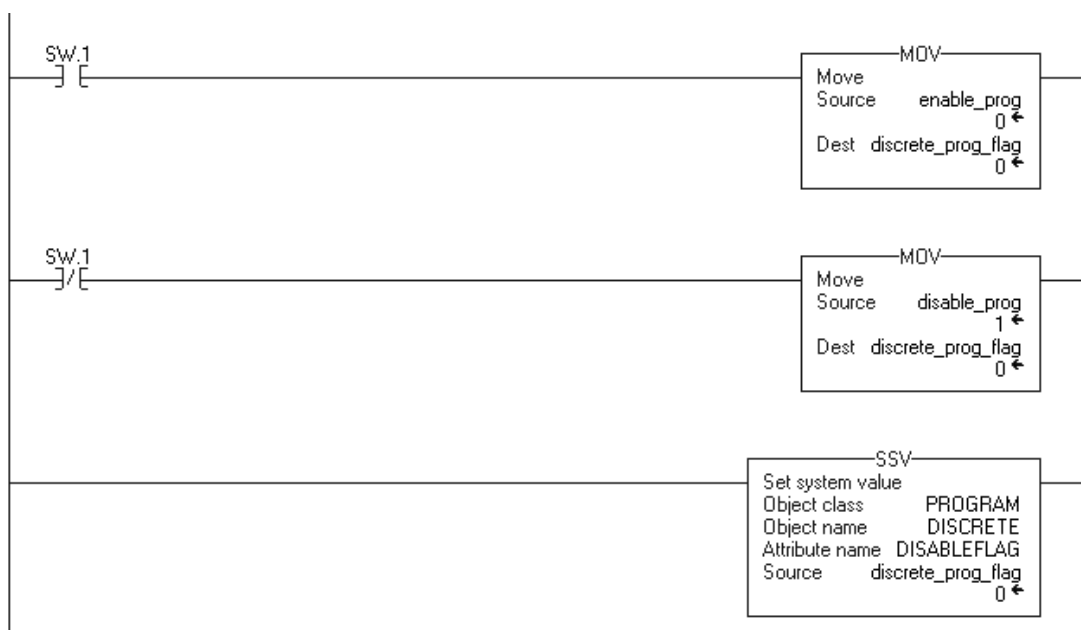
GSV(TASK, IO_TEST, WATCHDOG, io_test_info.WatchDog);
```

Setting enable and disable flags

The following example uses the SSV instruction to enable or disable a program. You could also use this method to enable or disable an I/O module, which is a similar to using inhibit bits with a PLC-5 processor.

Example: Based on the status of *SW.1*, place the appropriate value in the *disableflag* attribute of program *discrete*.

Relay Ladder



Structured Text

```
IF SW.1 THEN

    discrete_prog_flag := enable_prog;

ELSE

    discrete_prog_flag := disable_prog;

END_IF;

SSV (PROGRAM,DISCRETE,DISABLEFLAG,discrete_prog_flag);
```

Immediate Output (IOT)

The IOT instruction immediately updates the specified output data (output tag or produced tag).

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Update Tag		tag	tag that you want to update, either: <ul style="list-style-type: none"> • output tag of an I/O module • produced tag <p><i>Do not</i> choose a member or element of a tag. For example, Local:5:0 is OK but Local:5:0.Data is <i>not</i> OK.</p>



`IOT(output_tag);`

Structured Text

The operands are the same as those for the relay ladder IOT instruction.

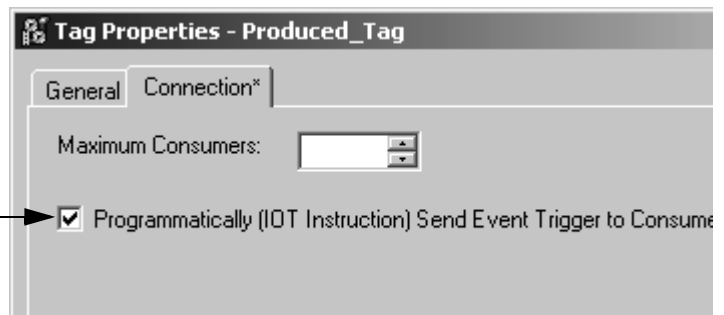
Description: The IOT instruction overrides the requested packet interval (RPI) of an output connection and sends fresh data over the connection.

- An output connection is a connection that is associated with the output tag of an I/O module or with a produced tag.
- If the connection is for a produced tag, the IOT instruction also sends the event trigger to the consuming controller. This lets the IOT instruction trigger an event task in the consuming controller.

To use an IOT instruction and a produced tag to trigger an event task in a consumer controller, configure the produced tag as follows:

This configures the tag to update its event trigger only via an IOT instruction.

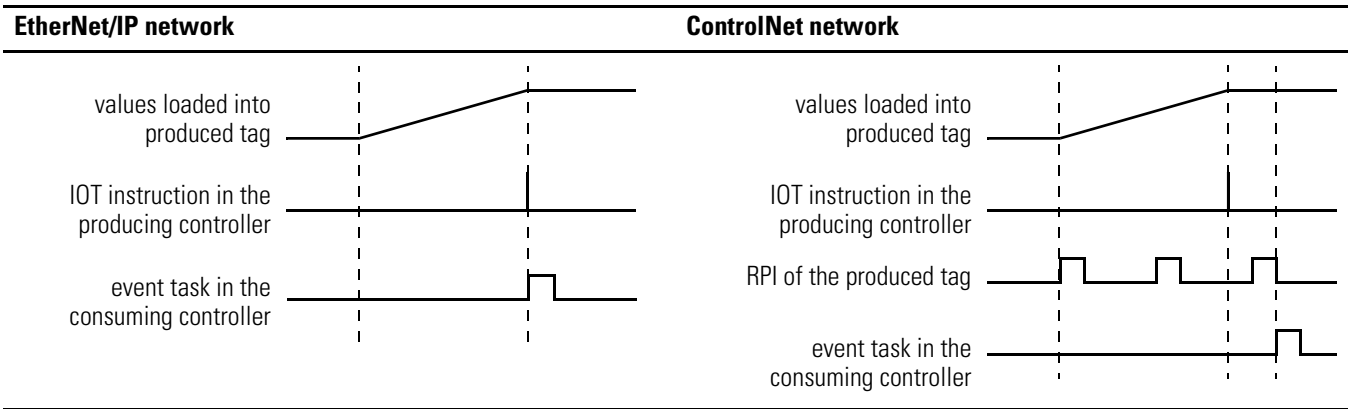
Check this box.



The type of network between the controllers determines when the consuming controller receives the new data and event trigger via the IOT instruction.

With this controller:	Over this network:	The consuming device receives the data and event trigger:
ControlLogix	backplane	immediately
	EtherNet/IP network	immediately
	ControlNet network	within the actual packet interval (API) of the consumed tag (connection)
SoftLogix5800	You can produce and consume tags only over a ControlNet network.	within the actual packet interval (API) of the consumed tag (connection)

The following diagrams compare the receipt of data via an IOT instruction over EtherNet/IP and ControlNet networks.



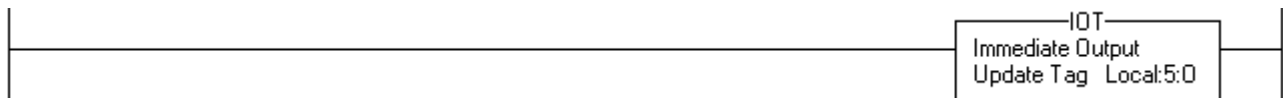
Arithmetic Status Flags: not affected

Fault Conditions: none

Execution:

Condition:	Relay Ladder Action:	Structured Text Action:
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	na
EnableIn is set	na	EnableIn is always set. The instruction executes.
instruction execution	The instruction: <ul style="list-style-type: none"> • updates the connection of the specified tag. • resets the RPI timer of the connection 	
postscan	The rung-condition-out is set to false.	No action taken.

Example 1: When the IOT instruction executes, it immediately sends the values of the *Local:5:0* tag to the output module.

Relay Ladder**Structured Text**

```
IOT (Local:5:0);
```

Example 2: This controller controls station 24 and produces data for the next station (station 25). To use an IOT instruction to signal the transmission of new data, the produced tag is configured as follows:

Produced_Tag is configured to update its event trigger via an IOT instruction.

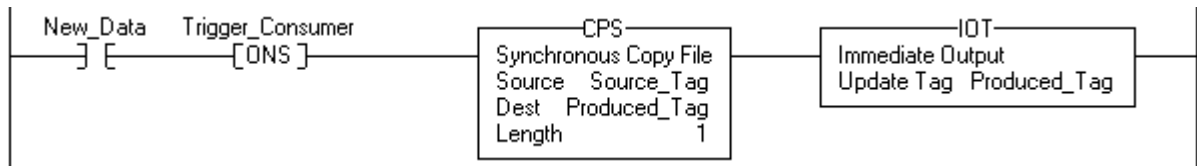


Relay Ladder

If *New_Data* = on, then the following occurs for one scan:

The CPS instruction sets *Produced_Tag* = *Source_Tag*.

The IOT instruction updates *Produced_Tag* and sends this update to the consuming controller (station 25). When the consuming controller receives this update, it triggers the associated event task in that controller.



Structured Text

```
IF New_Data AND NOT Trigger_Consumer THEN
    CPS (Source_Tag, Produced_Tag, 1);
    IOT (Produced_Tag);
END_IF;
Trigger_Consumer := New_Data;
```

Compare Instructions

(CMP, EQU, GEQ, GRT, LEQ, LES, LIM, MEQ, NEQ)

Introduction

The compare instructions let you compare values by using an expression or a specific compare instruction.

If you want to:	Use this instruction:	Available in these languages:	See page:
compare values based on an expression	CMP	relay ladder structured text ⁽¹⁾	4-2
test whether two values are equal	EQU	relay ladder structured text ⁽²⁾ function block	4-7
test whether one value is greater than or equal to a second value	GEQ	relay ladder structured text ⁽¹⁾ function block	4-11
test whether one value is greater than a second value	GRT	relay ladder structured text ⁽¹⁾ function block	4-15
test whether one value is less than or equal to a second value	LEQ	relay ladder structured text ⁽¹⁾ function block	4-19
test whether one value is less than a second value	LES	relay ladder structured text ⁽¹⁾ function block	4-23
test whether one value is between two other values	LIM	relay ladder structured text ⁽¹⁾ function block	4-27
pass two values through a mask and test whether they are equal	MEQ	relay ladder structured text ⁽¹⁾ function block	4-33
test whether one value is not equal to a second value	NEQ	relay ladder structured text ⁽¹⁾ function block	4-38

⁽¹⁾ There is no equivalent structured text instruction. Use other structured text programming to achieve the same result. See the description for the instruction.

⁽²⁾ There is no equivalent structured text instruction. Use the operator in an expression.

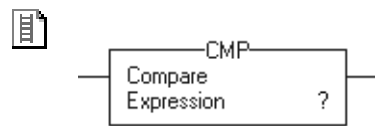
You can compare values of different data types, such as floating point and integer.

For relay ladder instructions, **bold** data types indicate optimal data types. An instruction executes faster and requires less memory if all the operands of the instruction use the same optimal data type, typically DINT or REAL.

Compare (CMP)

The CMP instruction performs a comparison on the arithmetic operations you specify in the expression.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Expression	SINT INT DINT REAL string	immediate tag	an expression consisting of tags and/or immediate values separated by operators

A SINT or INT tag converts to a DINT value by sign-extension.



Structured Text

Structured text does not have a CMP instruction, but you can achieve the same results using an IF...THEN construct and expression.

```
IF BOOL_expression THEN
    <statement>;
END_IF;
```

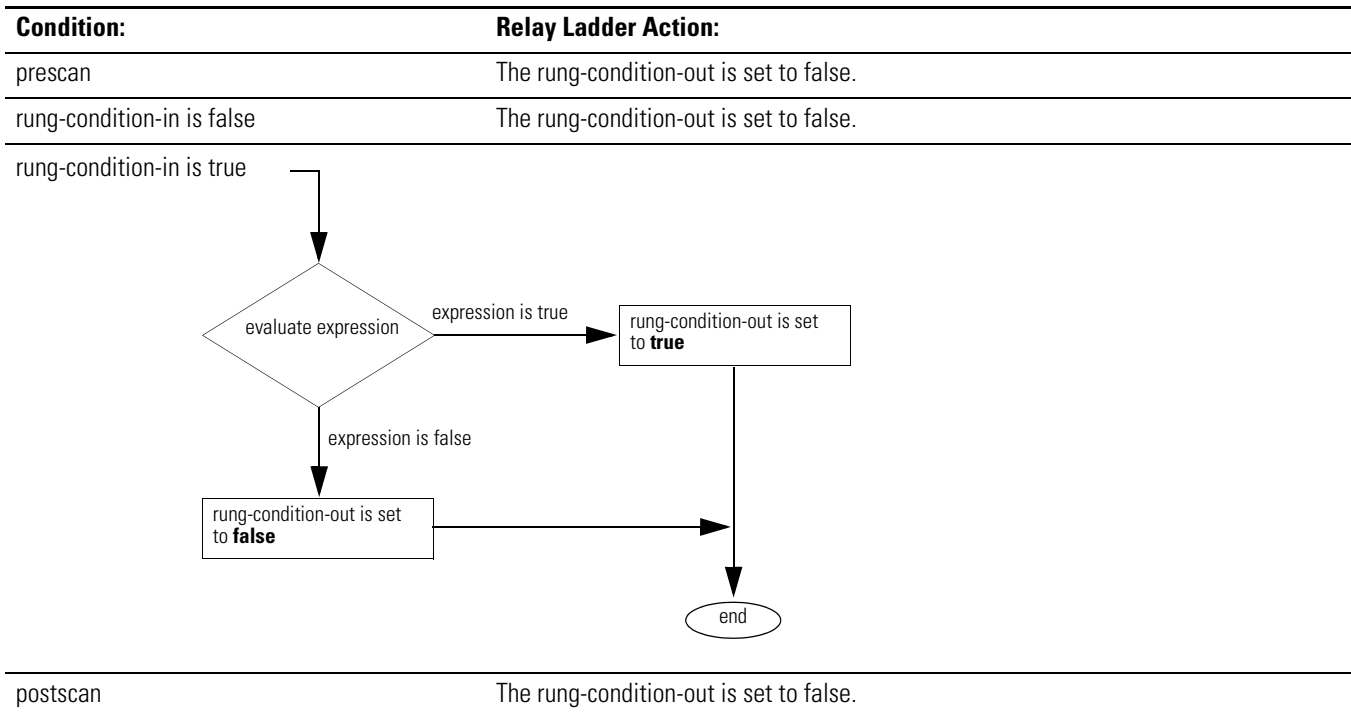
See Appendix C for information on the syntax of constructs and expressions within structured text.

Description: Define the CMP expression using operators, tags, and immediate values. Use parentheses () to define sections of more complex expressions.

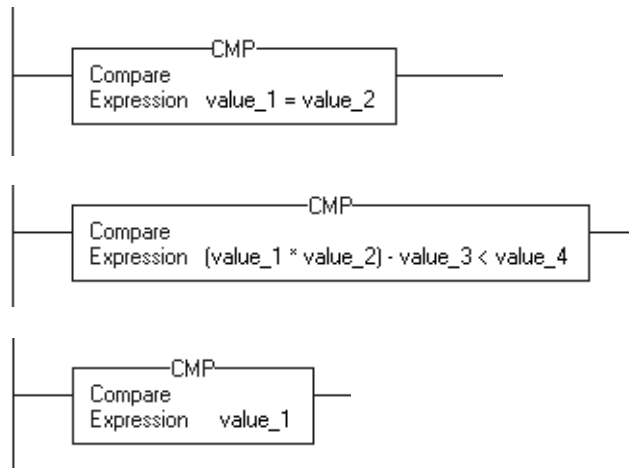
The execution of a CMP instruction is slightly slower and uses more memory than the execution of the other comparison instructions. The advantage of the CMP instruction is that it allows you to enter complex expressions in one instruction.

Arithmetic Status Flags: The CMP instruction only affects the arithmetic status flags if the expression contains an operator (e.g., +, −, *, /) that affects the arithmetic status flags.

Fault Conditions: none

Execution:

Examples: If the CMP instruction finds the expression true, the rung-condition-out is set to true.



If you enter an expression without a comparison operator, such as *value_1 + value_2*, or *value_1*, the instruction evaluates the expression as:

If the expression:	The rung-condition-out is set to:
non zero	true
zero	false

CMP expressions

You program expressions in CMP instructions the same as expressions in FSC instructions. Use the following sections for information on valid operators, format, and order of operation, which are common to both instructions.

Valid operators

Operator:	Description:	Optimal:
+	add	DINT, REAL
-	subtract/negate	DINT, REAL
*	multiply	DINT, REAL
/	divide	DINT, REAL
=	equal	DINT, REAL
<	less than	DINT, REAL
<=	less than or equal	DINT, REAL
>	greater than	DINT, REAL
>=	greater than or equal	DINT, REAL
<>	not equal	DINT, REAL
**	exponent (x to y)	DINT, REAL
ABS	absolute value	DINT, REAL
ACS	arc cosine	REAL
AND	bitwise AND	DINT
ASN	arc sine	REAL
ATN	arc tangent	REAL
COS	cosine	REAL

Operator:	Description:	Optimal:
DEG	radians to degrees	DINT, REAL
FRD	BCD to integer	DINT
LN	natural log	REAL
LOG	log base 10	REAL
MOD	modulo-divide	DINT, REAL
NOT	bitwise complement	DINT
OR	bitwise OR	DINT
RAD	degrees to radians	DINT, REAL
SIN	sine	REAL
SQR	square root	DINT, REAL
TAN	tangent	REAL
TOD	integer to BCD	DINT
TRN	truncate	DINT, REAL
XOR	bitwise exclusive OR	DINT

Formatting expressions

For each operator that you use in an expression, you have to provide one or two operands (tags or immediate values). Use the following table to format operators and operands within an expression:

For operators that operate on:	Use this format:	Examples:
one operand	operator(operand)	$ABS(tag_a)$
two operands	operand_a operator operand_b	<ul style="list-style-type: none"> $tag_b + 5$ $tag_c \text{ AND } tag_d$ $(tag_e ** 2) \text{ MOD } (tag_f / tag_g)$

Determining the order of operation

The operations you write into the expression are performed by the instruction in a prescribed order, not necessarily the order you write them. You can override the order of operation by grouping terms within parentheses, forcing the instruction to perform an operation within the parentheses ahead of other operations.

Operations of equal order are performed from left to right.

Order:	Operation:
1.	()
2.	ABS, ACS, ASN, ATN, COS, DEG, FRD, LN, LOG, RAD, SIN, SQR, TAN, TOD, TRN
3.	**
4.	– (negate), NOT
5.	*, /, MOD
6.	<, <=, >, >=, =
7.	– (subtract), +
8.	AND
9.	XOR
10.	OR

Using strings in an expression

Use a relay ladder or structured text expression to compare string data types. To use strings in an expression, follow these guidelines:

- An expression lets you compare two string tags.
- You *cannot* enter ASCII characters directly into the expression.
- Only the following operators are permitted

Operator:	Description:
=	equal
<	less than
<=	less than or equal
>	greater than
>=	greater than or equal
<>	not equal

- Strings are equal if their characters match.
- ASCII characters are case sensitive. Upper case “A” (\$41) is *not* equal to lower case “a” (\$61).
- The hexadecimal values of the characters determine if one string is less than or greater than another string. For the hex code of a character, see the back cover of this manual.
- When the two strings are sorted as in a telephone directory, the order of the strings determines which one is greater.

ASCII Characters	Hex Codes
1ab	\$31\$61\$62
1b	\$31\$62
A	\$41
AB	\$41\$42
B	\$42
a	\$61
ab	\$61\$62

l
e
s
s
e
r

↑

g
r
e
a
t
e
r

↓

—

AB < B

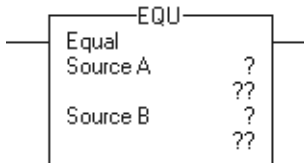
—

a > B

Equal to (EQU)

The EQU instruction tests whether Source A is equal to Source B.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Source A	SINT INT DINT REAL string	immediate tag	value to test against Source B
Source B	SINT INT DINT REAL string	immediate tag	value to test against Source A

- If you enter a SINT or INT tag, the value converts to a DINT value by sign-extension.
- REAL values are rarely absolutely equal. If you need to determine the equality of two REAL values, use the LIM instruction.
- String data types are:
 - default STRING data type
 - any new string data type that you create
- To test the characters of a string, enter a string tag for both Source A and Source B.

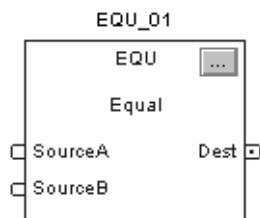


```
IF sourceA = sourceB THEN
    <statements>;
```

Structured Text

Use the equal sign “=” as an operator within an expression. This expression evaluates whether *sourceA* is equal to *sourceB*.

See Appendix C for information on the syntax of expressions within structured text.



Function Block

Operand:	Type:	Format:	Description:
EQU tag	FBD_COMPARE	structure	EQU structure

FBD_COMPARE Structure

Input Parameter:	Data Type:	Description:
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set.
SourceA	REAL	Value to test against SourceB. Valid = any float
SourceB	REAL	Value to test against SourceA. Valid = any float
Output Parameter:	Data Type:	Description:
EnableOut	BOOL	The instruction produced a valid result.
Dest	BOOL	Result of the instruction. This is equivalent to rung-condition-out of the relay ladder EQU instruction.

Description: Use the EQU instruction to compare two numbers or two strings of ASCII characters. When you compare strings:

- Strings are equal if their characters match.
- ASCII characters are case sensitive. Upper case “A” (\$41) is *not* equal to lower case “a” (\$61).

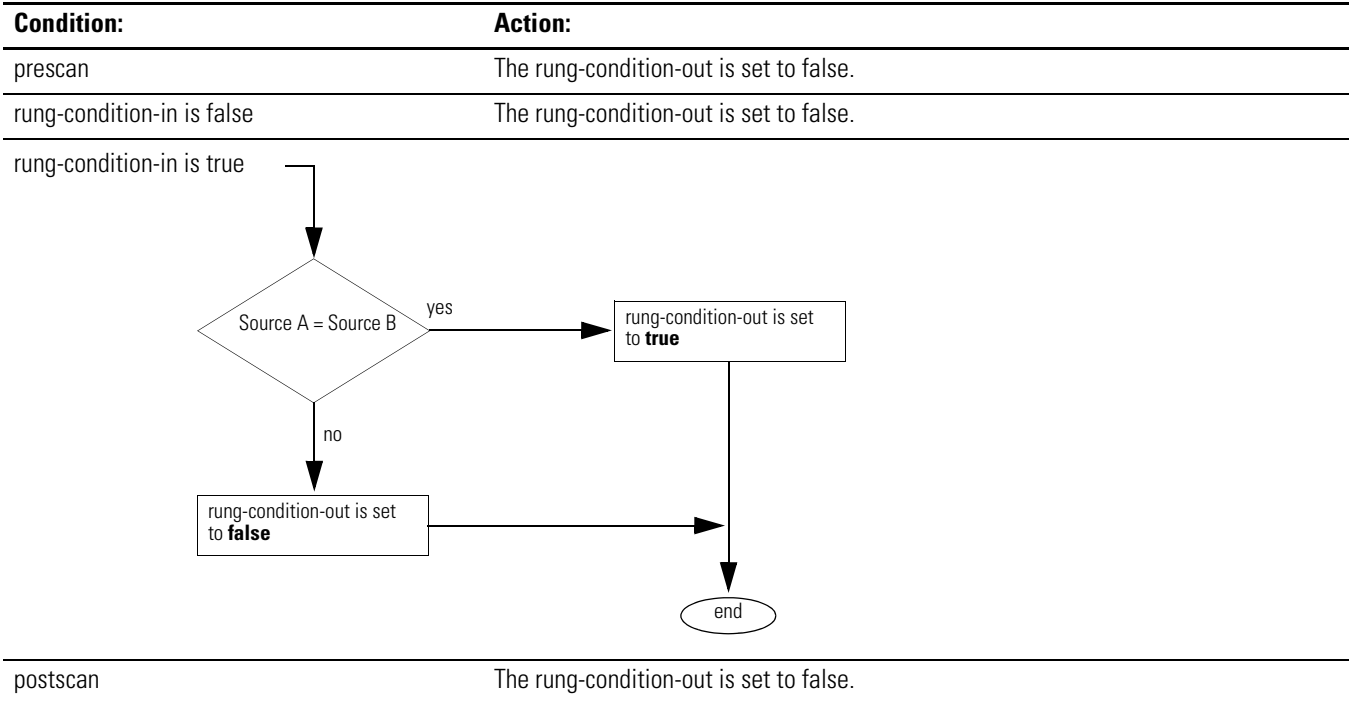
Arithmetic Status Flags: not affected

Fault Conditions: none

Execution:



Relay Ladder



Function Block

Condition:	Action:
prescan	No action taken.
instruction first scan	No action taken.
instruction first run	No action taken.
EnableIn is cleared	EnableOut is cleared.
EnableIn is set	The instruction executes. EnableOut is set.
postscan	No action taken.

Example: If *value_1* is equal to *value_2*, set *light_a*. If *value_1* is not equal to *value_2*, clear *light_a*.

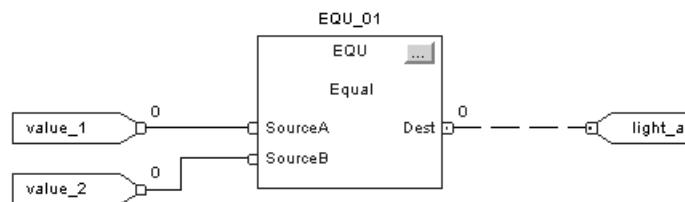
Relay Ladder



Structured Text

```
light_a := (value_1 = value_2);
```

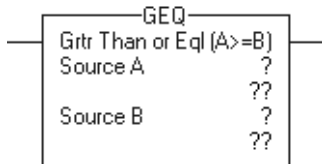
Function Block



Greater than or Equal to (GEQ)

The GEQ instruction tests whether Source A is greater than or equal to Source B.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Source A	SINT INT DINT REAL string	immediate tag	value to test against Source B
Source B	SINT INT DINT REAL string	immediate tag	value to test against Source A

- If you enter a SINT or INT tag, the value converts to a DINT value by sign-extension.
- String data types are:
 - default STRING data type
 - any new string data type that you create
- To test the characters of a string, enter a string tag for both Source A and Source B.

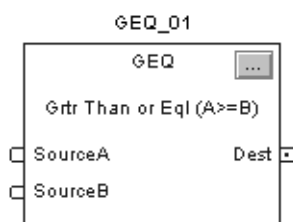


```
IF sourceA >= sourceB THEN
    <statements>;
```

Structured Text

Use adjacent greater than and equal signs “>=” as an operator within an expression. This expression evaluates whether *sourceA* is greater than or equal to *sourceB*.

See Appendix C for information on the syntax of expressions within structured text.



Function Block

Operand:	Type:	Format:	Description:
GEQ tag	FBD_COMPARE	structure	GEQ structure

FBD_COMPARE Structure

Input Parameter:	Data Type:	Description:
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set.
SourceA	REAL	Value to test against SourceB. Valid = any float
SourceB	REAL	Value to test against SourceA. Valid = any float
Output Parameter:	Data Type:	Description:
EnableOut	BOOL	The instruction produced a valid result.
Dest	BOOL	Result of the instruction. This is equivalent to rung-condition-out for the relay ladder GEQ instruction.

Description: The GEQ instruction tests whether Source A is greater than or equal to Source B.

When you compare strings:

- The hexadecimal values of the characters determine if one string is less than or greater than another string. For the hex code of a character, see the back cover of this manual.
- When the two strings are sorted as in a telephone directory, the order of the strings determines which one is greater.

ASCII Characters	Hex Codes
1ab	\$31\$61\$62
1b	\$31\$62
A	\$41
AB	\$41\$42
B	\$42
a	\$61
ab	\$61\$62

l
e
s
s
e
r

↑

g
r
e
a
t
e
r

↓

AB < B

a > B

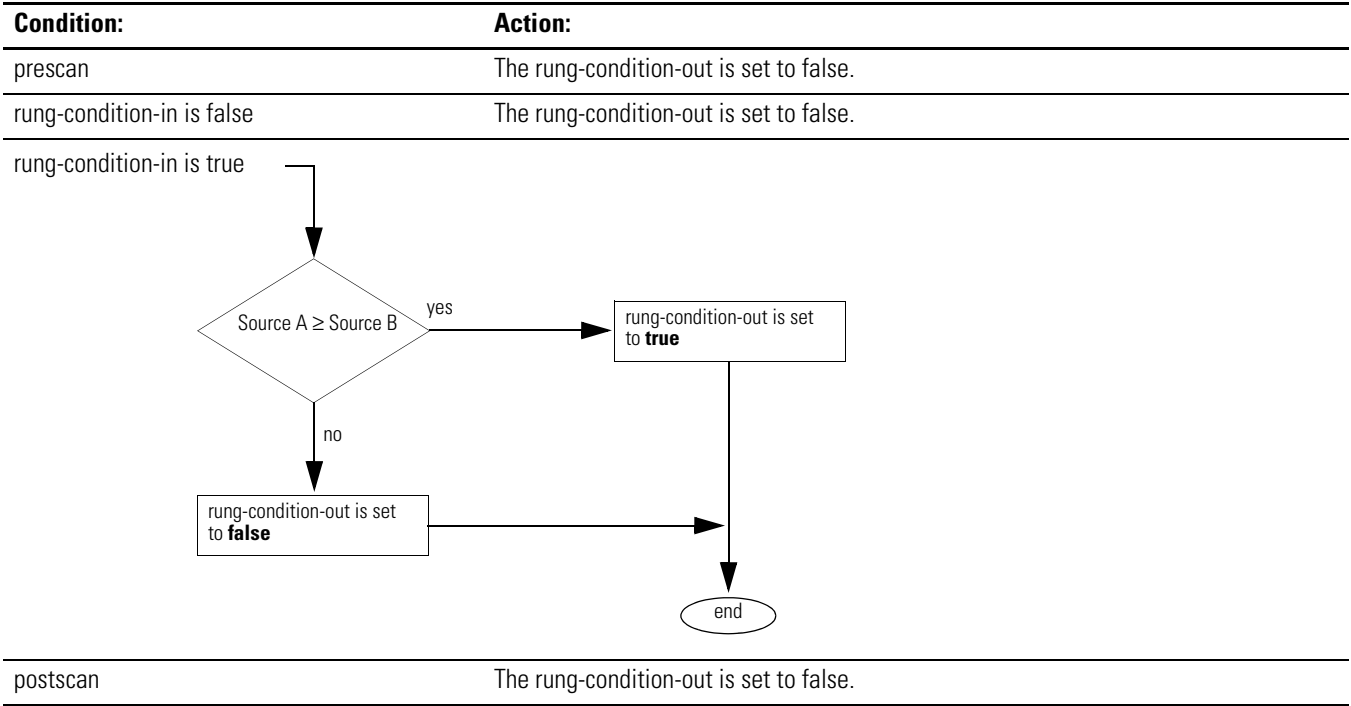
Arithmetic Status Flags: not affected

Fault Conditions: none

Execution:



Relay Ladder



Function Block

Condition:	Action:
prescan	No action taken.
instruction first scan	No action taken.
instruction first run	No action taken.
EnableIn is cleared	EnableOut is cleared.
EnableIn is set	The instruction executes. EnableOut is set.
postscan	No action taken.

Example: If *value_1* is greater than or equal to *value_2*, set *light_b*. If *value_1* is less than *value_2*, clear *light_b*.

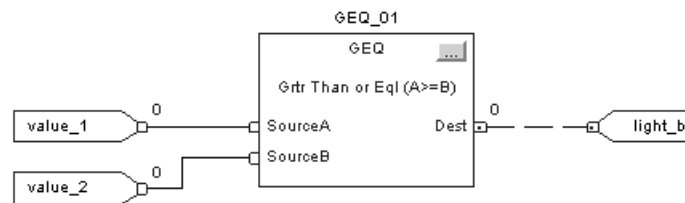
Relay Ladder



Structured Text

```
light_b := (value_1 >= value_2);
```

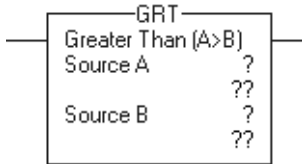
Function Block



Greater Than (GRT)

The GRT instruction tests whether Source A is greater than Source B.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Source A	SINT INT DINT REAL string	immediate tag	value to test against Source B
Source B	SINT INT DINT REAL string	immediate tag	value to test against Source A

- If you enter a SINT or INT tag, the value converts to a DINT value by sign-extension.
- String data types are:
 - default STRING data type
 - any new string data type that you create
- To test the characters of a string, enter a string tag for both Source A and Source B.

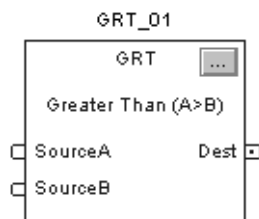


```
IF sourceA > sourceB THEN
  <statements>;
```

Structured Text

Use the greater than sign “>” as an operator within an expression. This expression evaluates whether *sourceA* is greater than *sourceB*.

See Appendix C for information on the syntax of expressions within structured text.



Function Block

Operand:	Type:	Format:	Description:
GRT tag	FBD_COMPARE	structure	GRT structure

FBD_COMPARE Structure

Input Parameter:	Data Type:	Description:
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set.
SourceA	REAL	Value to test against SourceB. Valid = any float
SourceB	REAL	Value to test against SourceA. Valid = any float
Output Parameter:	Data Type:	Description:
EnableOut	BOOL	The instruction produced a valid result.
Dest	BOOL	Result of the instruction. This is equivalent to rung-condition-out for the relay ladder GRT instruction.

Description: The GRT instruction tests whether Source A is greater than Source B.

When you compare strings:

- The hexadecimal values of the characters determine if one string is less than or greater than another string. For the hex code of a character, see the back cover of this manual.
- When the two strings are sorted as in a telephone directory, the order of the strings determines which one is greater.

ASCII Characters	Hex Codes
1ab	\$31\$61\$62
1b	\$31\$62
A	\$41
AB	\$41\$42
B	\$42
a	\$61
ab	\$61\$62

↑
l
e
s
s
e
r
 ↓
g
r
e
a
t
e
r

— AB < B
 — a > B

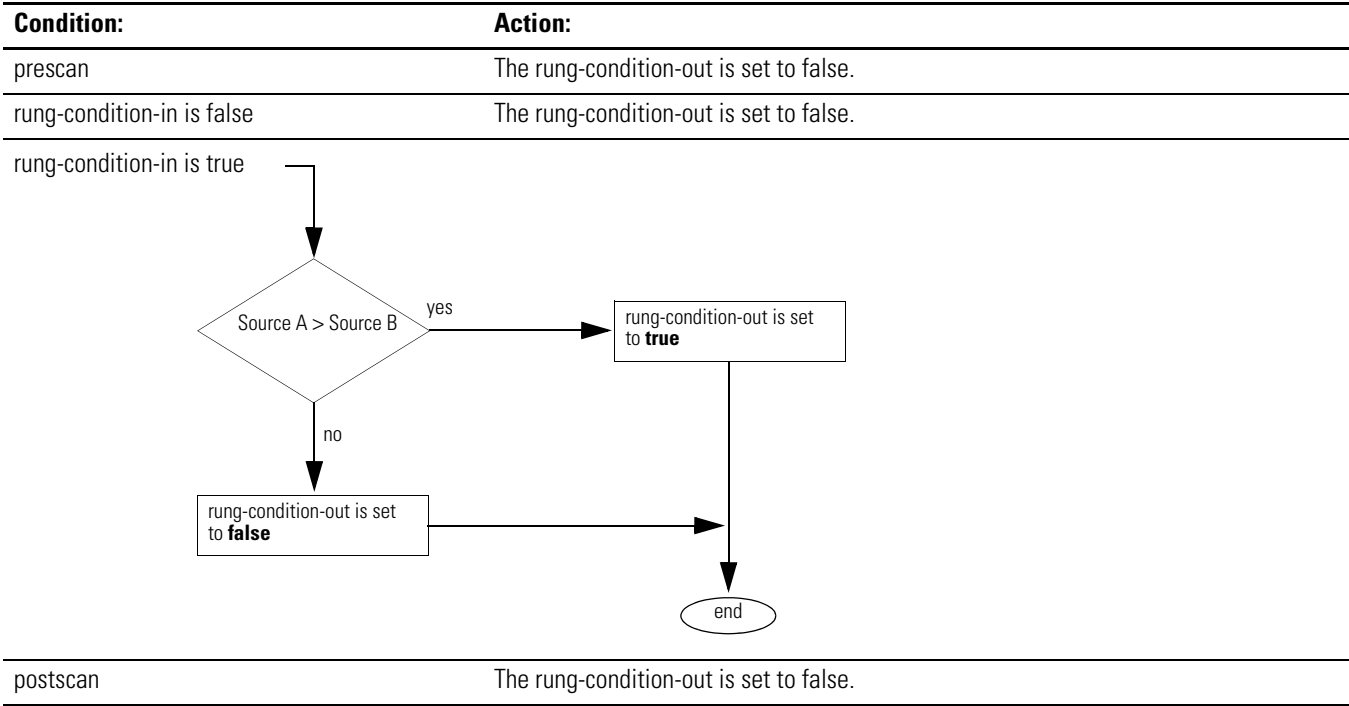
Arithmetic Status Flags: not affected

Fault Conditions: none

Execution:



Relay Ladder



Function Block

Condition:	Action:
prescan	No action taken.
instruction first scan	No action taken.
instruction first run	No action taken.
EnableIn is cleared	EnableOut is cleared.
EnableIn is set	The instruction executes. EnableOut is set.
postscan	No action taken.

Example: If *value_1* is greater than *value_2*, set *light_1*. If *value_1* is less than or equal to *value_2*, clear *light_1*.

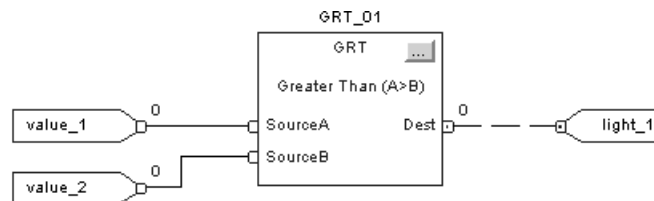
Relay Ladder



Structured Text

```
light_1 := (value_1 > value_2);
```

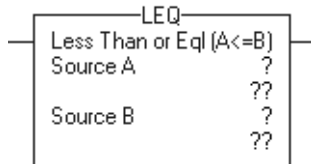
Function Block



Less Than or Equal to (LEQ)

The LEQ instruction tests whether Source A is less than or equal to Source B.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Source A	SINT INT DINT REAL string	immediate tag	value to test against Source B
Source B	SINT INT DINT REAL string	immediate tag	value to test against Source A

- If you enter a SINT or INT tag, the value converts to a DINT value by sign-extension.
- String data types are:
 - default STRING data type
 - any new string data type that you create
- To test the characters of a string, enter a string tag for both Source A and Source B.

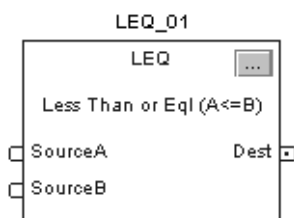


```
IF sourceA <= sourceB THEN
    <statements>;
```

Structured Text

Use adjacent less than and equal signs “<=” as an operator within an expression. This expression evaluates whether *sourceA* is less than or equal to *sourceB*.

See Appendix C for information on the syntax of expressions within structured text.



Function Block

Operand:	Type:	Format:	Description:
LEQ tag	FBD_COMPARE	structure	LEQ structure

FBD_COMPARE Structure

Input Parameter:	Data Type:	Description:
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set.
SourceA	REAL	Value to test against SourceB. Valid = any float
SourceB	REAL	Value to test against SourceA. Valid = any float
Output Parameter:	Data Type:	Description:
EnableOut	BOOL	The instruction produced a valid result.
Dest	BOOL	Result of the instruction. This is equivalent to rung-condition-out for the relay ladder LEQ instruction.

Description: The LEQ instruction tests whether Source A is less than or equal to Source B.

When you compare strings:

- The hexadecimal values of the characters determine if one string is less than or greater than another string. For the hex code of a character, see the back cover of this manual.
- When the two strings are sorted as in a telephone directory, the order of the strings determines which one is greater.

ASCII Characters	Hex Codes
1ab	\$31\$61\$62
1b	\$31\$62
A	\$41
AB	\$41\$42
B	\$42
a	\$61
ab	\$61\$62

l
e
s
s
e
r

↑

g
r
e
a
t
e
r

↓

AB < B

a > B

Arithmetic Status Flags: not affected

Fault Conditions: none

Execution:



Relay Ladder

Condition:	Action:
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
<div><div>rung-condition-in is true</div><div><div><div>Source A ≤ Source B</div><div>yes</div><div>rung-condition-out is set to true</div><div>no</div><div>rung-condition-out is set to false</div><div>end</div></div></div></div>	
postscan	The rung-condition-out is set to false.

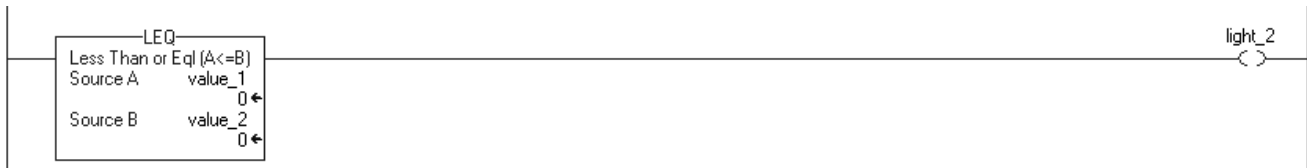


Function Block

Condition:	Action:
prescan	No action taken.
instruction first scan	No action taken.
instruction first run	No action taken.
EnableIn is cleared	EnableOut is cleared.
EnableIn is set	The instruction executes. EnableOut is set.
postscan	No action taken.

Example: If *value_1* is less than or equal to *value_2*, set *light_2*. If *value_1* is greater than *value_2*, clear *light_2*.

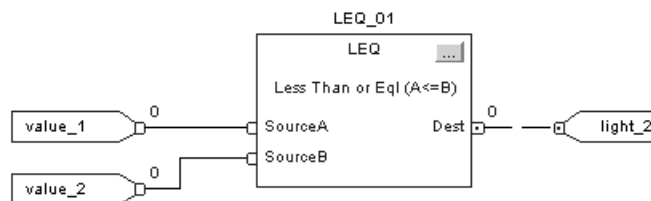
Relay Ladder



Structured Text

```
light_2 := (value_1 <= value_2);
```

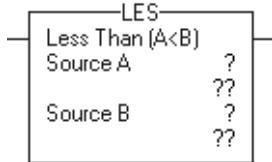
Function Block



Less Than (LES)

The LES instruction tests whether Source A is less than Source B.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Source A	SINT INT DINT REAL string	immediate tag	value to test against Source B
Source B	SINT INT DINT REAL string	immediate tag	value to test against Source A

- If you enter a SINT or INT tag, the value converts to a DINT value by sign-extension.
- String data types are:
 - default STRING data type
- any new string data type that you create
- To test the characters of a string, enter a string tag for both Source A and Source B.

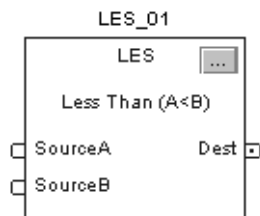


```
IF sourceA < sourceB THEN
  <statements>;
```

Structured Text

Use the less than sign “<” as an operator within an expression. This expression evaluates whether *sourceA* is less than *sourceB*.

See Appendix C for information on the syntax of expressions within structured text.



Function Block

Operand:	Type:	Format:	Description:
LES tag	FBD_COMPARE	structure	LES structure

FBD_COMPARE Structure

Input Parameter:	Data Type:	Description:
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set.
SourceA	REAL	Value to test against SourceB. Valid = any float
SourceB	REAL	Value to test against SourceA. Valid = any float
Output Parameter:	Data Type:	Description:
EnableOut	BOOL	The instruction produced a valid result.
Dest	BOOL	Result of the instruction. This is equivalent to rung-condition-out for the relay ladder LES instruction.

Description: The LES instruction tests whether Source A is less than Source B.

When you compare strings:

- The hexadecimal values of the characters determine if one string is less than or greater than another string. For the hex code of a character, see the back cover of this manual.
- When the two strings are sorted as in a telephone directory, the order of the strings determines which one is greater.

ASCII Characters	Hex Codes
1ab	\$31\$61\$62
1b	\$31\$62
A	\$41
AB	\$41\$42
B	\$42
a	\$61
ab	\$61\$62

↑
 l
e
s
s
e
r
 ↓

↓
 g
r
e
a
t
e
r
 ↓

— AB < B
 — a > B

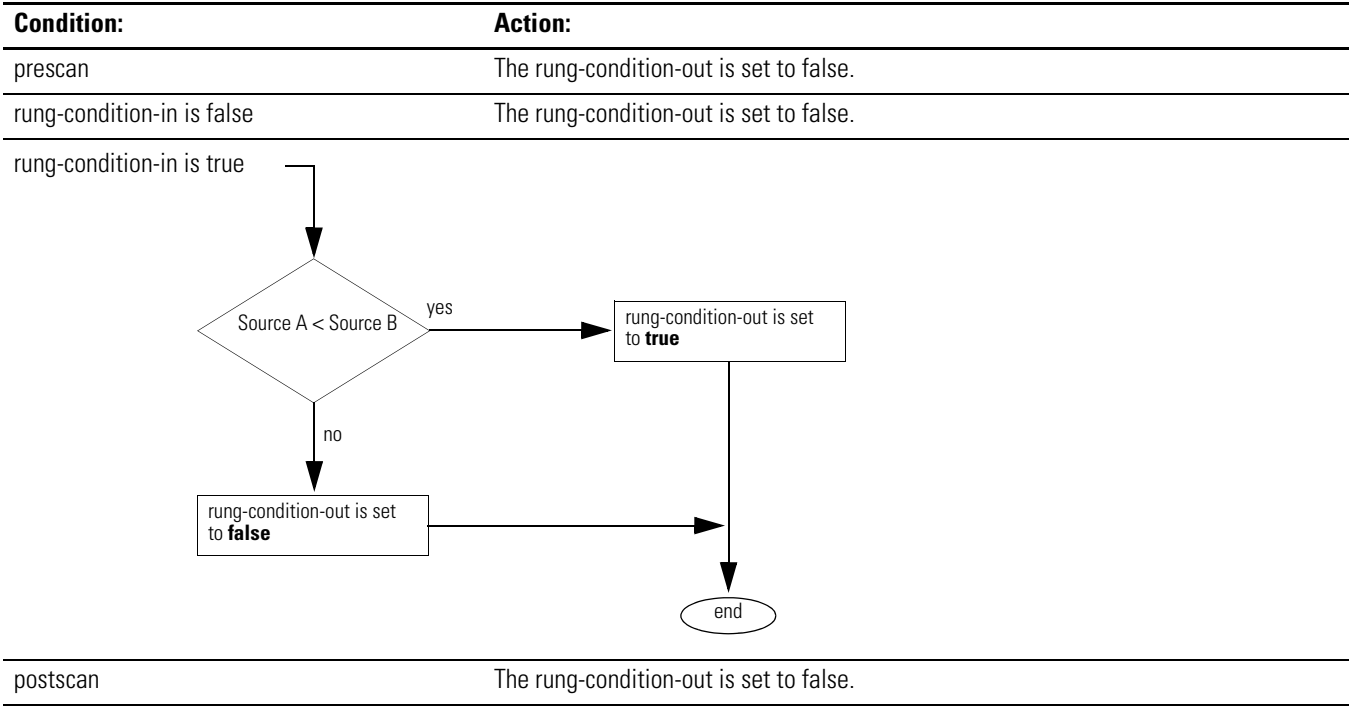
Arithmetic Status Flags: not affected

Fault Conditions: none

Execution:



Relay Ladder



Function Block

Condition:	Action:
prescan	No action taken.
instruction first scan	No action taken.
instruction first run	No action taken.
EnableIn is false	EnableOut is cleared.
EnableIn is true	The instruction executes. EnableOut is set.
postscan	No action taken.

Example: If *value_1* is less than *value_2*, set *light_3*. If *value_1* is greater than or equal to *value_2*, clear *light_3*.

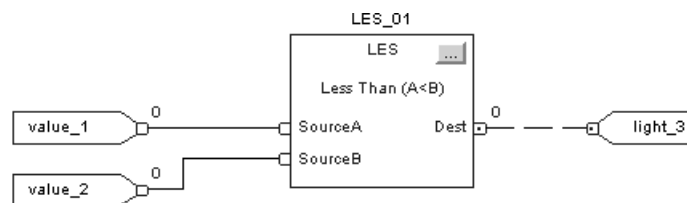
Relay Ladder



Structured Text

```
light_3 := (value_1 < value_2);
```

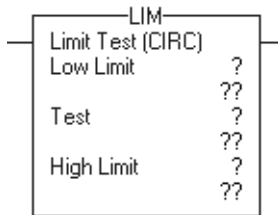
Function Block



Limit (LIM)

The LIM instruction tests whether the Test value is within the range of the Low Limit to the High Limit.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Low limit	SINT INT DINT REAL	immediate tag	value of lower limit
A SINT or INT tag converts to a DINT value by sign-extension.			
Test	SINT INT DINT REAL	immediate tag	value to test
A SINT or INT tag converts to a DINT value by sign-extension.			
High limit	SINT INT DINT REAL	immediate tag	value of upper limit
A SINT or INT tag converts to a DINT value by sign-extension.			



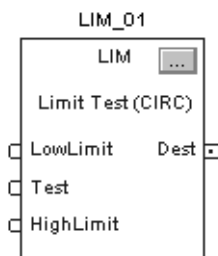
Structured Text

Structured text does not have a LIM instruction, but you can achieve the same results using structured text.

```
IF (LowLimit <= HighLimit AND
    (Test >= LowLimit AND Test <= HighLimit)) OR
    (LowLimit >= HighLimit AND
    (Test <= LowLimit OR Test >= HighLimit)) THEN
```

```
    <statement>;
```

```
END_IF;
```



Function Block

Operand:	Type:	Format:	Description:
LIM tag	FBD_LIMIT	structure	LIM structure

FBD_LIMIT Structure

Input Parameter:	Data Type:	Description:
EnableIn	BOOL	If cleared, the instruction does not execute and outputs are not updated. If set, the instruction executes as described under Execution. Default is set.
LowLimit	REAL	Value of lower limit. Valid = any float
Test	REAL	Value to test against limits. Valid = any float
Output Parameter:	Data Type:	Description:
EnableOut	BOOL	The instruction produced a valid result.
Dest	BOOL	Result of the instruction. This is equivalent to rung-condition-out for the relay ladder LIM instruction.
HighLimit	REAL	Value of upper limit. Valid = any float

Description: The LIM instruction tests whether the Test value is within the range of the Low Limit to the High Limit.

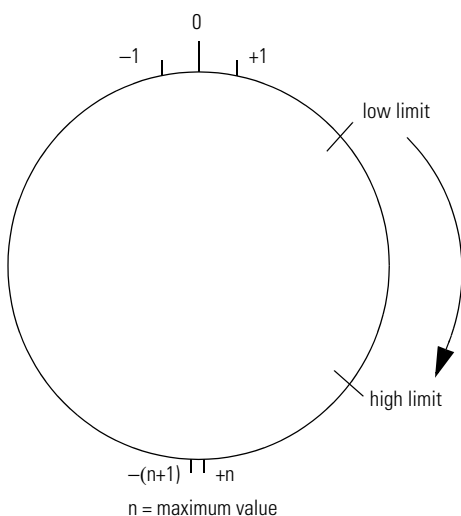
If Low Limit:	And Test value is:	The rung-condition-out is:
\leq High Limit	equal to or between limits	true
	not equal to or outside limits	false
\geq High Limit	equal to or outside limits	true
	not equal to or inside limits	false

Signed integers “roll over” from the maximum positive number to the maximum negative number when the most significant bit is set. For example, in 16-bit integers (INT type), the maximum positive integer is 32,767, which is represented in hexadecimal as 16#7FFF (bits 0 through 14 are all set). If you increment that number by one, the result is 16#8000 (bit 15 is set). For signed integers, hexadecimal 16#8000 is equal to -32,768 decimal. Incrementing from this point on until all 16 bits are set ends up at 16#FFFF, which is equal to -1 decimal.

This can be shown as a circular number line (see the following diagrams). The LIM instruction starts at the Low Limit and increments clockwise until it reaches the High Limit. Any Test value in the clockwise range from the Low Limit to the High Limit sets the rung-condition-out to true. Any Test value in the clockwise range from the High Limit to the Low Limit sets the rung-condition-out to false.

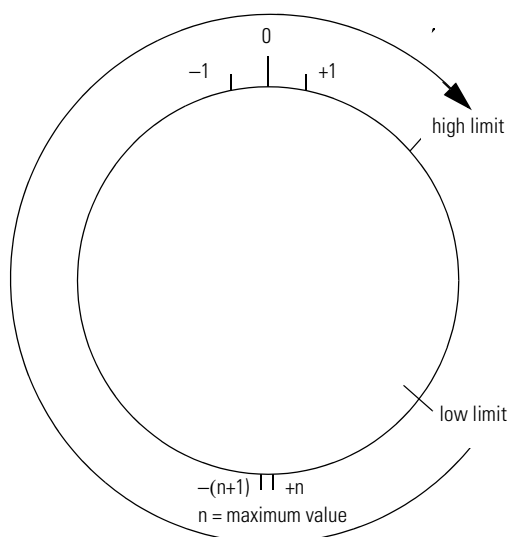
Low Limit \leq High Limit

The instruction is true if the test value is equal to or between the low and high limit



Low Limit \geq High Limit

The instruction is true if the test value is equal to or outside the low and high limit



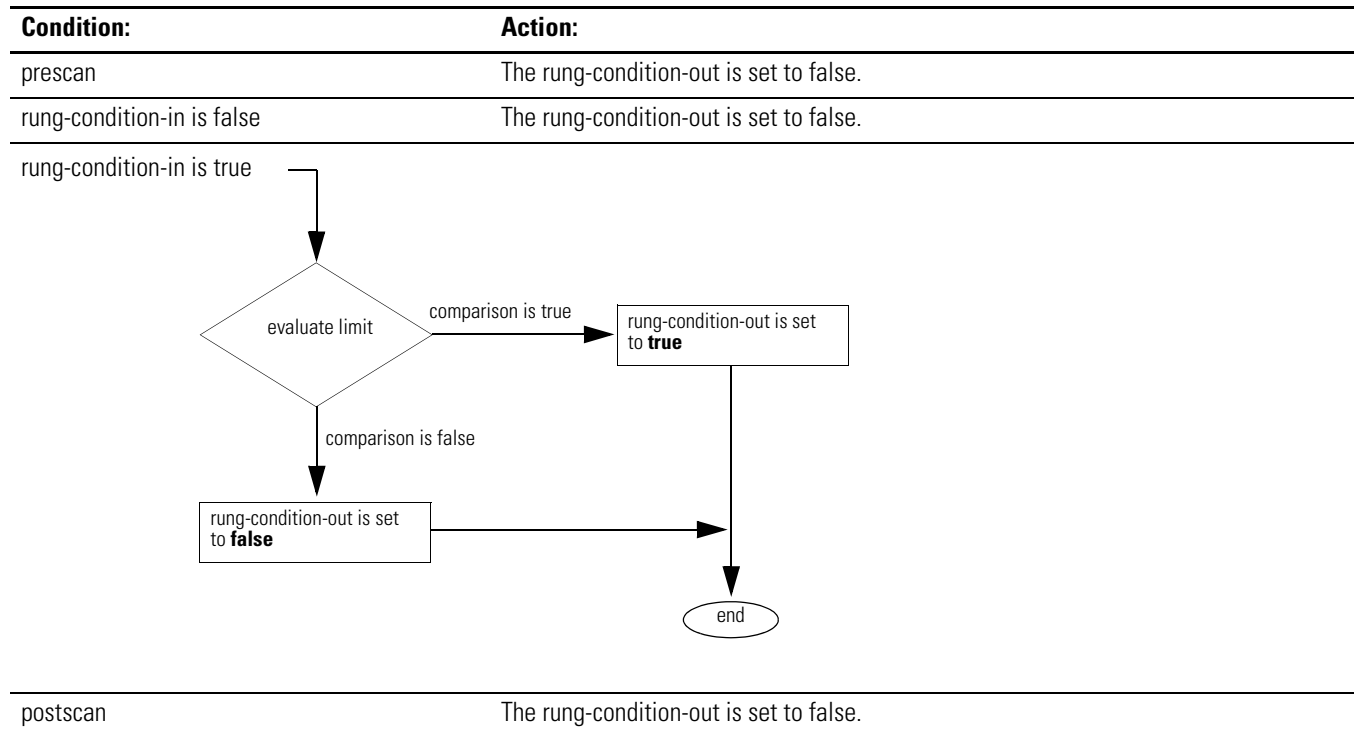
Arithmetic Status Flags: not affected

Fault Conditions: none

Execution:



Relay Ladder

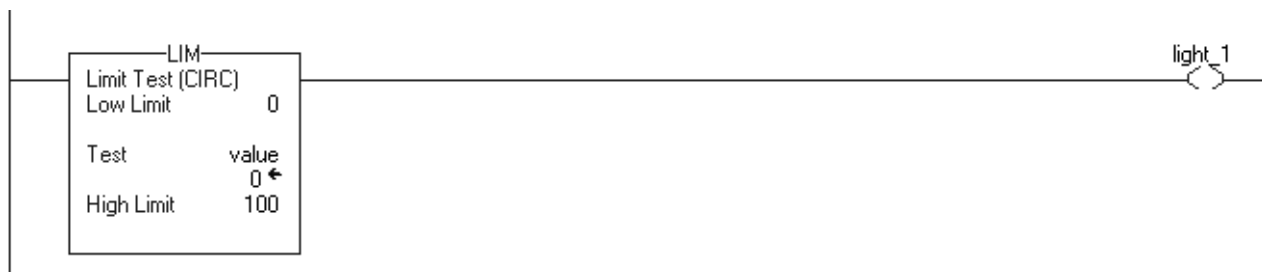


Function Block

Condition:	Action:
prescan	No action taken.
instruction first scan	No action taken.
instruction first run	No action taken.
EnableIn is cleared	EnableOut is cleared, the instruction does nothing, and the outputs are not updated.
EnableIn is set	The instruction executes. EnableOut is set.
postscan	No action taken.

Example 1: Low Limit ≤ High Limit:

When $0 \leq \text{value} \leq 100$, set *light_1*. If *value* < 0 or *value* > 100, clear *light_1*.

Relay Ladder**Structured Text**

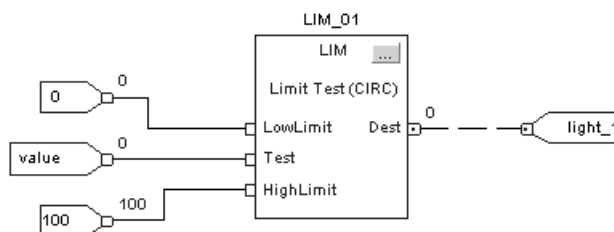
```
IF (value <= 100 AND (value >= 0 AND value <= 100)) OR
   (value >= 100 AND value <= 0 OR value >= 100)) THEN

    light_1 := 1;

ELSE

    light_1 := 0;

END_IF;
```

Function Block

Example 2: Low Limit \geq High Limit:

When $value \geq 0$ or $value \leq -100$, set *light_1*. If $value < 0$ or $value > -100$, clear *light_1*.

Relay Ladder



Structured Text

```
IF (0 <= -100 AND value >= 0 AND value <= -100)) OR
   (0 >= -100 AND (value <= 0 OR value >= -100)) THEN

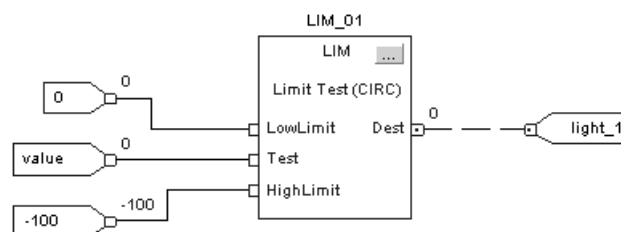
    light_1 := 1;

ELSE

    light_1 := 0;

END_IF;
```

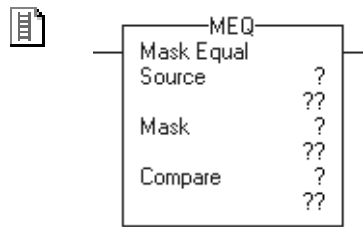
Function Block



Mask Equal to (MEQ)

The MEQ instruction passes the Source and Compare values through a Mask and compares the results.

Operands:



Relay Ladder

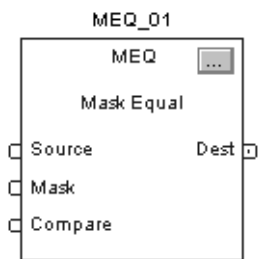
Operand:	Type:	Format:	Description:
Source	SINT INT DINT	immediate tag	value to test against Compare
A SINT or INT tag converts to a DINT value by zero-fill.			
Mask	SINT INT DINT	immediate tag	defines which bits to block or pass
A SINT or INT tag converts to a DINT value by zero-fill.			
Compare	SINT INT DINT	immediate tag	value to test against Source
A SINT or INT tag converts to a DINT value by zero-fill.			



Structured Text

Structured text does not have an MEQ instruction, but you can achieve the same results using structured text.

```
IF (Source AND Mask) = (Compare AND Mask) THEN  
  
    <statement>;  
  
END_IF;
```



Function Block

Operand:	Type:	Format:	Description:
MEQ tag	FBD_MASK_EQUAL	structure	MEQ structure

FBD_MASK_EQUAL Structure

Input Parameter:	Data Type:	Description:
EnableIn	BOOL	If cleared, the instruction does not execute and outputs are not updated. If set, the instruction executes as described under Execution. Default is set.
Source	DINT	Value to test against Compare. Valid = any integer
Mask	DINT	Defines which bits to block (mask). Valid = any integer
Compare	DINT	Compare value. Valid = any integer
Output Parameter:	Data Type:	Description:
EnableOut	BOOL	The instruction produced a valid result.
Dest	BOOL	Result of the instruction. This is equivalent to rung-condition-out for the relay ladder MEQ instruction.

Description: A “1” in the mask means the data bit is passed. A “0” in the mask means the data bit is blocked. Typically, the Source, Mask, and Compare values are all the same data type.

If you mix integer data types, the instruction fills the upper bits of the smaller integer data types with 0s so that they are the same size as the largest data type.

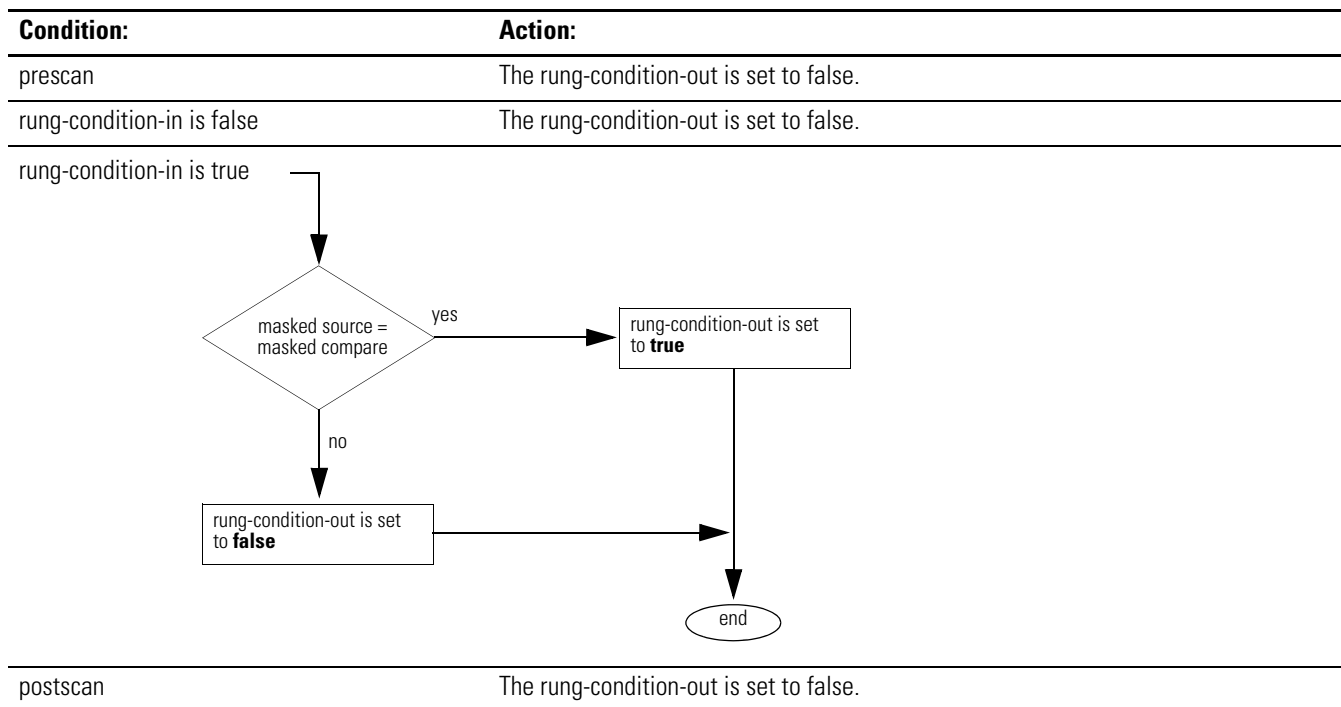
Entering an immediate mask value

When you enter a mask, the programming software defaults to decimal values. If you want to enter a mask using another format, precede the value with the correct prefix.

Prefix:	Description:
16#	hexadecimal for example; 16#0F0F
8#	octal for example; 8#16
2#	binary for example; 2#00110011

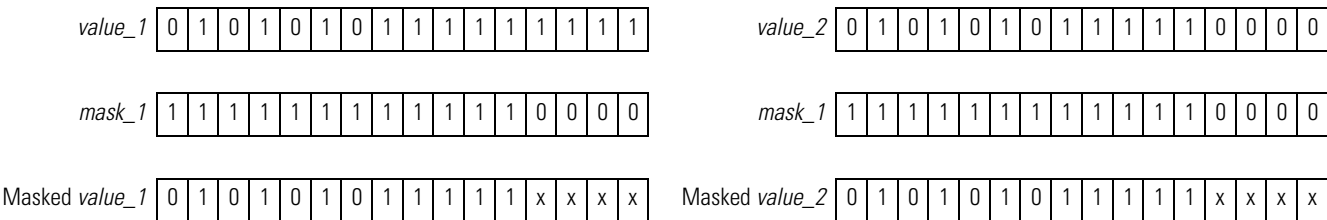
Arithmetic Status Flags: not affected

Fault Conditions: none

Execution:**Relay Ladder****Function Block**

Condition:	Action:
prescan	No action taken.
instruction first scan	No action taken.
instruction first run	No action taken.
EnableIn is cleared	EnableOut is cleared, the instruction does nothing, and the outputs are not updated.
EnableIn is set	The instruction executes. EnableOut is set.
postscan	No action taken.

Example 1: If the masked *value_1* is equal to the masked *value_2*, set *light_1*. If the masked *value_1* is not equal to the masked *value_2*, clear *light_1*. This example shows that the masked values are equal. A 0 in the mask restrains the instruction from comparing that bit (shown by x in the example).



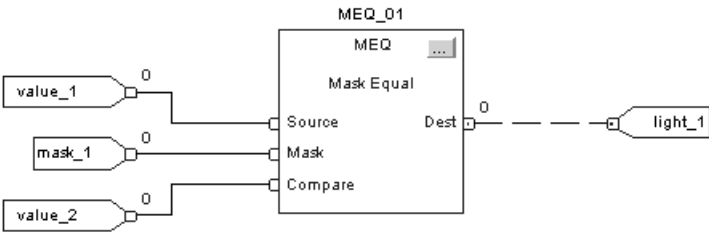
Relay Ladder



Structured Text

```
light_1 := ((value_1 AND mask_1)=(value_2 AND mask_2));
```

Function Block



Example 2: If the masked *value_1* is equal to the masked *value_2*, set *light_1*. If the masked *value_1* is not equal to the masked *value_2*, clear *light_1*. This example shows that the masked values are not equal. A 0 in the mask restrains the instruction from comparing that bit (shown by x in the example).

<i>value_1</i>	0	1	0	1	0	1	0	1	1	1	1	1	1	1	1	1
<i>mask_1</i>	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
Masked <i>value_1</i>	x	x	x	x	x	x	x	x	x	x	x	1	1	1	1	1
<i>value_2</i>	0	1	0	1	0	1	0	1	1	1	1	1	0	0	0	0
<i>mask_2</i>	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
Masked <i>value_2</i>	x	x	x	x	x	x	x	x	x	x	x	x	0	0	0	0

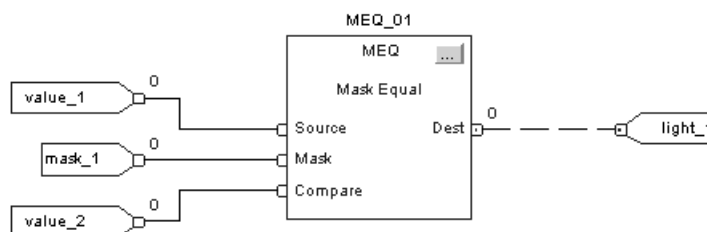
Relay Ladder



Structured Text

```
light_1 := ((value_1 AND mask_1)=(value_2 AND mask_2));
```

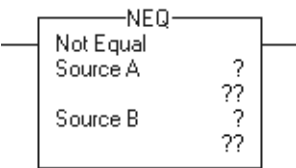
Function Block



Not Equal to (NEQ)

The NEQ instruction tests whether Source A is not equal to Source B.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Source A	SINT INT DINT REAL string	immediate tag	value to test against Source B
Source B	SINT INT DINT REAL string	immediate tag	value to test against Source A

- If you enter a SINT or INT tag, the value converts to a DINT value by sign-extension.
- String data types are:
 - default STRING data type
 - any new string data type that you create
- To test the characters of a string, enter a string tag for both Source A and Source B.



```
IF sourceA <> sourceB THEN
    <statements>;
```

Structured Text

Use the less than and greater than signs “<>” together as an operator within an expression. This expression evaluates whether *sourceA* is not equal to *sourceB*.

See Appendix C for information on the syntax of expressions within structured text.



Function Block

Operand:	Type:	Format:	Description:
NEQ tag	FBD_COMPARE	structure	NEQ structure

FBD_COMPARE Structure

Input Parameter:	Data Type:	Description:
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set.
SourceA	REAL	Value to test against SourceB. Valid = any float
SourceB	REAL	Value to test against SourceA. Valid = any float
Output Parameter:	Data Type:	Description:
EnableOut	BOOL	The instruction produced a valid result.
Dest	BOOL	Result of the instruction. This is equivalent to rung-condition-out for the relay ladder NEQ instruction.

Description: The NEQ instruction tests whether Source A is not equal to Source B.

When you compare strings:

- Strings are not equal if any of their characters do not match.
- ASCII characters are case sensitive. Upper case “A” (\$41) is *not* equal to lower case “a” (\$61).

ASCII Characters	Hex Codes
1ab	\$31\$61\$62
1b	\$31\$62
A	\$41
AB	\$41\$42
B	\$42
a	\$61
ab	\$61\$62

↑
 l
e
s
s
e
r
 ↓
 g
r
e
a
t
e
r

— AB < B
 — a > B

Arithmetic Status Flags: not affected

Fault Conditions: none

Execution:



Relay Ladder

Condition:	Action:
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
<div><p>rung-condition-in is true</p><pre>graph TD; Start([rung-condition-in is true]) --> Decision{Source A = Source B}; Decision -- yes --> SetTrue[rung-condition-out is set to true]; Decision -- no --> SetFalse[rung-condition-out is set to false]; SetTrue --> End([end]); SetFalse --> End;</pre></div>	
postscan	The rung-condition-out is set to false.



Function Block

Condition:	Action:
prescan	No action taken.
instruction first scan	No action taken.
instruction first run	No action taken.
EnableIn is cleared	EnableOut is cleared.
EnableIn is set	The instruction executes. EnableOut is set.
postscan	No action taken.

Example: If *value_1* is not equal to *value_2*, set *light_4*. If *value_1* is equal to *value_2*, clear *light_4*.

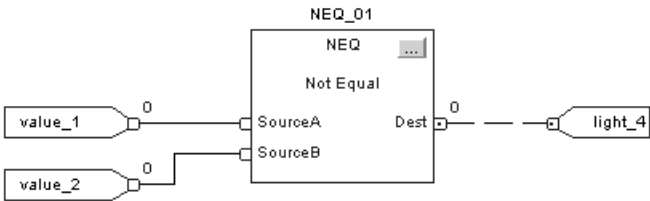
Relay Ladder



Structured Text

```
light_4 := (value_1 <> value_2);
```

Function Block



Notes:

Compute/Math Instructions

(CPT, ADD, SUB, MUL, DIV, MOD, SQR, SQRT, NEG, ABS)

Introduction

The compute/math instructions evaluate arithmetic operations using an expression or a specific arithmetic instruction.

If you want to:	Use this instruction:	Available in these languages:	See page:
evaluate an expression	CPT	relay ladder structured text ⁽¹⁾	5-2
add two values	ADD	relay ladder structured text ⁽²⁾ function block	5-6
subtract two values	SUB	relay ladder structured text ⁽²⁾ function block	5-9
multiply two values	MUL	relay ladder structured text ⁽²⁾ function block	5-12
divide two values	DIV	relay ladder structured text ⁽²⁾ function block	5-15
determine the remainder after one value is divided by another	MOD	relay ladder structured text ⁽²⁾ function block	5-19
calculate the square root of a value	SQR SQRT ⁽³⁾	relay ladder structured text function block	5-23
take the opposite sign of a value	NEG	relay ladder structured text ⁽²⁾ function block	5-26
take the absolute value of a value	ABS	relay ladder structured text function block	5-29

⁽¹⁾ There is no equivalent structured text instruction. Use other structured text programming to achieve the same result. See the description for the instruction.

⁽²⁾ There is no equivalent structured text instruction. Use the operator in an expression.

⁽³⁾ Structured text only.

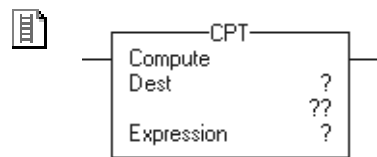
You can mix data types, but loss of accuracy and rounding error might occur and the instruction takes more time to execute. Check the S:V bit to see whether the result was truncated.

For relay ladder instructions, **bold** data types indicate optimal data types. An instruction executes faster and requires less memory if all the operands of the instruction use the same optimal data type, typically DINT or REAL.

Compute (CPT)

The CPT instruction performs the arithmetic operations you define in the expression.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Destination	SINT INT DINT REAL	tag	tag to store the result
Expression	SINT INT DINT REAL	immediate tag	an expression consisting of tags and/or immediate values separated by operators

A SINT or INT tag converts to a DINT value by sign-extension.



Structured Text

Structured text does not have a CPT instruction, but you can achieve the same results using an assignment and expression.

```
destination := numeric_expression;
```

See Appendix C for information on the syntax of assignments and expressions within structured text.

Description: The CPT instruction performs the arithmetic operations you define in the expression. When enabled, the CPT instruction evaluates the expression and places the result in the Destination.

The execution of a CPT instruction is slightly slower and uses more memory than the execution of the other compute/math instructions. The advantage of the CPT instruction is that it allows you to enter complex expressions in one instruction.

TIP

There is *no limit* to the length of an expression.

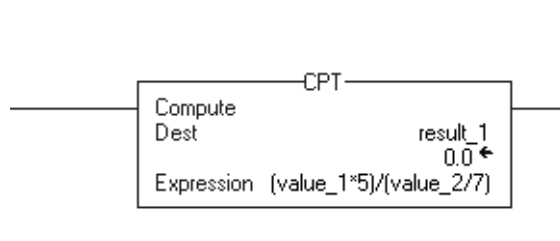
Arithmetic Status Flags: Arithmetic status flags are affected.

Fault Conditions: none

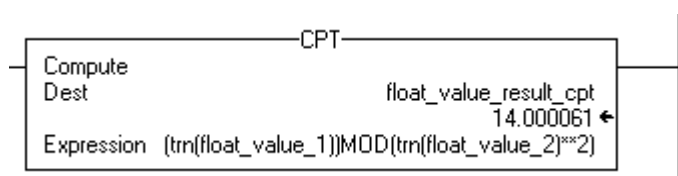
Execution:

Condition:	Relay Ladder Action:
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	The instruction evaluates the Expression and places the result in the Destination. The rung-condition-out is set to true.
postscan	The rung-condition-out is set to false.

Example 1: When enabled, the CPT instruction evaluates *value_1* multiplied by 5 and divides that result by the result of *value_2* divided by 7 and places the final result in *result_1*.



Example 2: When enabled, the CPT instruction truncates *float_value_1* and *float_value_2*, raises the truncated *float_value_2* to the power of two and divides the truncated *float_value_1* by that result, and stores the remainder after the division in *float_value_result_cpt*.



Valid operators

Operator:	Description:	Optimal:
+	add	DINT, REAL
-	subtract/negate	DINT, REAL
*	multiply	DINT, REAL
/	divide	DINT, REAL
**	exponent (x to y)	DINT, REAL
ABS	absolute value	DINT, REAL
ACS	arc cosine	REAL
AND	bitwise AND	DINT
ASN	arc sine	REAL
ATN	arc tangent	REAL
COS	cosine	REAL
DEG	radians to degrees	DINT, REAL
FRD	BCD to integer	DINT
LN	natural log	REAL

Operator:	Description:	Optimal:
LOG	log base 10	REAL
MOD	modulo-divide	DINT, REAL
NOT	bitwise complement	DINT
OR	bitwise OR	DINT
RAD	degrees to radians	DINT, REAL
SIN	sine	REAL
SQR	square root	DINT, REAL
TAN	tangent	REAL
TOD	integer to BCD	DINT
TRN	truncate	DINT, REAL
XOR	bitwise exclusive OR	DINT

Formatting expressions

For each operator that you use in an expression, you have to provide one or two operands (tags or immediate values). Use the following table to format operators and operands within an expression:

For operators that operate on:	Use this format:	Examples:
one operand	operator(operand)	ABS(<i>tag_a</i>)
two operands	operand_a operator operand_b	<ul style="list-style-type: none"> <i>tag_b</i> + 5 <i>tag_c</i> AND <i>tag_d</i> (<i>tag_e</i> ** 2) MOD (<i>tag_f</i> / <i>tag_g</i>)

Determining the order of operation

The operations you write into the expression are performed by the instruction in a prescribed order, not necessarily the order you write them. You can override the order of operation by grouping terms within parentheses, forcing the instruction to perform an operation within the parentheses ahead of other operations.

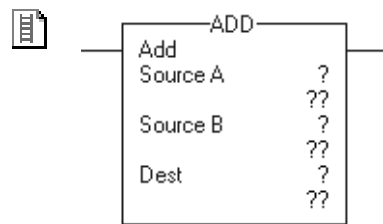
Operations of equal order are performed from left to right.

Order:	Operation:
1.	()
2.	ABS, ACS, ASN, ATN, COS, DEG, FRD, LN, LOG, RAD, SIN, SQR, TAN, TOD, TRN
3.	**
4.	– (negate), NOT
5.	*, /, MOD
6.	– (subtract), +
7.	AND
8.	XOR
9.	OR

Add (ADD)

The ADD instruction adds Source A to Source B and places the result in the Destination.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Source A	SINT INT DINT REAL	immediate tag	value to add to Source B
A SINT or INT tag converts to a DINT value by sign-extension.			
Source B	SINT INT DINT REAL	immediate tag	value to add to Source A
A SINT or INT tag converts to a DINT value by sign-extension.			
Destination	SINT INT DINT REAL	tag	tag to store the result



```

dest := sourceA + sourceB;

```

Structured Text

Use the plus sign “+” as an operator within an expression. This expression adds *sourceA* to *sourceB* and stores the result in *dest*.

See Appendix C for information on the syntax of expressions within structured text.



Function Block

Operand:	Type:	Format:	Description:
ADD tag	FBD_MATH	structure	ADD structure

FBD_MATH Structure

Input Parameter:	Data Type:	Description:
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set.
SourceA	REAL	Value to add to SourceB. Valid = any float
SourceB	REAL	Value to add to SourceA. Valid = any float
Output Parameter:	Data Type:	Description:
EnableOut	BOOL	The instruction produced a valid result.
Dest	REAL	Result of the math instruction. Arithmetic status flags are set for this output.

Description: The ADD instruction adds Source A to Source B and places the result in the Destination.

Arithmetic Status Flags: Arithmetic status flags are affected.

Fault Conditions: none

Execution:



Relay Ladder

Condition:	Action:
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	Destination = Source A + Source B The rung-condition-out is set to true.
postscan	The rung-condition-out is set to false.

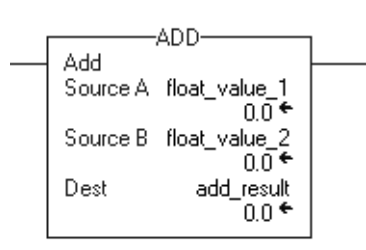


Function Block

Condition:	Action:
prescan	No action taken.
instruction first scan	No action taken.
instruction first run	No action taken.
EnableIn is cleared	EnableOut is cleared.
EnableIn is set	The instruction executes. EnableOut is set.
postscan	No action taken.

Example: Add *float_value_1* to *float_value_2* and place the result in *add_result*.

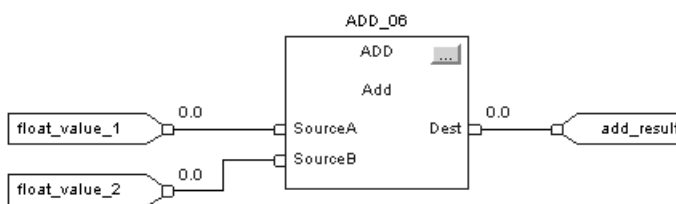
Relay Ladder



Structured Text

```
add_result := float_value_1 + float_value_2;
```

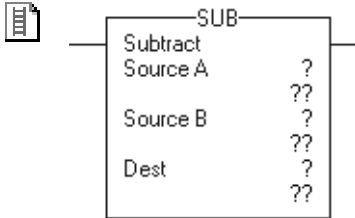
Function Block



Subtract (SUB)

The SUB instruction subtracts Source B from Source A and places the result in the Destination.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Source A	SINT INT DINT REAL	immediate tag	value from which to subtract Source B
A SINT or INT tag converts to a DINT value by sign-extension.			
Source B	SINT INT DINT REAL	immediate tag	value to subtract from Source A
A SINT or INT tag converts to a DINT value by sign-extension.			
Destination	SINT INT DINT REAL	tag	tag to store the result

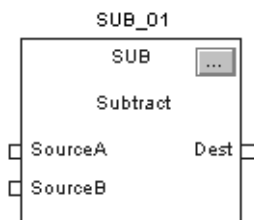


```
dest := sourceA - sourceB;
```

Structured Text

Use the minus sign “-” as an operator in an expression. This expression subtracts *sourceB* from *sourceA* and stores the result in *dest*.

See Appendix C for information on the syntax of expressions within structured text.



Function Block

Operand:	Type:	Format:	Description:
SUB tag	FBD_MATH	structure	SUB structure

FBD_MATH Structure

Input Parameter:	Data Type:	Description:
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set.
SourceA	REAL	Value from which to subtract SourceB. Valid = any float
SourceB	REAL	Value to subtract from SourceA. Valid = any float
Output Parameter:	Data Type:	Description:
EnableOut	BOOL	The instruction produced a valid result.
Dest	REAL	Result of the math instruction. Arithmetic status flags are set for this output.

Description: The SUB instruction subtracts Source B from Source A and places the result in the Destination.

Arithmetic Status Flags: Arithmetic status flags are affected.

Fault Conditions: none

Execution:

**Relay Ladder**

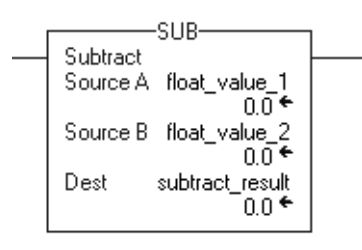
Condition:	Action:
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	Destination = Source B - Source A The rung-condition-out is set to true.
postscan	The rung-condition-out is set to false.

**Function Block**

Condition:	Action:
prescan	No action taken.
instruction first scan	No action taken.
instruction first run	No action taken.
EnableIn is cleared	EnableOut is cleared.
EnableIn is set	The instruction executes. EnableOut is set.
postscan	No action taken.

Example: Subtract *float_value_2* from *float_value_1* and place the result in *subtract_result*.

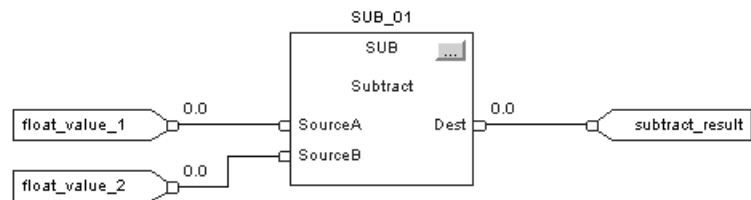
Relay Ladder



Structured Text

```
subtract_result := float_value_1 - float_value_2;
```

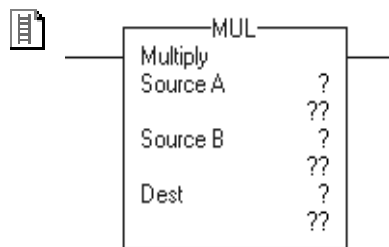
Function Block



Multiply (MUL)

The MUL instruction multiplies Source A with Source B and places the result in the Destination.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Source A	SINT INT DINT REAL	immediate tag	value of the multiplicand
A SINT or INT tag converts to a DINT value by sign-extension.			
Source B	SINT INT DINT REAL	immediate tag	value of the multiplier
A SINT or INT tag converts to a DINT value by sign-extension.			
Destination	SINT INT DINT REAL	tag	tag to store the result

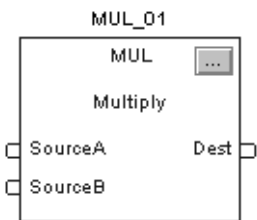


```
dest := sourceA * sourceB;
```

Structured Text

Use the multiply sign “*” as an operator in an expression. This expression multiplies *sourceA* by *sourceB* and stores the result in *dest*.

See Appendix C for information on the syntax of expressions within structured text.



Function Block

Operand:	Type:	Format:	Description:
MUL tag	FBD_MATH	structure	MUL structure

FBD_MATH Structure

Input Parameter:	Data Type:	Description:
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set.
Source A	REAL	Value of the multiplicand. Valid = any float
Source B	REAL	Value of the multiplier. Valid = any float
Output Parameter:	Data Type:	Description:
EnableOut	BOOL	The instruction produced a valid result.
Dest	REAL	Result of the math instruction. Arithmetic status flags are set for this output.

Description: The MUL instruction multiplies Source A with Source B and places the result in the Destination.

Arithmetic Status Flags: Arithmetic status flags are affected.

Fault Conditions: none

Execution:



Relay Ladder

Condition:	Action:
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	Destination = Source B x Source A The rung-condition-out is set to true.
postscan	The rung-condition-out is set to false.

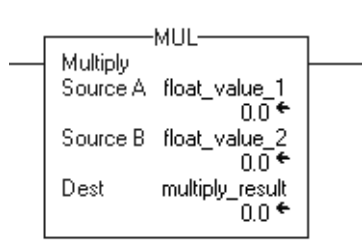


Function Block

Condition:	Action:
prescan	No action taken.
instruction first scan	No action taken.
instruction first run	No action taken.
EnableIn is cleared	EnableOut is cleared.
EnableIn is set	The instruction executes. EnableOut is set.
postscan	No action taken.

Example: Multiply *float_value_1* by *float_value_2* and place the result in *multiply_result*.

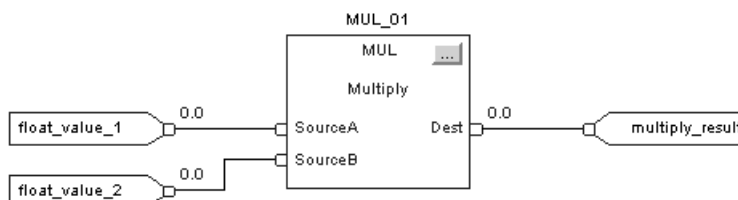
Relay Ladder



Structured Text

```
multiply_result := float_value_1 * float_value_2;
```

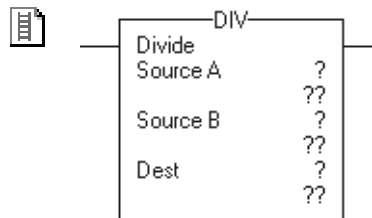
Function Block



Divide (DIV)

The DIV instruction divides Source A by Source B and places the result in the Destination.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Source A	SINT INT DINT REAL	immediate tag	value of the dividend
A SINT or INT tag converts to a DINT value by sign-extension.			
Source B	SINT INT DINT REAL	immediate tag	value of the divisor
A SINT or INT tag converts to a DINT value by sign-extension.			
Destination	SINT INT DINT REAL	tag	tag to store the result

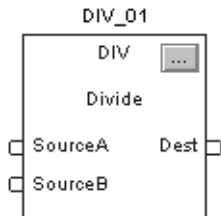


```
dest := sourceA / sourceB;
```

Structured Text

Use the divide sign “/” as an operator in an expression. This expression divides *sourceA* by *sourceB* and stores the result in *dest*.

See Appendix C for information on the syntax of expressions within structured text.



Function Block

Operand:	Type:	Format:	Description:
DIV tag	FBD_MATH	structure	DIV structure

FBD_MATH Structure

Input Parameter:	Data Type:	Description:
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set.
Source A	REAL	Value of the dividend. Valid = any float
Source B	REAL	Value of the divisor. Valid = any float
Output Parameter:	Data Type:	Description:
EnableOut	BOOL	The instruction produced a valid result.
Dest	REAL	Result of the math instruction. Arithmetic status flags are set for this output.

Description: If the Destination is *not* a REAL, the instruction handles the fractional portion of the result as follows:

If Source A:	Then the fractional portion of the result:	Example:
and Source B are <i>not</i> REALs	truncates	Source A DINT 5
		Source B DINT 3
		Destination DINT 1
or Source B is a REAL	rounds	Source A REAL 5.0
		Source B DINT 3
		Destination DINT 2

If Source B (the divisor) is zero:

- a minor fault occurs:
 - Type 4: program fault
 - Code 4: arithmetic overflow
- the destination is set as follows:

If Source B is zero and:	And the destination is a:	And the result is:	Then the destination is set to:
all operands are integers (SINT, INT, or DINT)	→	→	Source A
at least one operand is a REAL	SINT, INT, or DINT	positive	-1
		negative	0
	REAL	positive	1.\$ (positive infinity)
		negative	-1.\$ (negative infinity)

To detect a possible divide-by-zero, examine the minor fault bit (S:MINOR). See *Logix5000 Controllers Common Procedures*, publication 1756-PM001.

Arithmetic Status Flags: Arithmetic status flags are affected.

Fault Conditions:

A minor fault occurs if:	Fault type:	Fault code:
the divisor is zero	4	4

Execution:

Relay Ladder

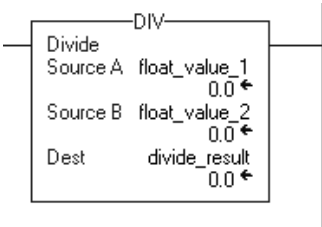
Condition:	Action:
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	Destination = Source A / Source B The rung-condition-out is set to true.
postscan	The rung-condition-out is set to false.

Function Block

Condition:	Action:
prescan	No action taken.
instruction first scan	No action taken.
instruction first run	No action taken.
EnableIn is cleared	EnableOut is cleared.
EnableIn is set	The instruction executes. EnableOut is set.
postscan	No action taken.

Example 1: Divide *float_value_1* by *float_value_2* and place the result in *divide_result*.

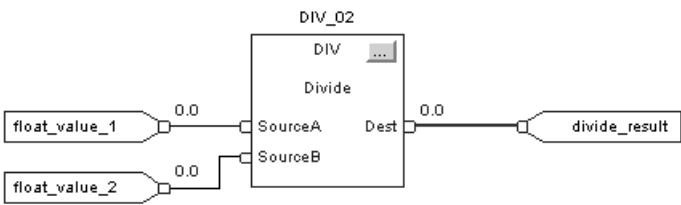
Relay Ladder



Structured Text

```
divide_result := float_value_1 / float_value_2;
```

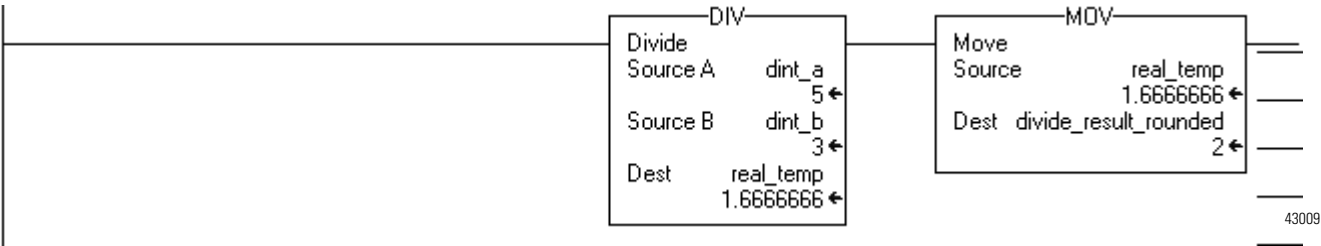
Function Block



Example 2: The DIV and MOV instructions work together to divide two integers, round the result, and place the result in an integer tag:

- The DIV instruction divides *dint_a* by *dint_b*.
- To round the result, the Destination is a REAL tag. (If the destination was an integer tag (SINT, INT, or DINT), the instruction would truncate the result.)
- The MOV instruction moves the rounded result (*real_temp*) from the DIV to *divide_result_rounded*.
- Since *divide_result_rounded* is a DINT tag the value from *real_temp* is rounded and placed in the DINT destination.

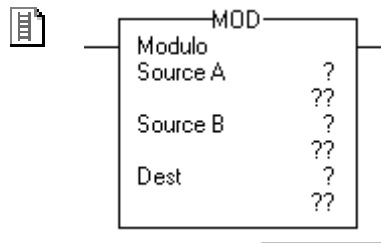
Relay Ladder



Modulo (MOD)

The MOD instruction divides Source A by Source B and places the remainder in the Destination.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Source A	SINT INT DINT REAL	immediate tag	value of the dividend
A SINT or INT tag converts to a DINT value by sign-extension.			
Source B	SINT INT DINT REAL	immediate tag	value of the divisor
A SINT or INT tag converts to a DINT value by sign-extension.			
Destination	SINT INT DINT REAL	tag	tag to store the result

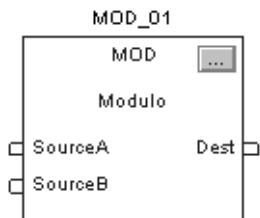


```
dest := sourceA MOD sourceB;
```

Structured Text

Use MOD as an operator in an expression. This expression divides *sourceA* by *sourceB* and stores the remainder in *dest*.

See Appendix C for information on the syntax of expressions within structured text.



Function Block

Operand:	Type:	Format:	Description:
MOD tag	FBD_MATH	structure	MOD structure

FBD_MATH Structure

Input Parameter:	Data Type:	Description:
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set.
Source A	REAL	Value of the dividend. Valid = any float
Source B	REAL	Value of the divisor. Valid = any float
Output Parameter:	Data Type:	Description:
EnableOut	BOOL	The instruction produced a valid result.
Dest	REAL	Result of the math instruction. Arithmetic status flags are set for this output.

Description: If Source B (the divisor) is zero:

- a minor fault occurs:
 - Type 4: program fault
 - Code 4: arithmetic overflow
- the destination is set as follows:

If Source B is zero and:	And the destination is a:	And the result is:	Then the destination is set to:
all operands are integers (SINT, INT, or DINT)	—————→	—————→	Source A
at least one operand is a REAL	SINT, INT, or DINT	positive	-1
		negative	0
	REAL	positive	1.\$ (positive infinity)
		negative	-1.\$ (negative infinity)

To detect a possible divide-by-zero, examine the minor fault bit (S:MINOR). See *Logix5000 Controllers Common Procedures*, publication 1756-PM001.

Arithmetic Status Flags: Arithmetic status flags are affected.

Fault Conditions:

A minor fault occurs if:	Fault type:	Fault code:
the divisor is zero	4	4

Execution:



Relay Ladder

Condition:	Action:
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	Destination = Source A – (TRN (Source A / Source B) * Source B) The rung-condition-out is set to true.
postscan	The rung-condition-out is set to false.

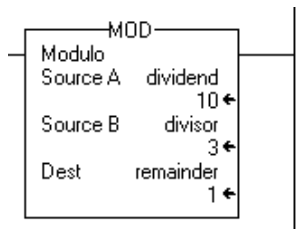


Function Block

Condition:	Action:
prescan	No action taken.
instruction first scan	No action taken.
instruction first run	No action taken.
EnableIn is cleared	EnableOut is cleared.
postscan	No action taken.

Example: Divide *dividend* by *divisor* and place the remainder in *remainder*. In this example, three goes into 10 three times, with a remainder of one.

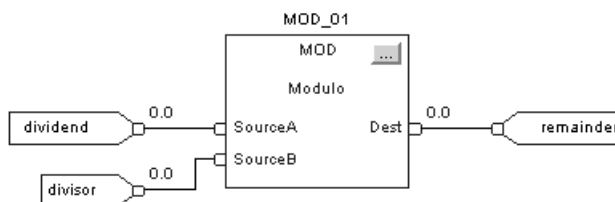
Relay Ladder



Structured Text

```
remainder := dividend MOD divisor;
```

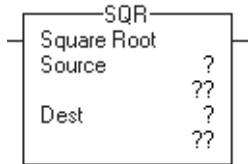
Function Block



Square Root (SQR)

The SQR instruction computes the square root of the Source and places the result in the Destination.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Source	SINT INT DINT REAL	immediate tag	find the square root of this value
A SINT or INT tag converts to a DINT value by sign-extension.			
Destination	SINT INT DINT REAL	tag	tag to store the result

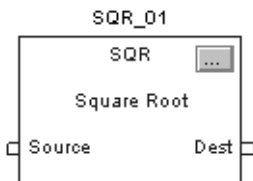


```
dest := SQRT(source);
```

Structured Text

Use SQRT as a function. This expression computes the square root of *source* and stores the result in *dest*.

See Appendix C for information on the syntax of expressions within structured text.



Function Block

Operand:	Type:	Format:	Description:
SQR tag	FBD_MATH_ADVANCED	structure	SQR structure

FBD_MATH_ADVANCED Structure

Input Parameter:	Data Type:	Description:
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set.
Source	REAL	Find the square root of this value. Valid = any float
Output Parameter:	Data Type:	Description:
EnableOut	BOOL	The instruction produced a valid result.
Dest	REAL	Result of the math instruction. Arithmetic status flags are set for this output.

Description: If the Destination is *not* a REAL, the instruction handles the fractional portion of the result as follows:

If the Source is:	Then the fractional portion of the result:	Example:		
<i>not</i> a REAL	truncates	Source	DINT	3
		Destination	DINT	1
a REAL	rounds	Source	REAL	3.0
		Destination	DINT	2

If the Source is negative, the instruction takes the absolute value of the Source before calculating the square root.

Arithmetic Status Flags: Arithmetic status flags are affected.

Fault Conditions: none

Execution:



Relay Ladder

Condition:	Action:
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	$Destination = \sqrt{Source}$ The rung-condition-out is set to true.
postscan	The rung-condition-out is set to false.

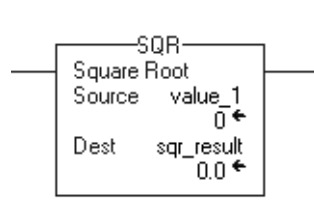


Function Block

Condition:	Action:
prescan	No action taken.
instruction first scan	No action taken.
instruction first run	No action taken.
EnableIn is cleared	EnableOut is cleared.
EnableIn is set	The instruction executes. EnableOut is set.
postscan	No action taken.

Example: Calculate the square root of *value_1* and place the result in *sqr_result*.

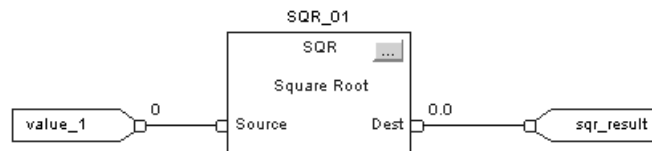
Relay Ladder



Structured Text

```
sqr_result := SQR(value_1);
```

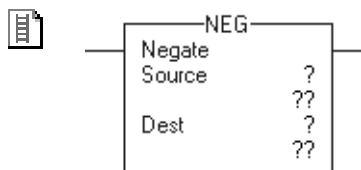
Function Block



Negate (NEG)

The NEG instruction changes the sign of the Source and places the result in the Destination.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Source	SINT INT DINT REAL	immediate tag	value to negate
A SINT or INT tag converts to a DINT value by sign-extension.			
Destination	SINT INT DINT REAL	tag	tag to store the result

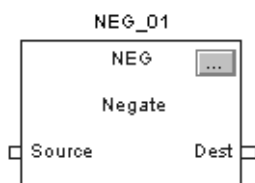


```
dest := -source;
```

Structured Text

Use the minus sign “-” as an operator in an expression. This expression changes the sign of *source* and stores the result in *dest*.

See Appendix C for information on the syntax of expressions within structured text.



Function Block

Operand:	Type:	Format:	Description:
NEG tag	FBD_MATH_ADVANCED	structure	NEG structure

FBD_MATH Structure

Input Parameter:	Data Type:	Description:
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. default is set
Source	REAL	Value to negate. valid = any float
Output Parameter:	Data Type:	Description:
EnableOut	BOOL	The instruction produced a valid result.
Dest	REAL	Result of the math instruction. Arithmetic status flags are set for this output.

Description: If you negate a negative value, the result is positive. If you negate a positive value, the result is negative.

Arithmetic Status Flags: Arithmetic status flags are affected.

Fault Conditions: none

Execution:



Relay Ladder

Condition:	Action:
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	Destination = 0 – Source The rung-condition-out is set to true.
postscan	The rung-condition-out is set to false.

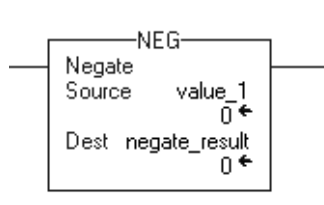


Function Block

Condition:	Action:
prescan	No action taken.
instruction first scan	No action taken.
instruction first run	No action taken.
EnableIn is cleared	EnableOut is cleared.
EnableIn is set	The instruction executes. EnableOut is set.
postscan	No action taken.

Example: Change the sign of *value_1* and place the result in *negate_result*.

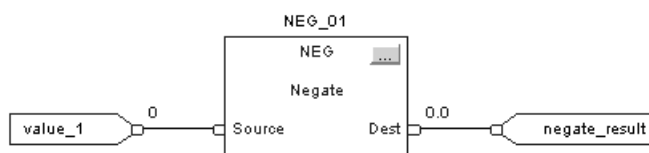
Relay Ladder



Structured Text

```
negate_result := -value_1;
```

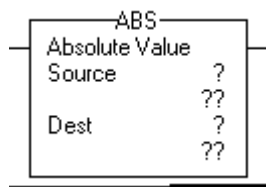
Function Block



Absolute Value (ABS)

The ABS instruction takes the absolute value of the Source and places the result in the Destination.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Source	SINT INT DINT REAL	immediate tag	value of which to take the absolute value
A SINT or INT tag converts to a DINT value by sign-extension.			
Destination	SINT INT DINT REAL	tag	tag to store the result

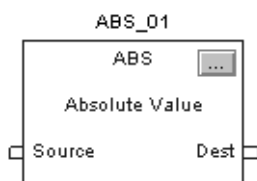


```
dest := ABS(source);
```

Structured Text

Use ABS as a function. This expression computes the absolute value of *source* and stores the result in *dest*.

See Appendix C for information on the syntax of expressions within structured text.



Function Block

Operand:	Type:	Format:	Description:
ABS tag	FBD_MATH_ADVANCED	structure	ABS structure

FBD_MATH_ADVANCED Structure

Input Parameter:	Data Type:	Description:
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set.
Source	REAL	Value of which to take the absolute value. Valid = any float
Output Parameter:	Data Type:	Description:
EnableOut	BOOL	The instruction produced a valid result.
Dest	REAL	Result of the math instruction. Arithmetic status flags are set for this output.

Description: The ABS instruction takes the absolute value of the Source and places the result in the Destination.

Arithmetic Status Flags: Arithmetic status flags are affected.

Fault Conditions: none

Execution:



Relay Ladder

Condition:	Action:
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	Destination = Source The rung-condition-out is set to true.
postscan	The rung-condition-out is set to false.

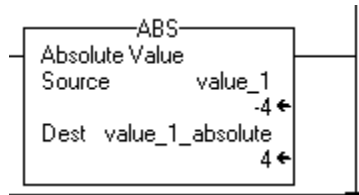


Function Block

Condition:	Action:
prescan	No action taken.
instruction first scan	No action taken.
instruction first run	No action taken.
EnableIn is cleared	EnableOut is cleared.
EnableIn is set	The instruction executes. EnableOut is set.
postscan	No action taken.

Example: Place the absolute value of *value_1* into *value_1_absolute*. In this example, the absolute value of negative four is positive four.

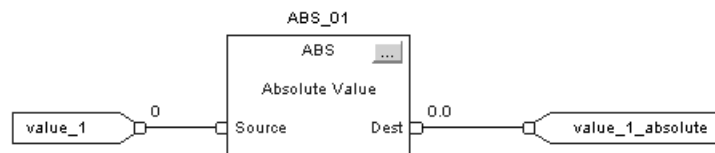
Relay Ladder



Structured Text

```
value_1_absolute := ABS(value_1);
```

Function Block



Notes:

Move/Logical Instructions

(MOV, MVM, BT, MVMT, BTD, CLR, SWPB, AND, OR, XOR, NOT, BAND, BOR, BXOR, BNOT)

Introduction

You can mix data types, but loss of accuracy and rounding error might occur and the instruction takes more time to execute. Check the S:V bit to see whether the result was truncated.

For relay ladder instructions, **bold** data types indicate optimal data types. An instruction executes faster and requires less memory if all the operands of the instruction use the same optimal data type, typically DINT or REAL.

The move instructions modify and move bits.

If you want to:	Use this instruction:	Available in these languages:	See page:
copy a value	MOV	relay ladder structured text ⁽¹⁾	6-3
copy a specific part of an integer	MVM	relay ladder	6-5
copy a specific part of an integer in function block	MVMT	structured text function block	6-8
move bits within an integer or between integers	BT	relay ladder	6-11
move bits within an integer or between integers in function block	BTD	structured text function block	6-14
clear a value	CLR	structured text ⁽¹⁾ relay ladder	6-17
rearrange the bytes of a INT, DINT, or REAL tag	SWPB	relay ladder structured text	6-19

⁽¹⁾ There is no equivalent structured text instruction. Use other structured text programming to achieve the same result. See the description for the instruction.

The logical instructions perform operations on bits.

If you want to:	Use this instruction:	Available in these languages:	See page:
bitwise AND operation	Bitwise AND & ⁽¹⁾	relay ladder structured text ⁽²⁾ function block	6-23
bitwise OR operation	Bitwise OR	relay ladder structured text ⁽²⁾ function block	6-26
bitwise, exclusive OR operation	Bitwise XOR	relay ladder structured text ⁽²⁾ function block	6-29
bitwise NOT operation	Bitwise NOT	relay ladder structured text ⁽²⁾ function block	6-32
logically AND as many as eight boolean inputs.	Boolean AND (BAND)	structured text ⁽²⁾ function block	6-35
logically OR as many as eight boolean inputs.	Boolean OR (BOR)	structured text ⁽²⁾ function block	6-38
perform an exclusive OR on two boolean inputs.	Boolean Exclusive OR (BXOR)	structured text ⁽²⁾ function block	6-41
complement a boolean input.	Boolean NOT (BNOT)	structured text ⁽²⁾ function block	6-44

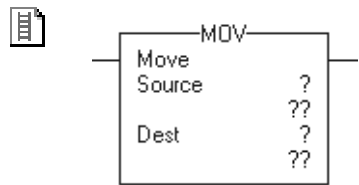
⁽¹⁾ Structured text only.

⁽²⁾ In structured text, the AND, OR, XOR, and NOT operations can be bitwise or logical.

Move (MOV)

The MOV instruction copies the Source to the Destination. The Source remains unchanged.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Source	SINT INT DINT REAL	immediate tag	value to move (copy)
A SINT or INT tag converts to a DINT value by sign-extension.			
Destination	SINT INT DINT REAL	tag	tag to store the result



dest := source;

Structured Text

Use an assignment “:=” with an expression. This assignment moves the value in *source* to *dest*.

See Structured Text Programming for information on the syntax of expressions and assignments within structured text.

Description: The MOV instruction copies the Source to the Destination. The Source remains unchanged.

Arithmetic Status Flags: Arithmetic status flags are affected.

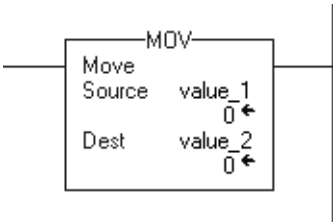
Fault Conditions: none

Execution:

Condition:	Relay Ladder Action:
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	The instruction copies the Source into the Destination. The rung-condition-out is set to true.
postscan	The rung-condition-out is set to false.

Example: Move the data in *value_1* to *value_2*.

Relay Ladder



Structured Text

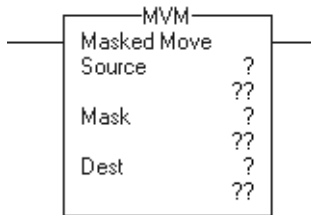
```
value_2 := value _1;
```

Masked Move (MVM)

The MVM instruction copies the Source to a Destination and allows portions of the data to be masked.

This instruction is available in structured text and function block as MVMT, see page 6-8.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Source	SINT INT DINT	immediate tag	value to move
A SINT or INT tag converts to a DINT value by zero-fill.			
Mask	SINT INT DINT	immediate tag	which bits to block or pass
A SINT or INT tag converts to a DINT value by zero-fill.			
Destination	SINT INT DINT	tag	tag to store the result



```
dest := (Dest AND NOT (Mask))
      OR (Source AND Mask);
```

Structured Text

This instruction is available in structured text as MVMT. Or you can combine bitwise logic within an expression and assign the result to the destination. This expression performs a masked move on *Source*.

See Structured Text Programming for information on the syntax of expressions and assignments within structured text.

Description: The MVM instruction uses a Mask to either pass or block Source data bits. A “1” in the mask means the data bit is passed. A “0” in the mask means the data bit is blocked.

If you mix integer data types, the instruction fills the upper bits of the smaller integer data types with 0s so that they are the same size as the largest data type.

Entering an immediate mask value

When you enter a mask, the programming software defaults to decimal values. If you want to enter a mask using another format, precede the value with the correct prefix.

Prefix:	Description:
16#	hexadecimal for example; 16#0F0F
8#	octal for example; 8#16
2#	binary for example; 2#00110011

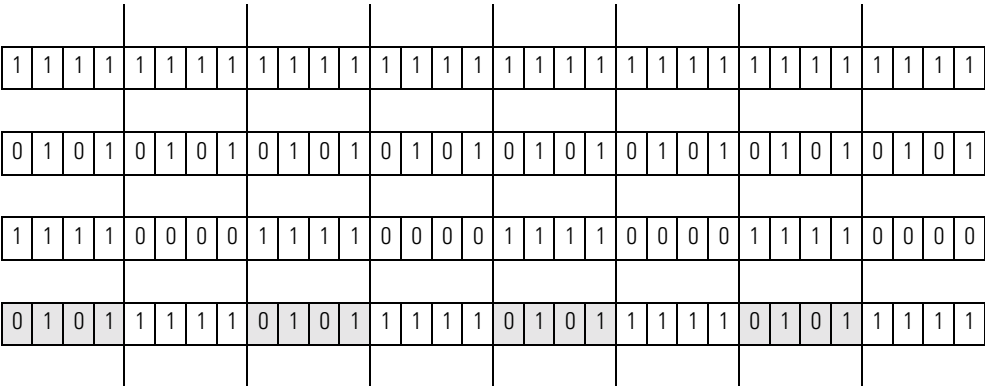
Arithmetic Status Flags: Arithmetic status flags are affected.

Fault Conditions: none

Execution:

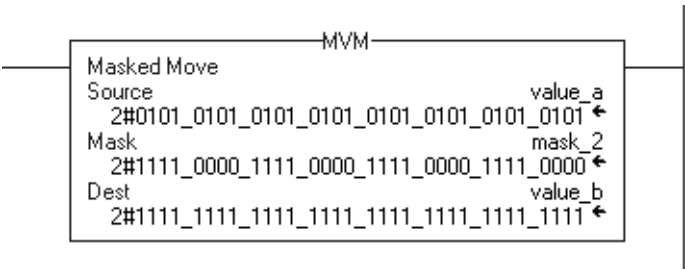
Condition:	Relay Ladder Action:
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	The instruction passes the Source through the Mask and copies the result into the Destination. Unmasked bits in the Destination remain unchanged. The rung-condition-out is set to true.
postscan	The rung-condition-out is set to false.

Example: Copy data from *value_a* to *value_b*, while allowing data to be masked (a 0 masks the data in *value_a*).



The shaded boxes show the bits that changed in *value_b*.

Relay Ladder



Structured Text

```
value_b := (value_b AND NOT (mask_2)) OR
           (value_a AND mask_2);
```

Masked Move with Target (MVMT)

The MVMT instruction first copies the Target to the Destination. Then the instruction compares the masked Source to the Destination and makes any required changes to the Destination. The Target and the Source remain unchanged.

This instruction is available in relay ladder as MVM, see page 6-5.

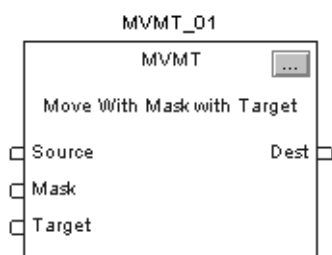
Operands:



MVMT (MVMT_tag) ;

Structured Text

Variable:	Type:	Format:	Description:
MVMT tag	FBD_MASKED_MOVE	structure	MVMT structure



Function Block

Operand:	Type:	Format:	Description:
MVMT tag	FBD_MASKED_MOVE	structure	MVMT structure

FBD_MASKED_MOVE Structure

Input Parameter:	Data Type:	Description:
EnableIn	BOOL	Function Block: If cleared, the instruction does not execute and outputs are not updated. If set, the instruction executes. Default is set. Structured Text: No effect. The instruction executes.
Source	DINT	Input value to move to Destination based on value of Mask. Valid = any integer
Mask	DINT	Mask of bits to move from Source to Dest. All bits set to one cause the corresponding bits to move from Source to Dest. All bits that are set to zero cause the corresponding bits not to move from Source to Dest. Valid = any integer
Target	DINT	Input value to move to Dest prior to moving Source bits through the Mask. Valid = any integer
Output Parameter:	Data Type:	Description:
EnableOut	BOOL	The instruction produced a valid result.
Dest	DINT	Result of masked move instruction. Arithmetic status flags are set for this output.

Description: When enabled, the MVMT instruction uses a Mask to either pass or block Source data bits. A “1” in the mask means the data bit is passed. A “0” in the mask means the data bit is blocked.

If you mix integer data types, the instruction fills the upper bits of the smaller integer data types with 0s so that they are the same size as the largest data type.

Entering an immediate mask value using an Input Reference

When you enter a mask, the programming software defaults to decimal values. If you want to enter a mask using another format, precede the value with the correct prefix.

Prefix:	Description:
16#	hexadecimal for example; 16#0F0F
8#	octal for example; 8#16
2#	binary for example; 2#00110011

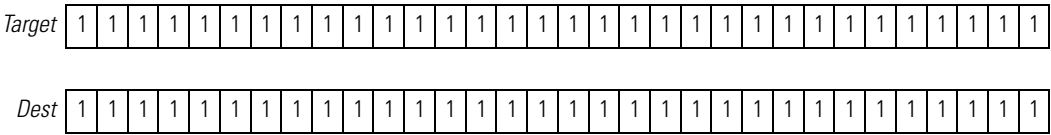
Arithmetic Status Flags: Arithmetic status flags are affected.

Fault Conditions: none

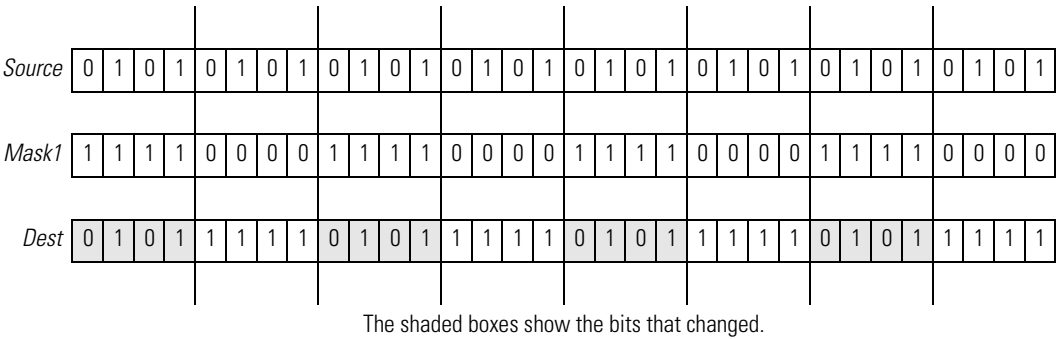
Execution:

Condition:	Function Block Action:	Structured Text Action:
prescan	No action taken.	No action taken.
instruction first scan	No action taken.	No action taken.
instruction first run	No action taken.	No action taken.
EnableIn is cleared	EnableOut is cleared, the instruction does nothing, and the outputs are not updated.	na
EnableIn is set	The instruction executes. EnableOut is set.	EnableIn is always set. The instruction executes.
postscan	No action taken.	No action taken.

Example: 1. Copy Target into Dest.



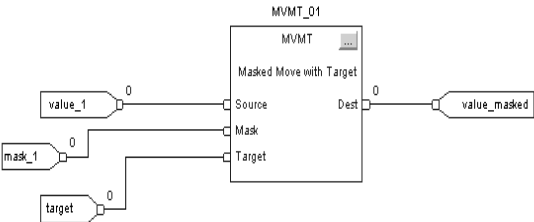
2. Mask Source and compare it to Dest. Any required changes are made in Dest. Source and Target remain unchanged. A 0 in the mask restrains the instruction from comparing that bit (shown by x in the example).



Structured Text

```
MVMT_01.Source := value_1;  
MVMT_01.Mask := mask1;  
MVMT_01.Target := target;  
  
MVMT(MVMT_01);  
  
value_masked := MVMT_01.Dest;
```

Function Block

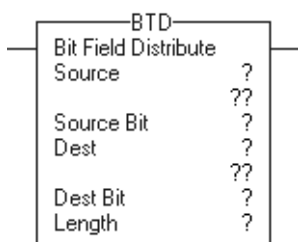


Bit Field Distribute (BTD)

The BTD instruction copies the specified bits from the Source, shifts the bits to the appropriate position, and writes the bits into the Destination.

This instruction is available in structured text and function block as BTDT, see page 6-14.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Source	SINT INT DINT	immediate tag	tag that contains the bits to move
A SINT or INT tag converts to a DINT value by zero-fill.			
Source bit	DINT	immediate (0-31 DINT) (0-15 INT) (0-7 SINT)	number of the bit (lowest bit number) from where to start the move must be within the valid range for the Source data type
Destination	SINT INT DINT	tag	tag where to move the bits
Destination bit	DINT	immediate (0-31 DINT) (0-15 INT) (0-7 SINT)	the number of the bit (lowest bit number) where to start copying bits from the Source must be within the valid range for the Destination data type
Length	DINT	immediate (1-32)	number of bits to move

Description: When enabled, the BTD instruction copies a group of bits from the Source to the Destination. The group of bits is identified by the Source bit (lowest bit number of the group) and the Length (number of bits to copy). The Destination bit identifies the lowest bit number bit to start with in the Destination. The Source remains unchanged.

If the length of the bit field extends beyond the Destination, the instruction does not save the extra bits. Any extra bits do not wrap to the next word.

If you mix integer data types, the instruction fills the upper bits of the smaller integer data types with 0s so that they are the same size as the largest data type.

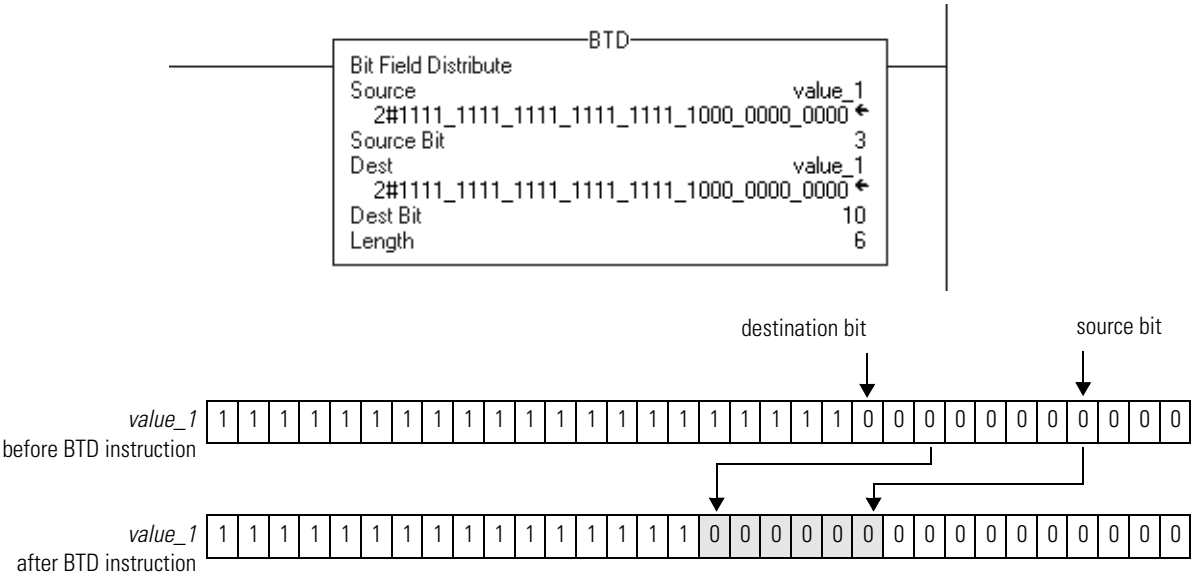
Arithmetic Status Flags: Arithmetic status flags are affected.

Fault Conditions: none

Execution:

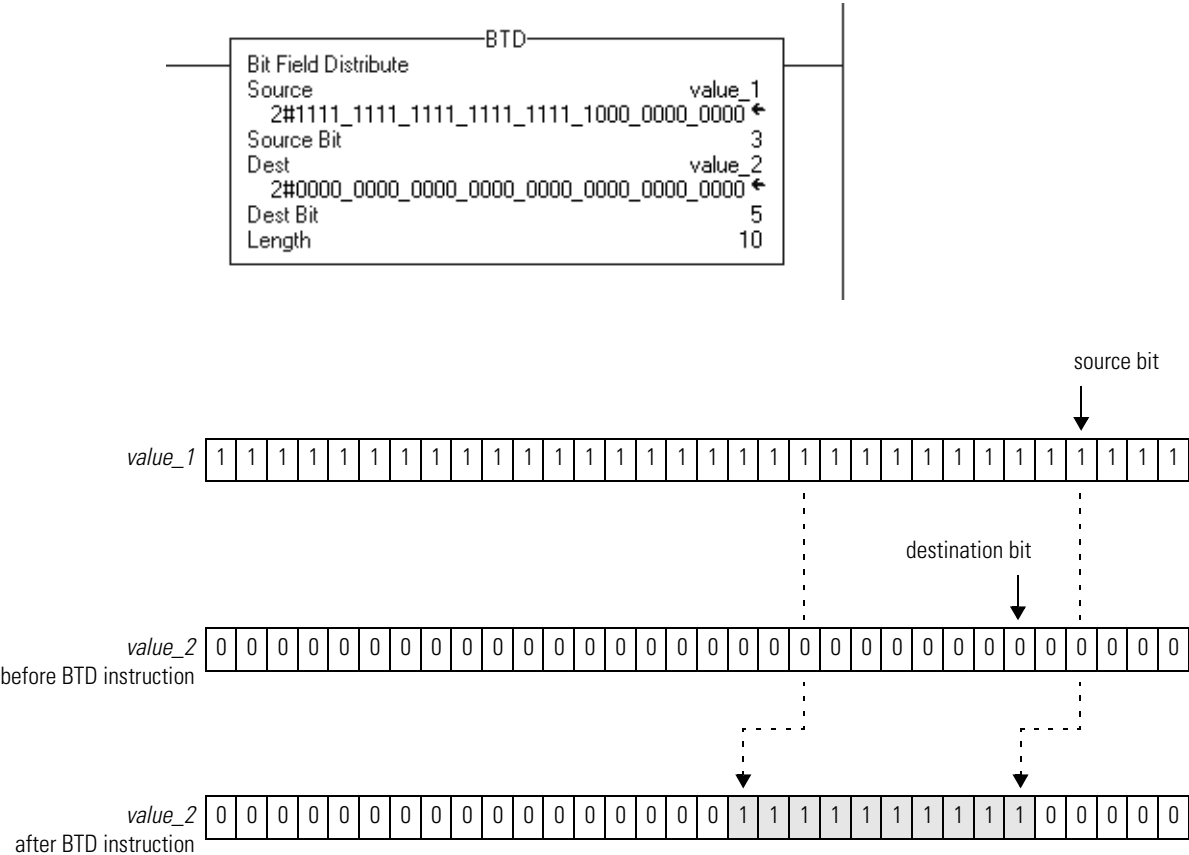
Condition:	Relay Ladder Action:
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	The instruction copies and shifts the Source bits to the Destination. The rung-condition-out is set to true.
postscan	The rung-condition-out is set to false.

Example 1: When enabled, the BTS instruction moves bits within *value_1*.



The shaded boxes show the bits that changed in *value_1*.

Example 2: When enabled, the BTD instruction moves 10 bits from *value_1* to *value_2*.



The shaded boxes show the bits that changed in *value_2*.

Bit Field Distribute with Target (BTDT)

The BTDT instruction first copies the Target to the Destination. Then the instruction copies the specified bits from the Source, shifts the bits to the appropriate position, and writes the bits into the Destination. The Target and Source remain unchanged.

This instruction is available in relay ladder as BTD, see page 6-11.

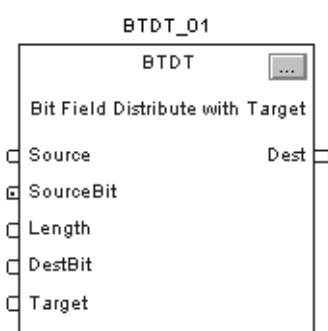
Operands:



BTDT (BTDT_tag) ;

Structured Text

Variable:	Type:	Format:	Description:
BTDT tag	FBD_BIT_FIELD_DISTRIBUTE	structure	BTDT structure



Function Block

Operand:	Type:	Format:	Description:
BTDT tag	FBD_BIT_FIELD_DISTRIBUTE	structure	BTDT structure

FBD_BIT_FIELD_DISTRIBUTE Structure

Input Parameter:	Data Type:	Description:
EnableIn	BOOL	Function Block: If cleared, the instruction does not execute and outputs are not updated. If set, the instruction executes. Default is set. Structured Text: No effect. The instruction executes.
Source	DINT	Input value containing the bits to move to Destination. Valid = any integer
SourceBit	DINT	The bit position in Source (lowest bit number from where to start the move). Valid = 0-31
Length	DINT	Number of bits to move Valid = 1-32
DestBit	DINT	The bit position in Dest (lowest bit number to start copying bits into). Valid = 0-31
Target	DINT	Input value to move to Dest prior to moving bits from the Source. Valid = any integer
Output Parameter:	Data Type:	Description:
EnableOut	BOOL	The instruction produced a valid result.
Dest	DINT	Result of the bit move operation. Arithmetic status flags are set for this output.

Description: When enabled, the BTD instruction copies a group of bits from the Source to the Destination. The group of bits is identified by the Source bit (lowest bit number of the group) and the Length (number of bits to copy). The Destination bit identifies the lowest bit number bit to start with in the Destination. The Source remains unchanged.

If the length of the bit field extends beyond the Destination, the instruction does not save the extra bits. Any extra bits do not wrap to the next word.

Arithmetic Status Flags: Arithmetic status flags are affected

Fault Conditions: none

Execution:

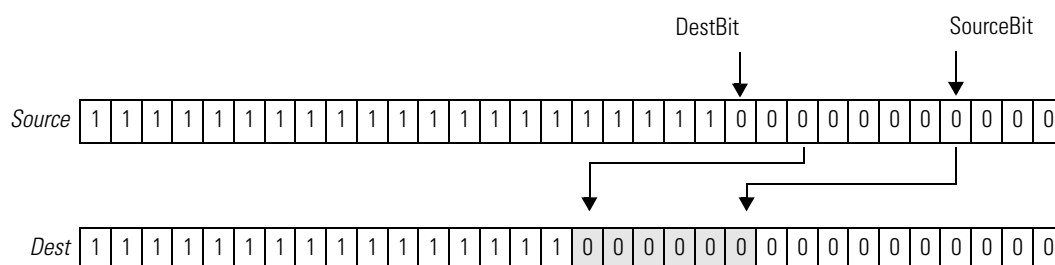
Condition:	Function Block Action:	Structured Text Action:
prescan	No action taken.	No action taken.
instruction first scan	No action taken.	No action taken.
instruction first run	No action taken.	No action taken.
EnableIn is cleared	EnableOut is cleared, the instruction does nothing, and the outputs are not updated.	na
EnableIn is set	The instruction executes. EnableOut is set.	EnableIn is always set. The instruction executes.
postscan	No action taken.	No action taken.

Example: 1. The controller copies Target into Dest.

Target 1 0 0 0 0 0 0 0 0 0 0 0 0

Dest 1 0 0 0 0 0 0 0 0 0 0 0 0

2. The SourceBit and the Length specify which bits in Source to copy into Dest, starting at DestBit. Source and Target remain unchanged.



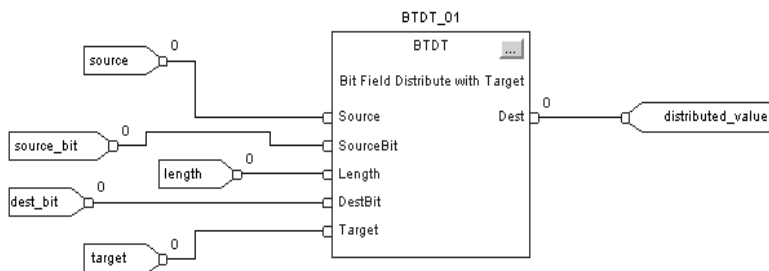
Structured Text

```
BTDT_01.Source := source;
BTDT_01.SourceBit := source_bit;
BTDT_01.Length := length;
BTDT_01.DestBit := dest_bit;
BTDT_01.Target := target;
```

```
BTDT(BTDT_01);
```

```
distributed_value := BTDT_01.Dest;
```

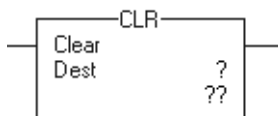
Function Block



Clear (CLR)

The CLR instruction clears all the bits of the Destination.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Destination	SINT INT DINT REAL	tag	tag to clear



```
dest := 0;
```

Structured Text

Structured text does not have a CLR instruction. Instead, assign 0 to the tag you want to clear. This assignment statement clears *dest*.

See Structured Text Programming for information on the syntax of expressions and assignment statements within structured text.

Description: The CLR instruction clears all the bits of the Destination.

Arithmetic Status Flags: Arithmetic status flags are affected.

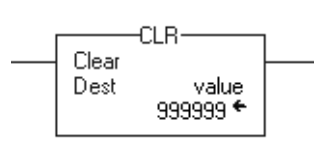
Fault Conditions: none

Execution:

Condition:	Relay Ladder Action:
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	The instruction clears the Destination. The rung-condition-out is set to true.
postscan	The rung-condition-out is set to false.

Example: Clear all the bits of *value* to 0.

Relay Ladder



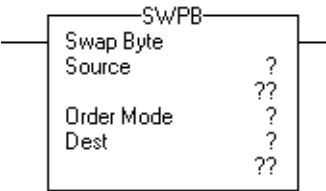
Structured Text

```
value := 0;
```

Swap Byte (SWPB)

The SWPB instruction rearranges the bytes of a value.

Operands:



Relay Ladder

Operand:	Type:	Format:	Enter:		
Source	INT DINT REAL	tag	tag that contains the bytes that you want to rearrange		
Order Mode			If the Source is an:	And you want to change the bytes to this pattern (each letter represents a different byte):	Then select:
			INT	n/a	any of the options
			DINT REAL	ABCD ⇒ DCBA	REVERSE (or enter 0)
				ABCD ⇒ CDAB	WORD (or enter 1)
				ABCD ⇒ BADC	HIGH/LOW (or enter 2)
Destination	INT DINT REAL	tag	tag to store the bytes in the new order		
			If the Source is an:	Then the Destination must be an:	
			INT	INT	
				DINT	
			DINT	DINT	
REAL	REAL				



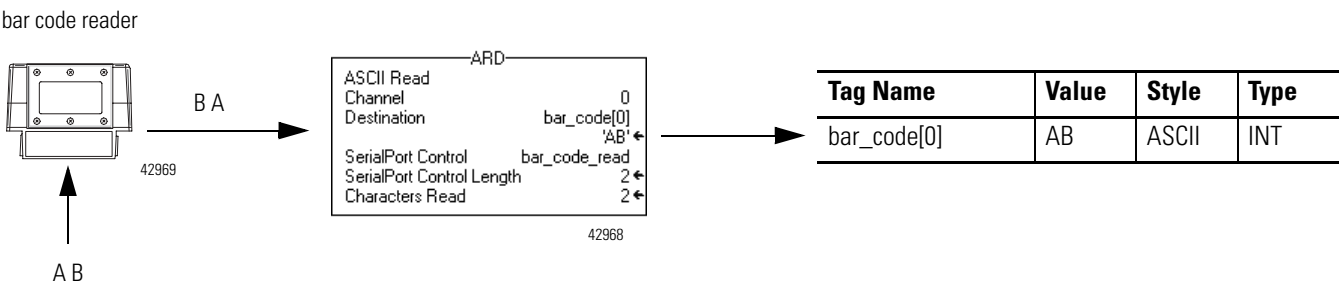
SWPB (Source,OrderMode,Dest) ;

Structured Text

The operands are the same as those for the relay ladder SWPB instruction. If you select the HIGH/LOW order mode, enter it as HIGHLOW or HIGH_LOW (without the slash).

Description: The SWPB instruction rearranges the order of the bytes of the Source. It places the result in the Destination.

When you read or write ASCII characters, you typically *do not* need to swap characters. The ASCII read and write instructions (ARD, ARL, AWA, AWT) automatically swap characters, as shown below.



Arithmetic Status Flags: not affected

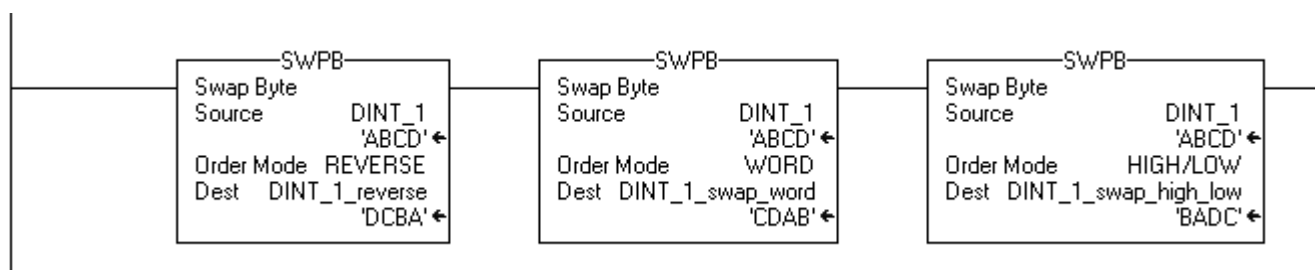
Fault Conditions: none

Execution:

Condition:	Relay Ladder Action:	Structured Text Action:
prescan	The rung-condition-out is set to false.	No action taken
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The rung-condition-out is set to true.	na
EnableIn is set	na	EnableIn is always set. The instruction executes.
instruction execution	The instruction rearranges the specified bytes.	The instruction rearranges the specified bytes.
postscan	The rung-condition-out is set to false.	No action taken.

Example 1: The three SWPB instructions each reorder the bytes of *DINT_1* according to a different order mode. The display style is ASCII, and each character represents one byte. Each instruction places the bytes, in the new order, in a different Destination.

Relay Ladder



Structured Text

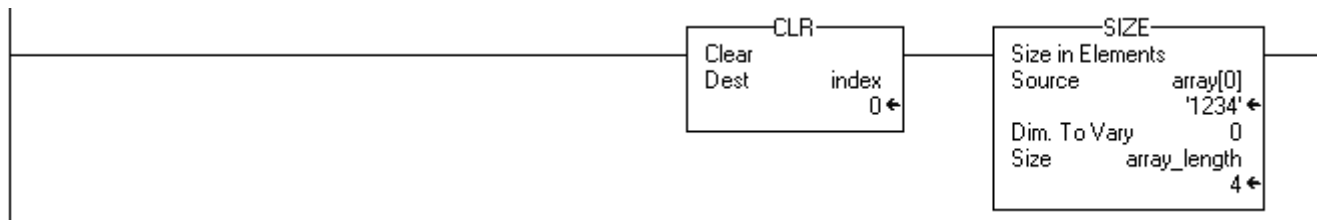
```
SWPB (DINT_1, REVERSE, DINT_1_reverse);
SWPB (DINT_1, WORD, DINT_1_swap_word);
SWPB (DINT_1, HIGHLOW, DINT_1_swap_high_low);
```

Example 2: The following example reverses the bytes in each element of an array. For an RSLogix 5000 project that contains this example, open the RSLogix 5000\Projects\Samples folder, Swap_Bytes_in_Array.ACD file.

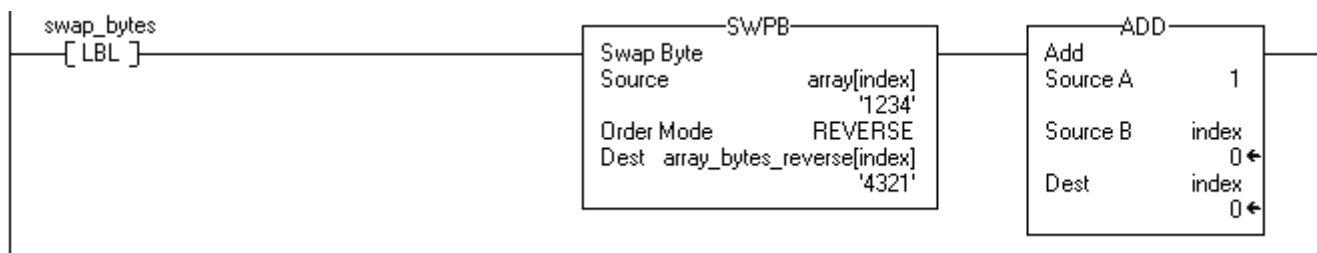
1. Initialize the tags. The SIZE instruction finds the number of elements in *array* and stores that value in *array_length*. A subsequent instruction uses this value to determine when the routine has acted on all the elements in the array.
2. Reverse the bytes in one element of *array*.
 - The SWPB instruction reverses the bytes of the element number that is indicated by the value of *index*. For example, when *index* equals 0, the SWPB instruction acts on *array[0]*.
 - The ADD instruction increments *index*. The next time the instruction executes, the SWPB instruction acts on the next element in *array*.
3. Determine when the SWPB instruction has acted on all the elements in the array.
 - If *index* is less than the number of elements in the array (*array_length*), then continue with the next element in the array.
 - If *index* equals *array_length*, then the SWPB has acted on all the elements in the array.

Relay Ladder

Initialize the tags.



Reverse the bytes.



Determine whether the SWPB instruction has acted on all the elements in the array.



Structured Text

```

index := 0;
SIZE (array[0], 0, array_length);
REPEAT
    SWPB(array[index], REVERSE, array_bytes_reverse[index]);
    index := index + 1;
UNTIL(index >= array_length)END_REPEAT;

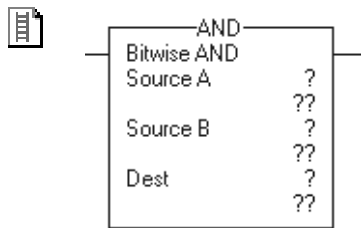
```

Bitwise AND (AND)

The AND instruction performs a bitwise AND operation using the bits in Source A and Source B and places the result in the Destination.

To perform a logical AND, see page 6-35.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Source A	SINT INT DINT	immediate tag	value to AND with Source B
A SINT or INT tag converts to a DINT value by zero-fill.			
Source B	SINT INT DINT	immediate tag	value to AND with Source A
A SINT or INT tag converts to a DINT value by zero-fill.			
Destination	SINT INT DINT	tag	stores the result



dest := sourceA AND sourceB

Structured Text

Use AND or the ampersand sign “&” as an operator within an expression. This expression evaluates *sourceA* AND *sourceB*.

See Structured Text Programming for information on the syntax of expressions within structured text.



Function Block

Operand:	Type:	Format:	Description:
AND tag	FBD_LOGICAL	structure	AND structure

FBD_LOGICAL Structure

Input Parameter:	Data Type:	Description:
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set.
SourceA	DINT	Value to AND with SourceB. Valid = any integer
SourceB	DINT	Value to AND with SourceA. Valid = any integer
Output Parameter:	Data Type:	Description:
EnableOut	BOOL	The instruction produced a valid result.
Dest	DINT	Result of the instruction. Arithmetic status flags are set for this output.

Description: When enabled, the instruction evaluates the AND operation:

If the bit in Source A is:	And the bit in Source B is:	The bit in the Destination is:
0	0	0
0	1	0
1	0	0
1	1	1

If you mix integer data types, the instruction fills the upper bits of the smaller integer data types with 0s so that they are the same size as the largest data type.

Arithmetic Status Flags: Arithmetic status flags are affected.

Fault Conditions: none

Execution:

**Relay Ladder**

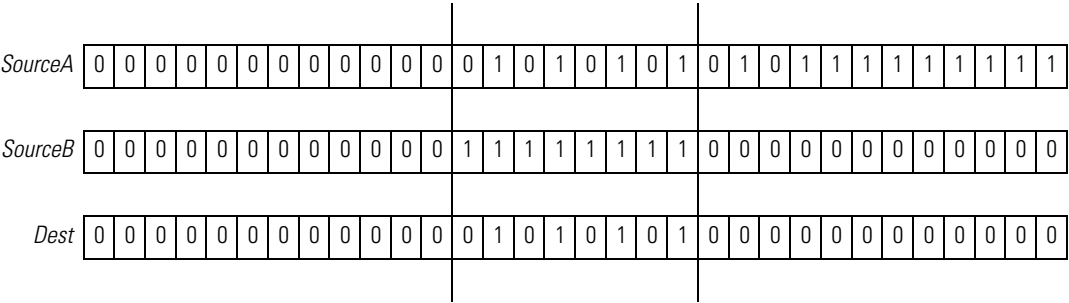
Condition:	Action:
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	The instruction performs a bitwise AND operation. The rung-condition-out is set to true.
postscan	The rung-condition-out is set to false.



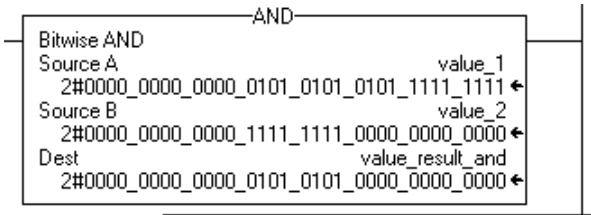
Function Block

Condition:	Action:
prescan	No action taken.
instruction first scan	No action taken.
instruction first run	No action taken.
EnableIn is cleared	EnableOut is cleared.
EnableIn is set	The instruction executes. EnableOut is set.
postscan	No action taken.

Example: When enabled, the AND instruction performs a bitwise AND operation on SourceA and SourceB and places the result in the Dest.



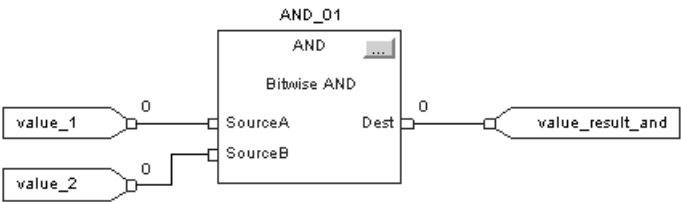
Relay Ladder



Structured Text

```
value_result_and := value_1 AND value_2;
```

Function Block

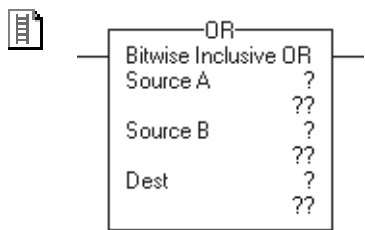


Bitwise OR (OR)

The OR instruction performs a bitwise OR operation using the bits in Source A and Source B and places the result in the Destination.

To perform a logical OR, see page 6-38.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Source A	SINT INT DINT	immediate tag	value to OR with Source B
A SINT or INT tag converts to a DINT value by zero-fill.			
Source B	SINT INT DINT	immediate tag	value to OR with Source A
A SINT or INT tag converts to a DINT value by zero-fill.			
Destination	SINT INT DINT	tag	stores the result



dest := sourceA OR sourceB

Structured Text

Use OR as an operator within an expression. This expression evaluates *sourceA* OR *sourceB*.

See Structured Text Programming for information on the syntax of expressions within structured text.



Function Block

Operand:	Type:	Format:	Description:
OR tag	FBD_LOGICAL	structure	OR structure

FBD_LOGICAL Structure

Input Parameter:	Data Type:	Description:
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set.
SourceA	DINT	Value to OR with SourceB. Valid = any integer
SourceB	DINT	Value to OR with SourceA. Valid = any integer
Output Parameter:	Data Type:	Description:
EnableOut	BOOL	The instruction produced a valid result.
Dest	DINT	Result of the instruction. Arithmetic status flags are set for this output.

Description: When enabled, the instruction evaluates the OR operation:

If the bit in Source A is:	And the bit in Source B is:	The bit in the Destination is:
0	0	0
0	1	1
1	0	1
1	1	1

If you mix integer data types, the instruction fills the upper bits of the smaller integer data types with 0s so that they are the same size as the largest data type.

Arithmetic Status Flags: Arithmetic status flags are affected.

Fault Conditions: none

Execution:

**Relay Ladder**

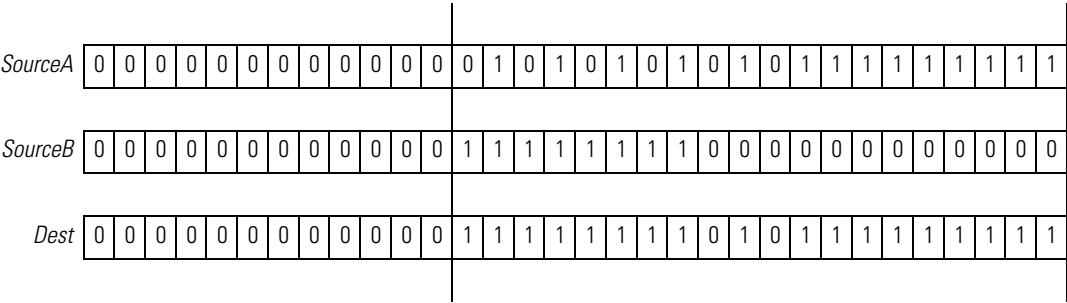
Condition:	Action:
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	The instruction performs a bitwise OR operation. The rung-condition-out is set to true.
postscan	The rung-condition-out is set to false.



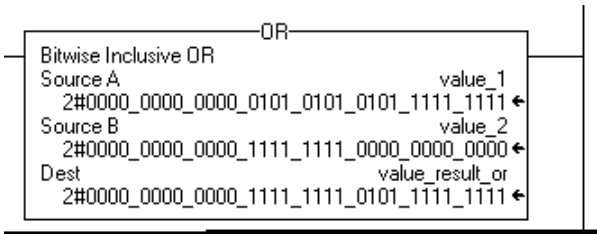
Function Block

Condition:	Action:
prescan	No action taken.
instruction first scan	No action taken.
instruction first run	No action taken.
EnableIn is cleared	EnableOut is cleared.
EnableIn is set	The instruction executes. EnableOut is set.
postscan	No action taken.

Example: When enabled, the OR instruction performs a bitwise OR operation on SourceA and SourceB and places the result in Dest.



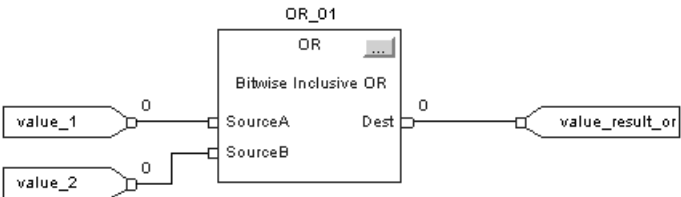
Relay Ladder



Structured Text

value_result_or := value_1 OR value_2;

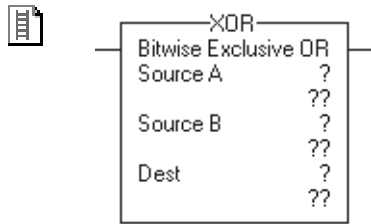
Function Block



Bitwise Exclusive OR (XOR) The XOR instruction performs a bitwise XOR operation using the bits in Source A and Source B and places the result in the Destination.

To perform a logical XOR, see page 6-41.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Source A	SINT INT DINT	immediate tag	value to XOR with Source B
A SINT or INT tag converts to a DINT value by zero-fill.			
Source B	SINT INT DINT	immediate tag	value to XOR with Source A
A SINT or INT tag converts to a DINT value by zero-fill.			
Destination	SINT INT DINT	tag	stores the result



dest := sourceA XOR sourceB

Structured Text

Use XOR as an operator within an expression. This expression evaluates *sourceA XOR sourceB*.

See Structured Text Programming for information on the syntax of expressions within structured text.



Function Block

Operand:	Type:	Format:	Description:
XOR tag	FBD_LOGICAL	structure	XOR structure

FBD_LOGICAL Structure

Input Parameter:	Data Type:	Description:
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set.
SourceA	DINT	Value to XOR with SourceB. Valid = any integer
SourceB	DINT	Value to XOR with SourceA. Valid = any integer
Output Parameter:	Data Type:	Description:
EnableOut	BOOL	The instruction produced a valid result.
Dest	DINT	Result of the instruction. Arithmetic status flags are set for this output.

Description: When enabled, the instruction evaluates the XOR operation:

If the bit in Source A is:	And the bit in Source B is:	The bit in the Destination is:
0	0	0
0	1	1
1	0	1
1	1	0

If you mix integer data types, the instruction fills the upper bits of the smaller integer data types with 0s so that they are the same size as the largest data type.

Arithmetic Status Flags: Arithmetic status flags are affected.

Fault Conditions: none

Execution:

**Relay Ladder**

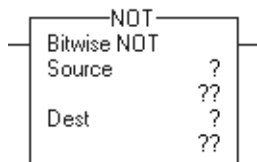
Condition:	Action:
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	The instruction performs a bitwise OR operation. The rung-condition-out is set to true.
postscan	The rung-condition-out is set to false.

Bitwise NOT (NOT)

The NOT instruction performs a bitwise NOT operation using the bits in the Source and places the result in the Destination.

To perform a logical NOT, see page 6-44.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Source	SINT INT DINT	immediate tag	value to NOT
A SINT or INT tag converts to a DINT value by zero-fill.			
Destination	SINT INT DINT	tag	stores the result

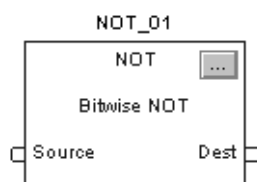


dest := NOT source

Structured Text

Use NOT as an operator within an expression. This expression evaluates NOT *source*.

See Structured Text Programming for information on the syntax of expressions within structured text.



Function Block

Operand:	Type:	Format:	Description:
NOT tag	FBD_LOGICAL	structure	NOT structure

FBD_LOGICAL Structure

Input Parameter:	Data Type:	Description:
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. default is set
Source	DINT	Value to NOT. valid = any integer
Output Parameter:	Data Type:	Description:
EnableOut	BOOL	The instruction produced a valid result.
Dest	DINT	Result of the instruction. Arithmetic status flags are set for this output.

Description: When enabled, the instruction evaluates the NOT operation:

If the bit in the Source is:	The bit in the Destination is:
0	1
1	0

If you mix integer data types, the instruction fills the upper bits of the smaller integer data types with 0s so that they are the same size as the largest data type.

Arithmetic Status Flags: Arithmetic status flags are affected.

Fault Conditions: none

Execution:



Relay Ladder

Condition:	Action:
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	The instruction performs a bitwise NOT operation. The rung-condition-out is set to true.
postscan	The rung-condition-out is set to false.



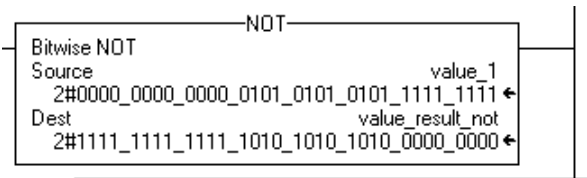
Function Block

Condition:	Action:
prescan	No action taken.
instruction first scan	No action taken.
instruction first run	No action taken.
EnableIn is cleared	EnableOut is cleared.
EnableIn is set	The instruction executes. EnableOut is set.
postscan	No action taken.

Example: When enabled, the NOT instruction performs a bitwise NOT operation on Source and places the result in Dest.

value_1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1
value_result_not	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0

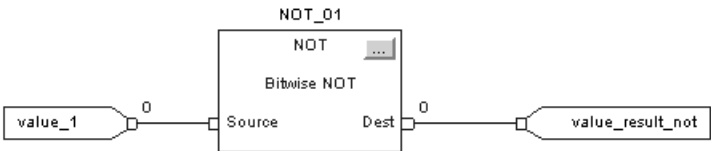
Relay Ladder



Structured Text

value_result_not := NOT value_1;

Function Block



Boolean AND (BAND)

The BAND instruction logically ANDs as many as 8 boolean inputs.

To perform a bitwise AND, see page 6-23.

Operands:

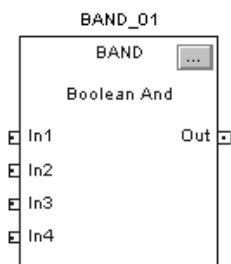


```
IF operandA AND operandB THEN
    <statement>;
END_IF;
```

Structured Text

Use AND or the ampersand sign “&” as an operator within an expression. The operands must be BOOL values or expressions that evaluate to BOOL values. This expression evaluates whether *operandA* and *operandB* are both set (true).

See Structured Text Programming for information on the syntax of expressions within structured text.



Function Block

Operand:	Type:	Format:	Description:
BAND tag	FBD_BOOLEAN_AND	structure	BAND structure

FBD_BOOLEAN_AND Structure

Input Parameter:	Data Type:	Description:
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set.
In1	BOOL	First boolean input. Default is set.
In2	BOOL	Second boolean input. Default is set.
In3	BOOL	Third boolean input. Default is set.
In4	BOOL	Fourth boolean input. Default is set.
In5	BOOL	Fifth boolean input. default is set.
In6	BOOL	Sixth boolean input. Default is set.
In7	BOOL	Seventh boolean input. Default is set.
In8	BOOL	Eighth boolean input. Default is set.
Output Parameter:	Data Type:	Description:
EnableOut	BOOL	Enable output.
Out	BOOL	The output of the instruction.

Description: The BAND instruction ANDs as many as eight boolean inputs. If an input is not used, it defaults to set (1).

Out = In1 AND In2 AND In3 AND In4 AND In5 AND In6 AND In7 AND In8

Arithmetic Status Flags: not affected

Fault Conditions: none

Execution:

Condition:	Function Block Action:
prescan	No action taken.
instruction first scan	No action taken.
instruction first run	No action taken.
EnableIn is cleared	EnableOut is cleared.
EnableIn is set	The instruction executes. EnableOut is set.
postscan	No action taken.

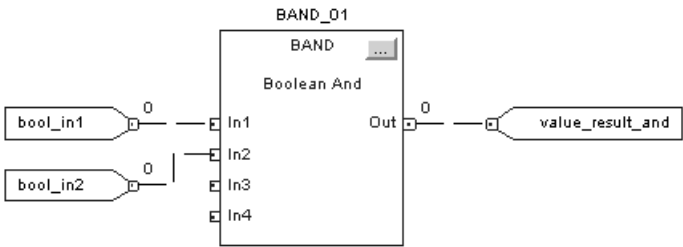
Example 1: This example ANDs *bool_in1* and *bool_in2* and places the result in *value_result_and*.

If bool_in1 is:	If bool_in2 is:	Then value_result_and is:
0	0	0
0	1	0
1	0	0
1	1	1

Structured Text

```
value_result_and := bool_in1 AND bool_in2;
```

Function Block



Example 2: If both *bool_in1* and *bool_in2* are set (true), *light1* is set (on). Otherwise, *light1* is cleared (off).

Structured Text

```
IF bool_in1 AND bool_in2 THEN
    light1 := 1;
ELSE
    light1 := 0;
END_IF;
```

Boolean OR (BOR)

The BOR instruction logically ORs as many as eight boolean inputs.

To perform a bitwise OR, see page 6-26.

Operands:

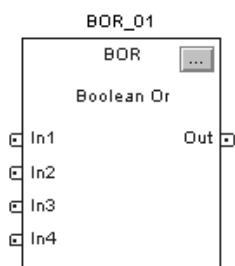


```
IF operandA OR operandB THEN
    <statement>;
END_IF;
```

Structured Text

Use OR as an operator within an expression. The operands must be BOOL values or expressions that evaluate to BOOL values. This expression evaluates whether *operandA* or *operandB* or both are set (true).

See Structured Text Programming for information on the syntax of expressions within structured text.



Function Block

Operand:	Type:	Format:	Description:
BOR tag	FBD_BOOLEAN_OR	structure	BOR structure

FBD_BOOLEAN_OR Structure

Input Parameter:	Data Type:	Description:
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set.
In1	BOOL	First boolean input. Default is cleared.
In2	BOOL	Second boolean input. Default is cleared.
In3	BOOL	Third boolean input. Default is cleared.
In4	BOOL	Fourth boolean input. Default is cleared.
In5	BOOL	Fifth boolean input. Default is cleared.
In6	BOOL	Sixth boolean input. Default is cleared.
In7	BOOL	Seventh boolean input. Default is cleared.
In8	BOOL	Eighth boolean input. Default is cleared.
Output Parameter:	Data Type:	Description:
EnableOut	BOOL	Enable output.
Out	BOOL	The output of the instruction.

Description: The BOR instruction ORs as many as eight boolean inputs. If an input is not used, it defaults to cleared (0).

Out = In1 OR In2 OR In3 OR In4 OR In5 OR In6 OR In7 OR In8

Arithmetic Status Flags: not affected

Fault Conditions: none

Execution:

Condition:	Function Block Action:
prescan	No action taken.
instruction first scan	No action taken.
instruction first run	No action taken.
EnableIn is cleared	EnableOut is cleared.
EnableIn is set	The instruction executes. EnableOut is set.
postscan	No action taken.

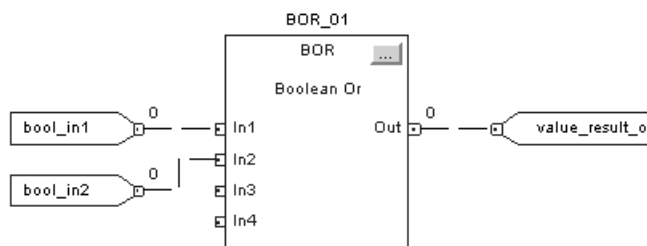
Example 1: This example ORs *bool_in1* and *bool_in2* and places the result in *value_result_or*.

If <i>bool_in1</i> is:	If <i>bool_in2</i> is:	Then <i>value_result_or</i> is:
0	0	0
0	1	1
1	0	1
1	1	1

Structured Text

```
value_result_or := bool_in1 OR bool_in2;
```

Function Block



Example 2: In this example, *light1* is set (on) if:

- only *bool_in1* is set (true).
- only *bool_in2* is set (true).
- both *bool_in1* and *bool_in2* are set (true).

Otherwise, *light1* is cleared (off).

Structured Text

```
IF bool_in1 OR bool_in2 THEN
    light1 := 1;
ELSE
    light1 := 0;
END_IF;
```


Boolean Exclusive OR (BXOR)

The BXOR performs an exclusive OR on two boolean inputs.

To perform a bitwise XOR, see page 6-29.

Operands:

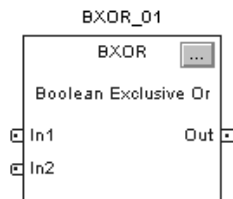


```
IF operandA XOR operandB THEN
    <statement>;
END_IF;
```

Structured Text

Use XOR as an operator within an expression. The operands must be BOOL values or expressions that evaluate to BOOL values. This expression evaluates whether only *operandA* or only *operandB* is set (true).

See Structured Text Programming for information on the syntax of expressions within structured text.



Function Block

Operand:	Type:	Format:	Description:
BXOR tag	FBD_BOOLEAN_XOR	structure	BXOR structure

FBD_BOOLEAN_XOR Structure

Input Parameter:	Data Type:	Description:
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set.
In1	BOOL	First boolean input. Default is cleared.
In2	BOOL	Second boolean input. Default is cleared.
Output Parameter:	Data Type:	Description:
EnableOut	BOOL	Enable output.
Out	BOOL	The output of the instruction.

Description: The BXOR instruction performs an exclusive OR on two boolean inputs.

Out = In1 XOR In2

Arithmetic Status Flags: not affected

Fault Conditions: none

Execution:

Condition:	Function Block Action:
prescan	No action taken.
instruction first scan	No action taken.
instruction first run	No action taken.
EnableIn is cleared	EnableOut is cleared.
EnableIn is set	The instruction executes. EnableOut is set.
postscan	No action taken.

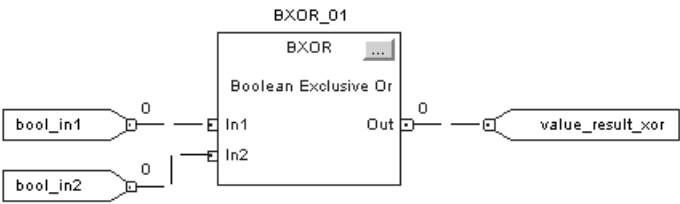
Example 1: This example performs an exclusive OR on *bool_in1* and *bool_in2* and places the result in *value_result_xor*.

If bool_in1 is:	If bool_in2 is:	Then value_result_xor is:
0	0	0
0	1	1
1	0	1
1	1	0

Structured Text

```
value_result_xor := bool_in1 XOR bool_in2;
```

Function Block



Example 2: In this example, *light1* is set (on) if

- only *bool_in1* is set (true).
- only *bool_in2* is set (true).

Otherwise, *light1* is cleared (off).

Structured Text

```
IF bool_in1 XOR bool_in2 THEN
    light1 := 1;
ELSE
    light1 := 0;
END_IF;
```

Boolean NOT (BNOT)

The BNOT instruction complements a boolean input.

To perform a bitwise NOT, see page 6-32.

Operands:

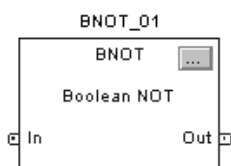


```
IF NOT operand THEN
    <statement>;
END_IF;
```

Structured Text

Use NOT as an operator within an expression. The operand must be a BOOL values or expressions that evaluate to BOOL values. This expression evaluates whether *operand* is cleared (false).

See Structured Text Programming for information on the syntax of expressions within structured text.



Function Block

Operand:	Type:	Format:	Description:
BNOT tag	FBD_BOOLEAN_NOT	structure	BNOT structure

FBD_BOOLEAN_NOT Structure

Input Parameter:	Data Type:	Description:
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set.
In	BOOL	Input to the instruction. Default is set.
Output Parameter:	Data Type:	Description:
EnableOut	BOOL	Enable output.
Out	BOOL	The output of the instruction.

Description: The BNOT instruction complements a boolean input.

Out = NOT In

Arithmetic Status Flags: not affected

Fault Conditions: none

Execution:

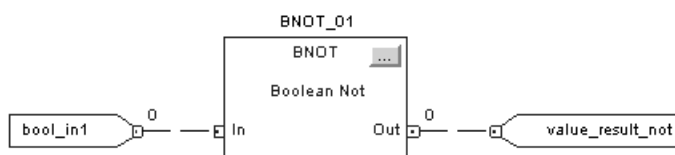
Condition:	Function Block Action:
prescan	No action taken.
instruction first scan	No action taken.
instruction first run	No action taken.
EnableIn is cleared	EnableOut is cleared.
EnableIn is set	The instruction executes. EnableOut is set.
postscan	No action taken.

Example 1: This example complements *bool_in1* and places the result in *value_result_not*.

If <i>bool_in1</i> is:	Then <i>value_result_not</i> is:
0	1
1	0

Structured Text

```
value_result_not := NOT bool_in1;
```

Function Block

Example 2: If *bool_in1* is cleared, *light1* is cleared (off). Otherwise, *light1* is set (on).

Structured Text

```
IF NOT bool_in1 THEN
    light1 := 0;
ELSE
    light1 := 1;
END_IF;
```

Notes:

Array (File)/Misc. Instructions

(FAL, FSC, COP, CPS, FLL, AVE, SRT, STD, SIZE)

Introduction

The file/miscellaneous instructions operate on arrays of data.

If you want to:	Use this instruction:	Available in these languages:	See page:
perform arithmetic, logic, shift, and function operations on values in arrays	FAL	relay ladder structured text ⁽¹⁾	7-7
search for and compare values in arrays	FSC	relay ladder	7-19
copy the contents of one array into another array	COP	relay ladder structured text	7-28
copy the contents of one array into another array without interruption	CPS	relay ladder structured text	7-28
fill an array with specific data	FLL	relay ladder structured text ⁽¹⁾	7-34
calculate the average of an array of values	AVE	relay ladder structured text ⁽¹⁾	7-38
sort one dimension of array data into ascending order	SRT	relay ladder structured text	7-43
calculate the standard deviation of an array of values	STD	relay ladder structured text ⁽¹⁾	7-48
find the size of a dimension of an array	SIZE	relay ladder structured text	7-53

⁽¹⁾ There is no equivalent structured text instruction. Use other structured text programming to achieve the same result. See the description for the instruction.

You can mix data types, but loss of accuracy and rounding error might occur and the instruction takes more time to execute. Check the S:V bit to see whether the result was truncated.

For relay ladder instructions, **bold** data types indicate optimal data types. An instruction executes faster and requires less memory if all the operands of the instruction use the same optimal data type, typically DINT or REAL.

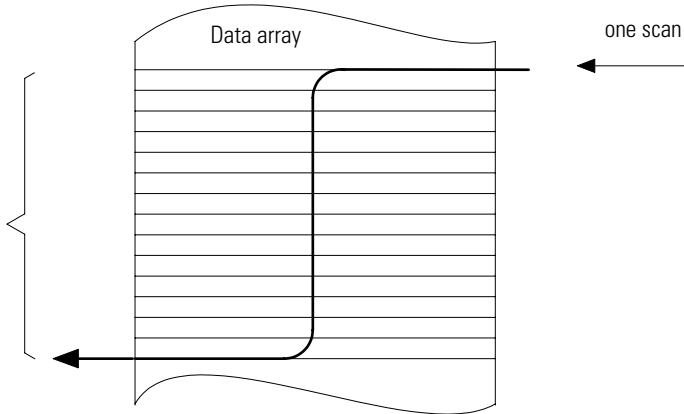
Selecting Mode of Operation

For FAL and FSC instructions, the mode tells the controller how to distribute the array operation.

If you want to:	Select this mode:
operate on all of the specified elements in an array before continuing on to the next instruction	All
distribute array operation over a number of scans enter the number of elements to operate on per scan (1-2147483647)	Numerical
manipulate one element of the array each time the rung-condition-in goes from false to true	Incremental

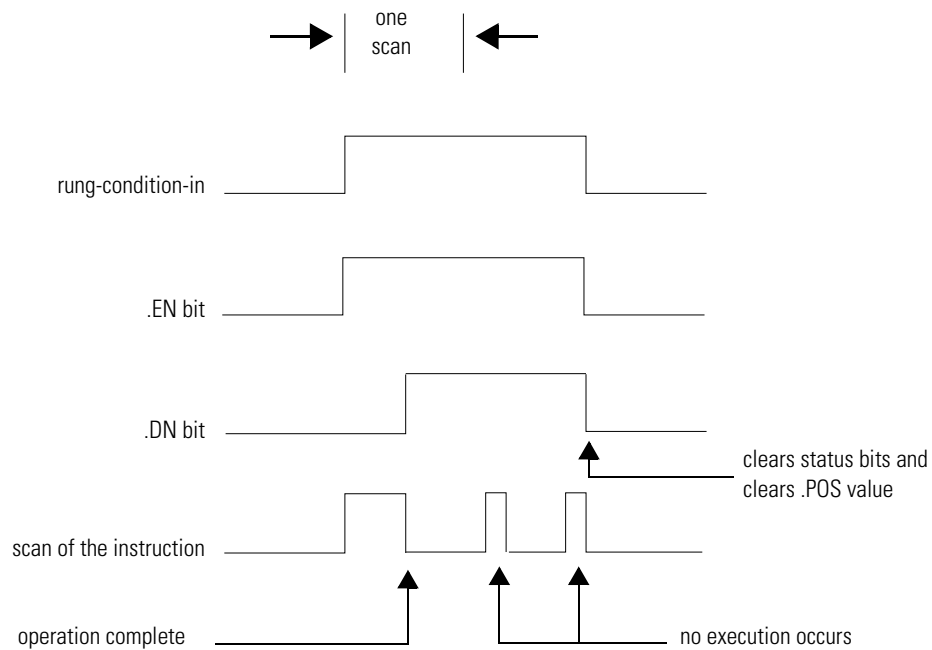
All mode

In All mode, all the specified elements in the array are operated on before continuing on to the next instruction. The operation begins when the instruction's rung-condition-in goes from false to true. The position (.POS) value in the control structure points to the element in the array that the instruction is currently using. Operation stops when the .POS value equals the .LEN value.



16639

The following timing diagram shows the relationship between status bits and instruction operation. When the instruction execution is complete, the .DN bit is set. The .DN bit, the .EN bit, and the .POS value are cleared when the rung-condition-in is false. Only then can another execution of the instruction be triggered by a false-to-true transition of rung-condition-in.

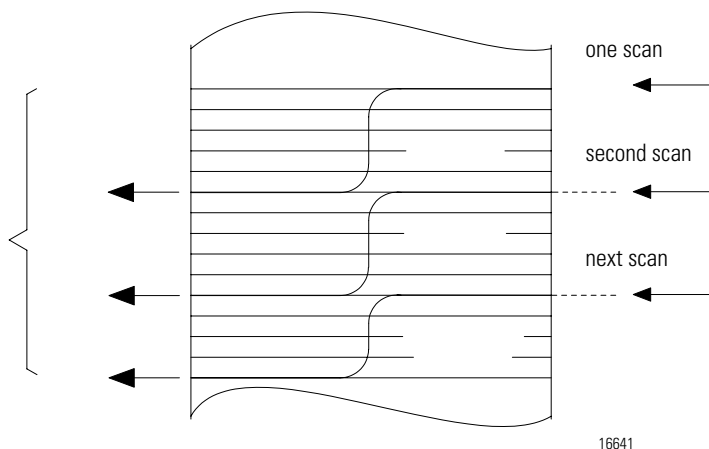


40010

Numerical mode

Numerical mode distributes the array operation over a number of scans. This mode is useful when working with non-time-critical data or large amounts of data. You enter the number of elements to operate on for each scan, which keeps scan time shorter.

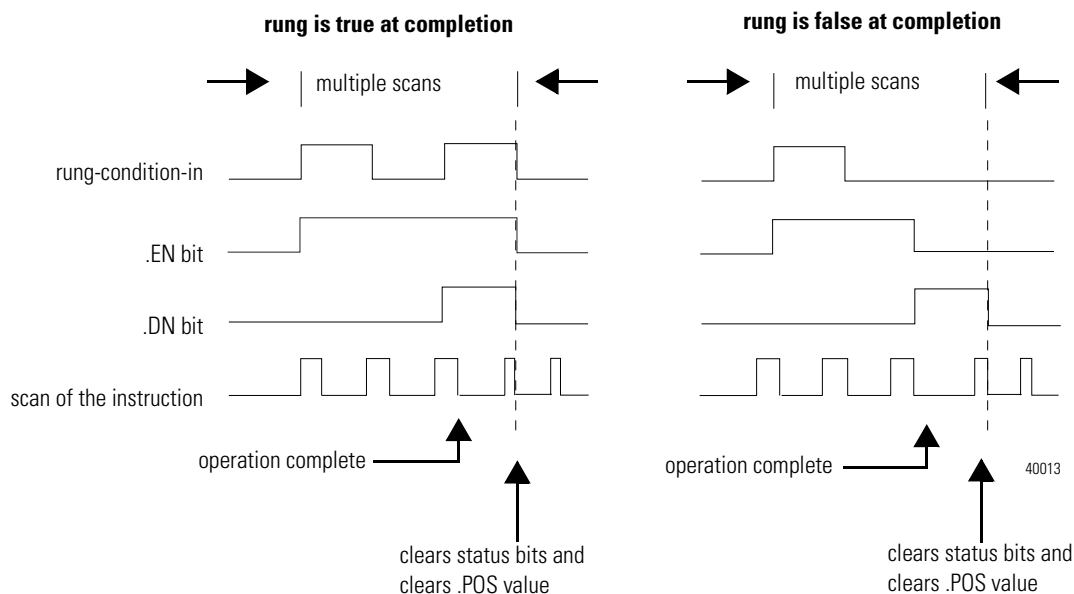
Execution is triggered when the rung-condition-in goes from false to true. Once triggered, the instruction is executed each time it is scanned for the number of scans necessary to complete operating on the entire array. Once triggered, rung-condition-in can change repeatedly without interrupting execution of the instruction.



IMPORTANT

Avoid using the results of a file instruction operating in numerical mode until the .DN bit is set.

The following timing diagram shows the relationship between status bits and instruction operation. When the instruction execution is complete, the .DN bit is set.

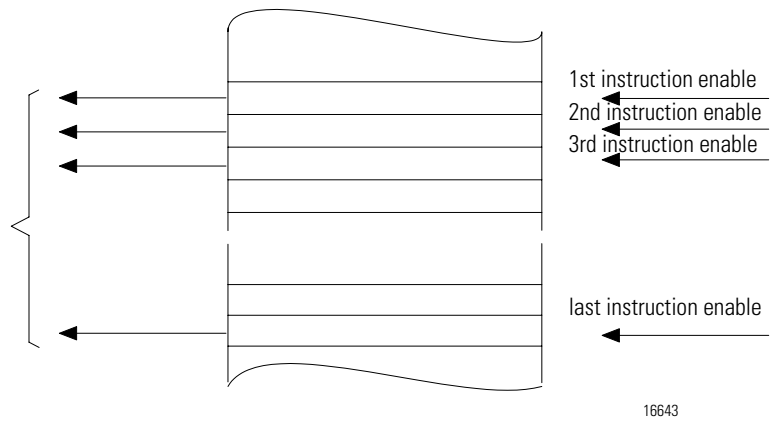


If the rung-condition-in is true at completion, the .EN and .DN bit are set until the rung-condition-in goes false. When the rung-condition-in goes false, these bits are cleared and the .POS value is cleared.

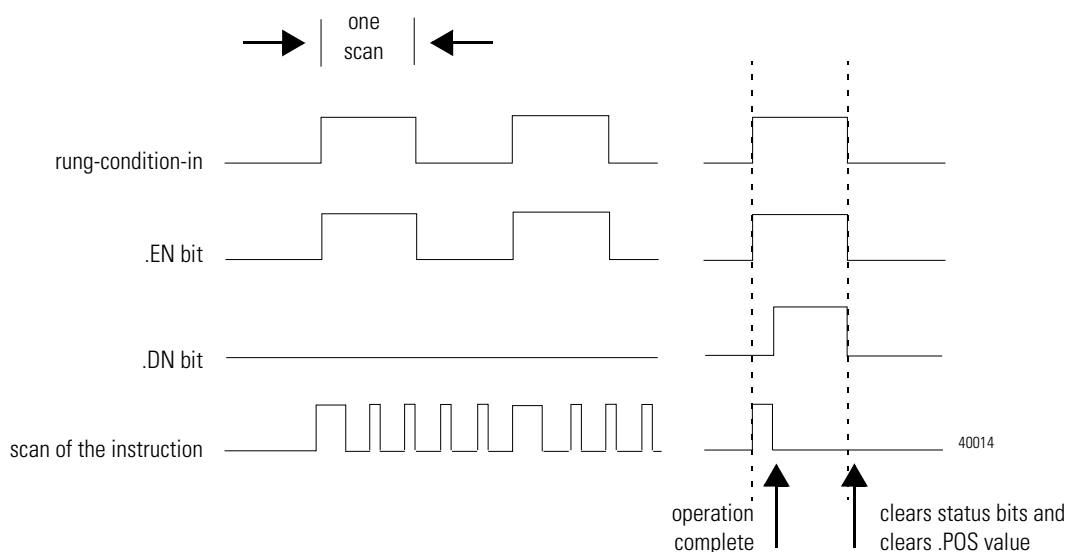
If the rung-condition-in is false at completion, the .EN bit is cleared immediately. One scan after the .EN bit is cleared, the .DN bit and the .POS value are cleared.

Incremental mode

Incremental mode manipulates one element of the array each time the instruction's rung-condition-in goes from false to true.



The following timing diagram shows the relationship between status bits and instruction operation. Execution occurs only in a scan in which the rung-condition-in goes from false to true. Each time this occurs, only one element of the array is manipulated. If the rung-condition-in remains true for more than one scan, the instruction only executes during the first scan.



The .EN bit is set when rung-condition-in is true. The .DN bit is set when the last element in the array has been manipulated. When the last element has been manipulated and the rung-condition-in goes false, the .EN bit, the .DN bit, and the .POS value are cleared.

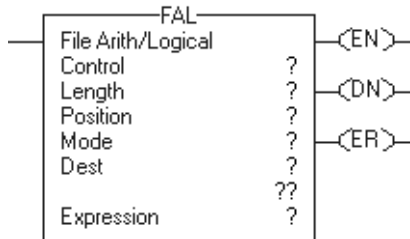
The difference between incremental mode and numerical mode at a rate of one element per scan is:

- Numerical mode with any number of elements per scan requires only one false-to-true transition of the rung-condition-in to start execution. The instruction continues to execute the specified number of elements each scan until completion regardless of the state of the rung-condition-in.
- Incremental mode requires the rung-condition-in to change from false to true to manipulate one element in the array.

File Arithmetic and Logic (FAL)

The FAL instruction performs copy, arithmetic, logic, and function operations on data stored in an array.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Control	CONTROL	tag	control structure for the operation
Length	DINT	immediate	number of elements in the array to be manipulated
Position	DINT	immediate	current element in array initial value is typically 0
Mode	DINT	immediate	how to distribute the operation select INC, ALL, or enter a number
Destination	SINT INT DINT REAL	tag	tag to store the result
Expression	SINT INT DINT REAL	immediate tag	an expression consisting of tags and/or immediate values separated by operators

A SINT or INT tag converts to a DINT value by sign-extension.



Structured Text

Structured text does not have an FAL instruction, but you can achieve the same results using a SIZE instruction and a FOR...DO or other loop construct.

```
SIZE(destination,0,length-1);
FOR position = 0 TO length DO
    destination[position] := numeric_expression;
END_FOR;
```

See Appendix C for information on the syntax of constructs within structured text.

CONTROL Structure

Mnemonic:	Data Type:	Description:
.EN	BOOL	The enable bit indicates that the FAL instruction is enabled.
.DN	BOOL	The done bit is set when the instruction has operated on the last element (.POS = .LEN).
.ER	BOOL	The error bit is set if the expression generates an overflow (S:V is set). The instruction stops executing until the program clears the .ER bit. The .POS value contains the position of the element that caused the overflow.
.LEN	DINT	The length specifies the number of elements in the array on which the FAL instruction operates.
.POS	DINT	The position contains the position of the current element that the instruction is accessing.

Description: The FAL instruction performs the same operations on arrays as the CPT instruction performs on elements.

The examples that start on page 7-14 show how to use the .POS value to step through an array. If a subscript in the expression of the Destination is out of range, the FAL instruction generates a major fault (type 4, code 20).

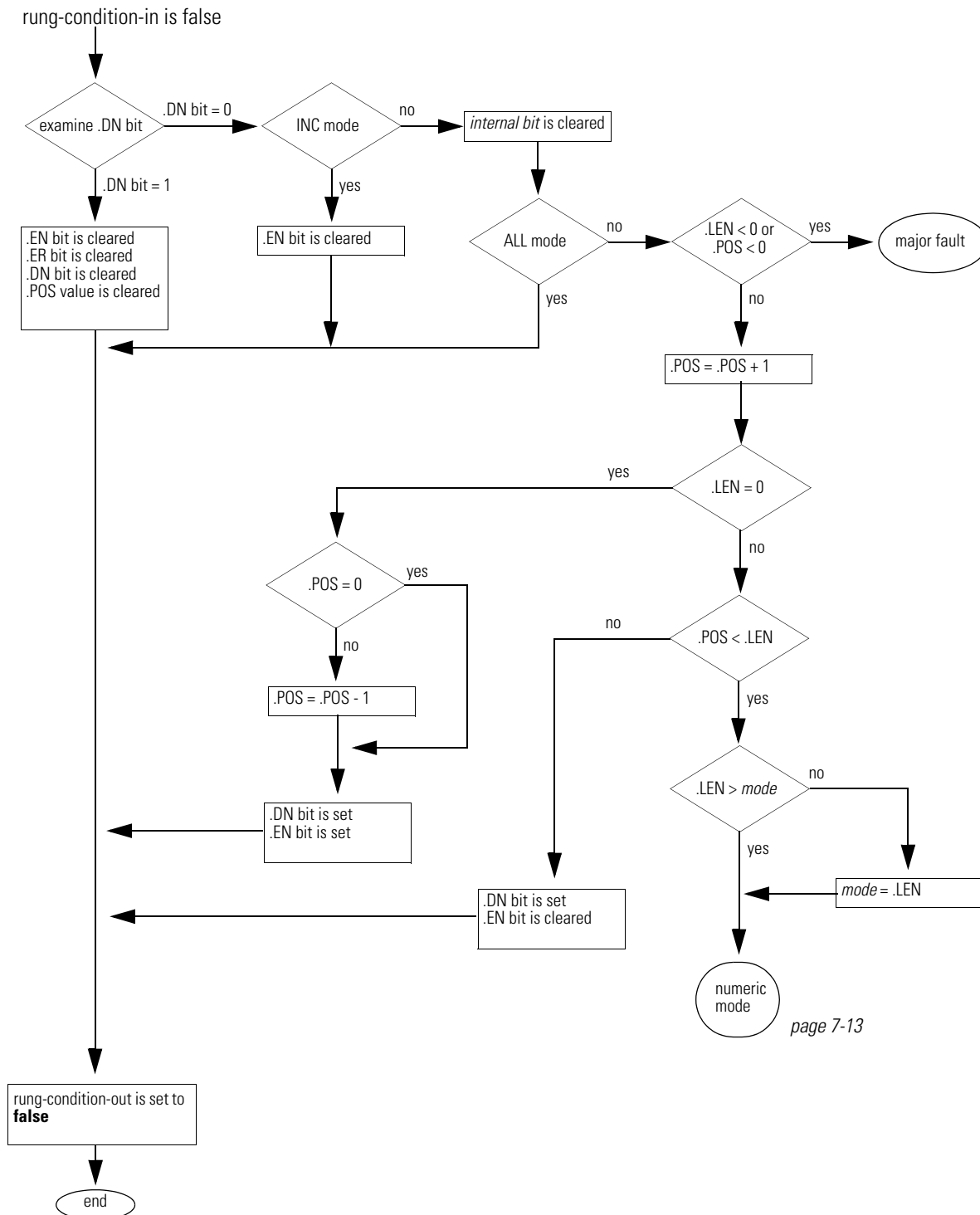
Arithmetic Status Flags: Arithmetic status flags are affected.

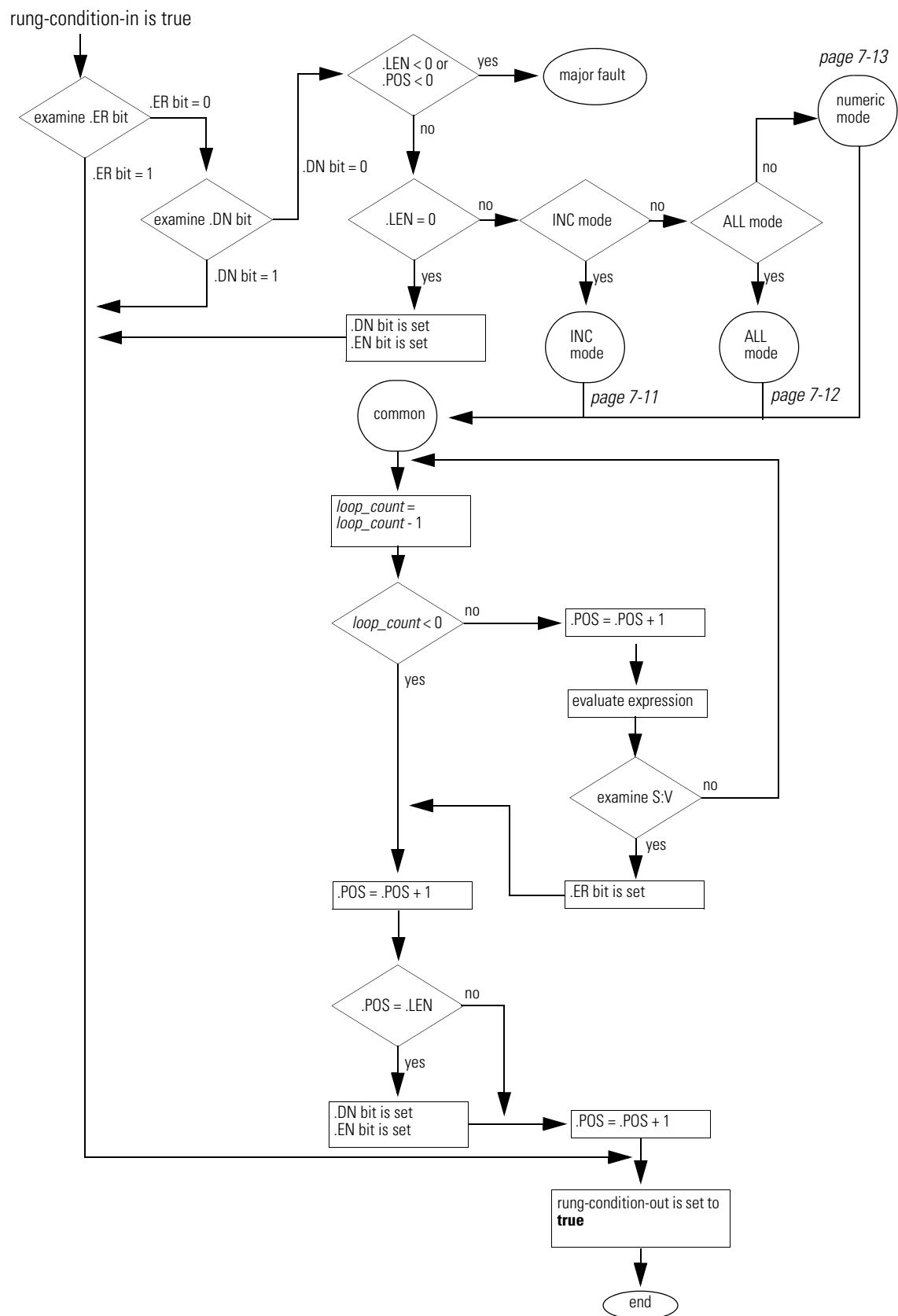
Fault Conditions:

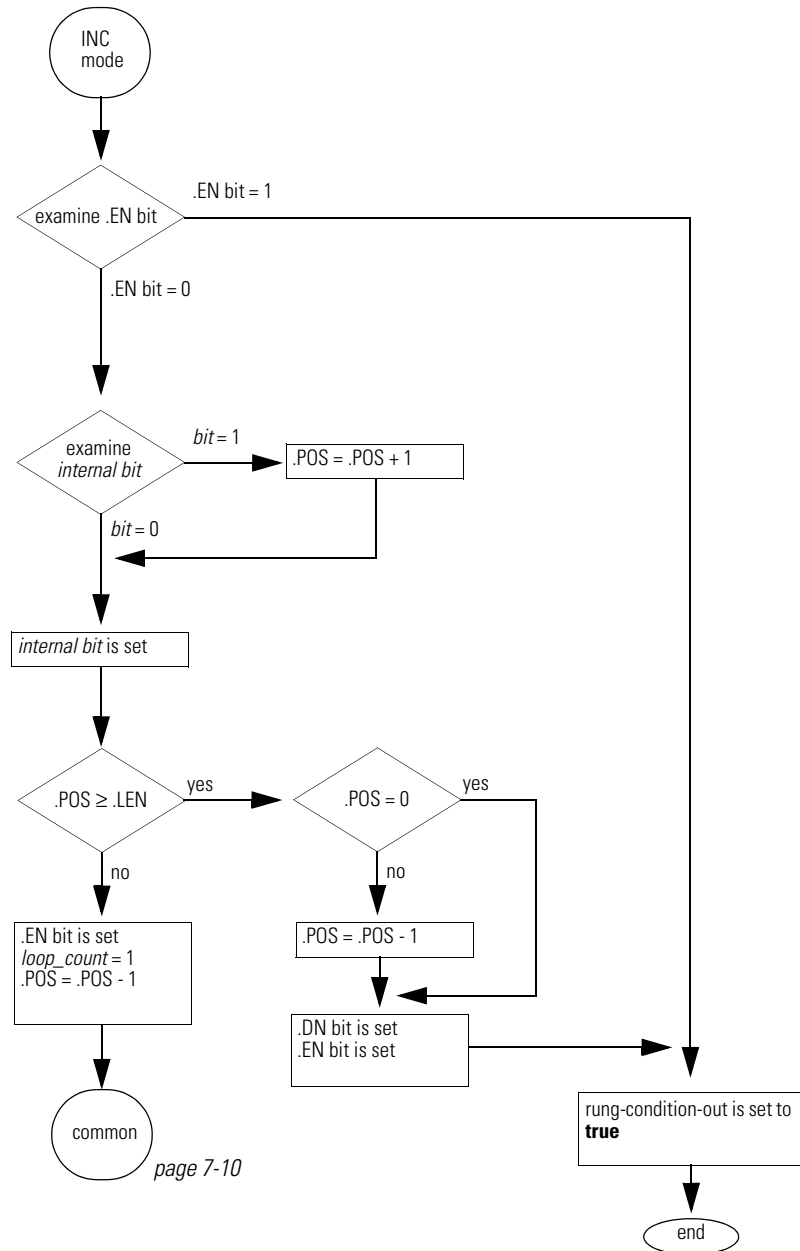
A major fault will occur if:	Fault type:	Fault code:
subscript is out of range	4	20
.POS < 0 or .LEN < 0	4	21

Execution:

Condition:	Relay Ladder Action:
prescan	The rung-condition-out is set to false.

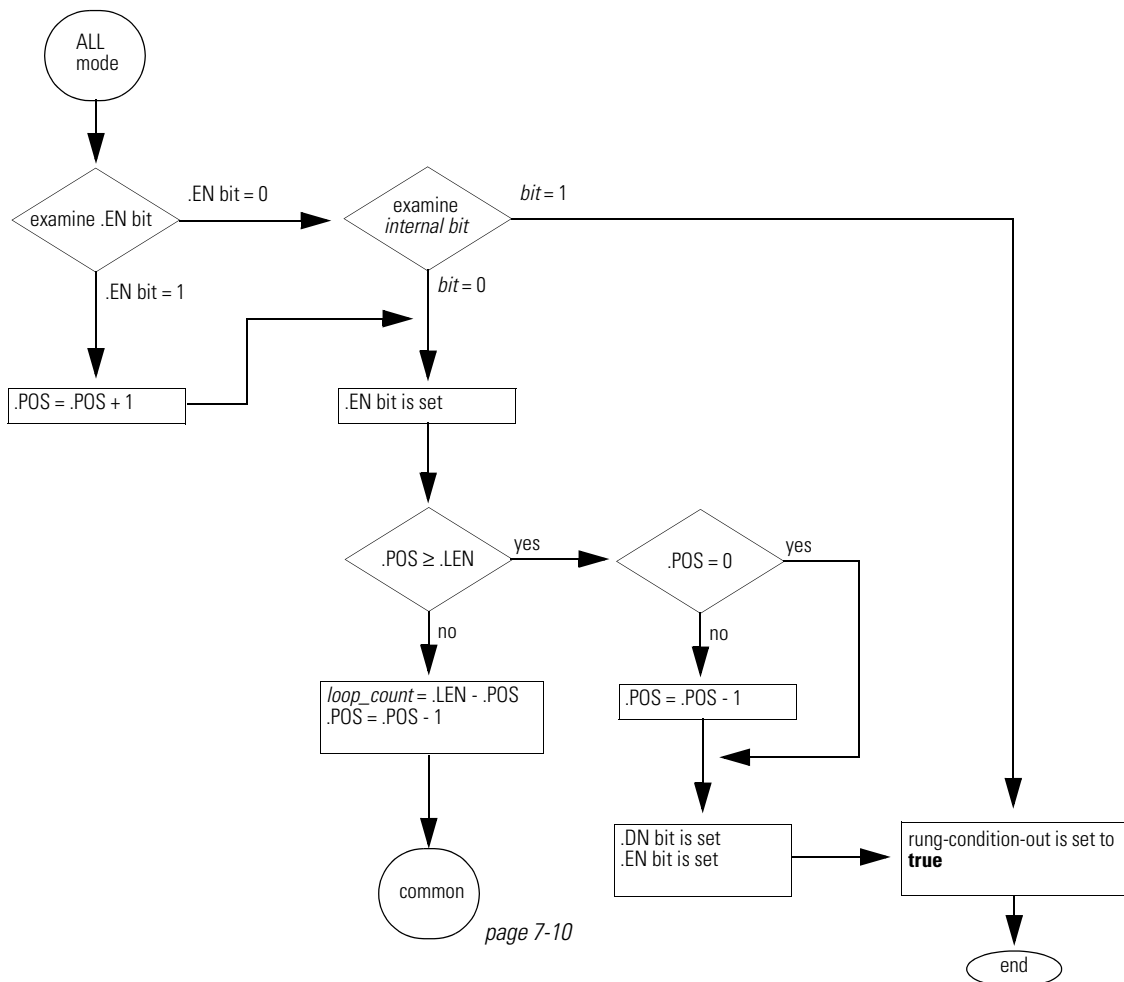


Condition:**Relay Ladder Action:**

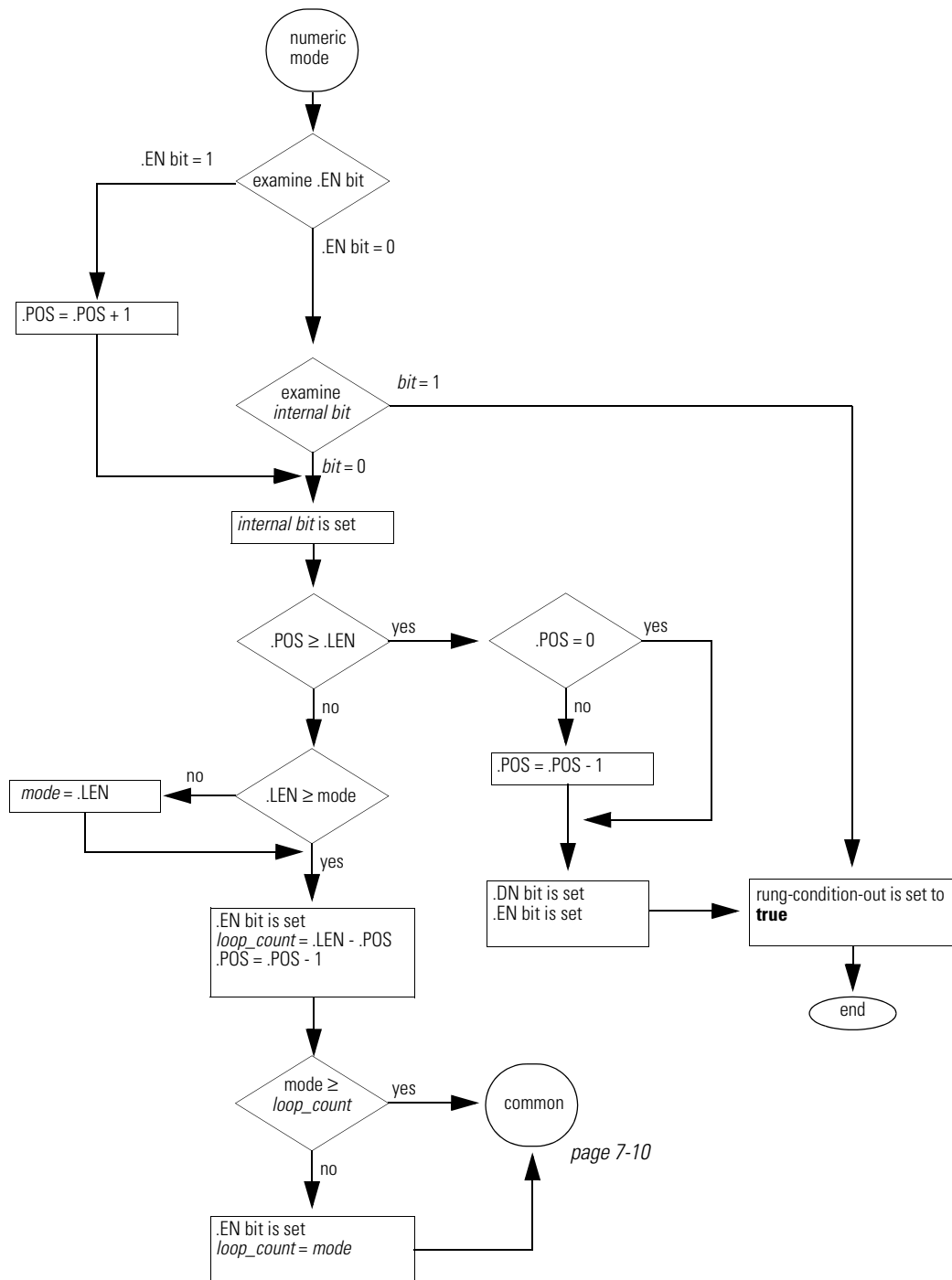
Condition:**Relay Ladder Action:**

Condition:

Relay Ladder Action:



page 7-10

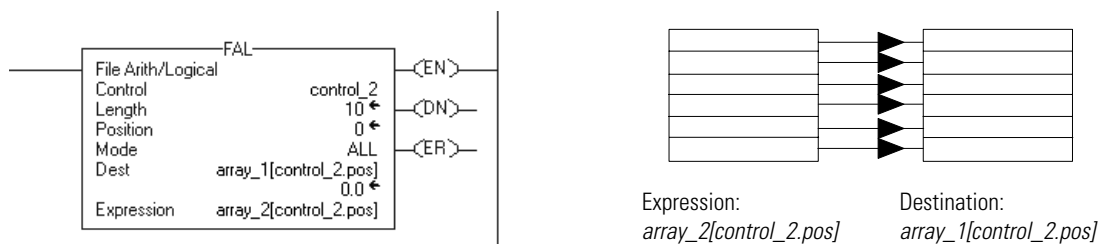
Condition:**Relay Ladder Action:**

postscan

The rung-condition-out is set to false.

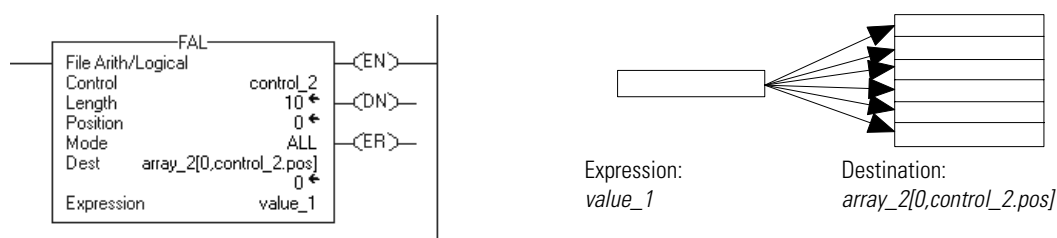
Example 1: When enabled, the FAL instruction copies each element of *array_2* into the same position within *array_1*.

array-to-array copy



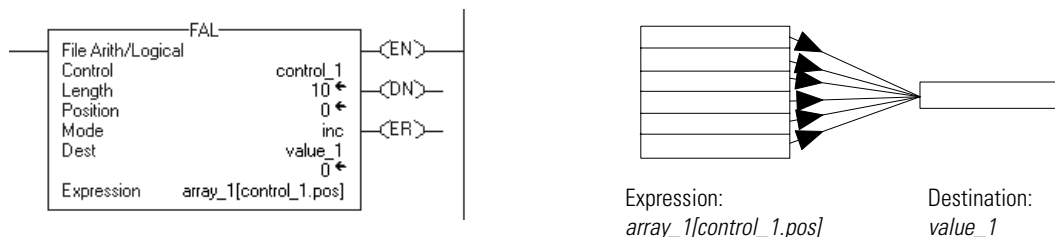
Example 2: When enabled, the FAL instruction copies *value_1* into the first 10 positions of the second dimension of *array_2*.

element-to-array copy



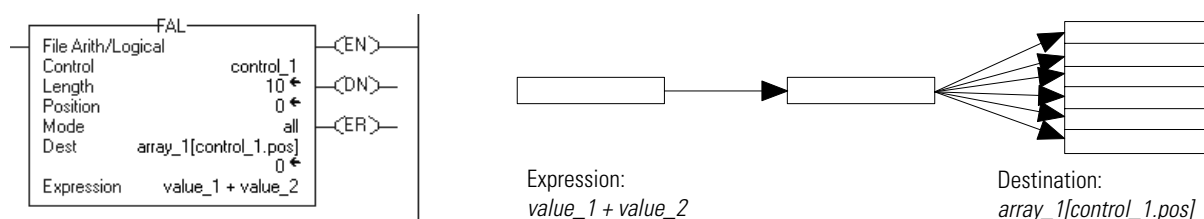
Example 3: Each time the FAL instruction is enabled, it copies the current value of *array_1* to *value_1*. The FAL instruction uses incremental mode, so only one array value is copied each time the instruction is enabled. The next time the instruction is enabled, the instruction overwrites *value_1* with the next value in *array_1*.

array-to-element copy

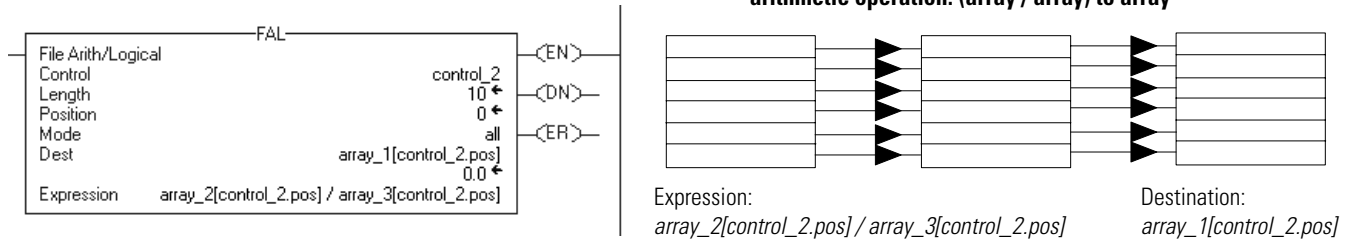


Example 4: When enabled, the FAL instruction adds *value_1* and *value_2* and stores the result in the current position of *array_1*.

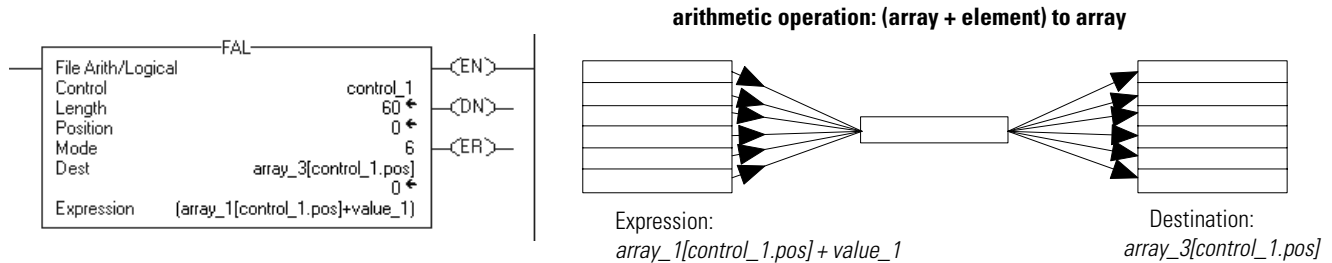
arithmetic operation: (element + element) to array



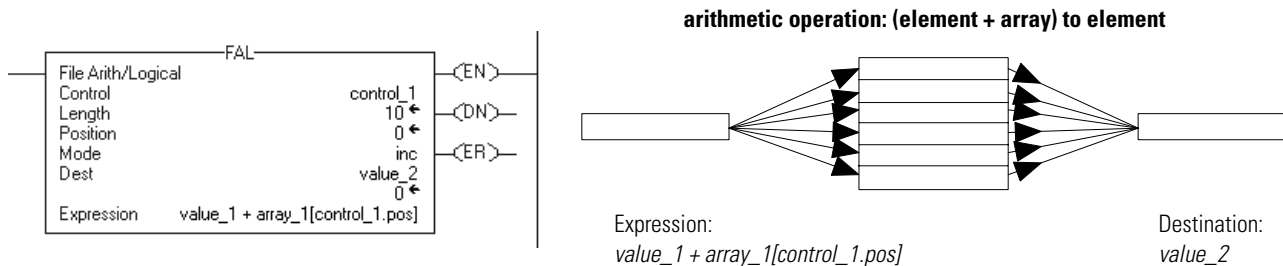
Example 5: When enabled, the FAL instruction divides the value in the current position of *array_2* with the value in the current position of *array_3* and stores the result in the current position of *array_1*.



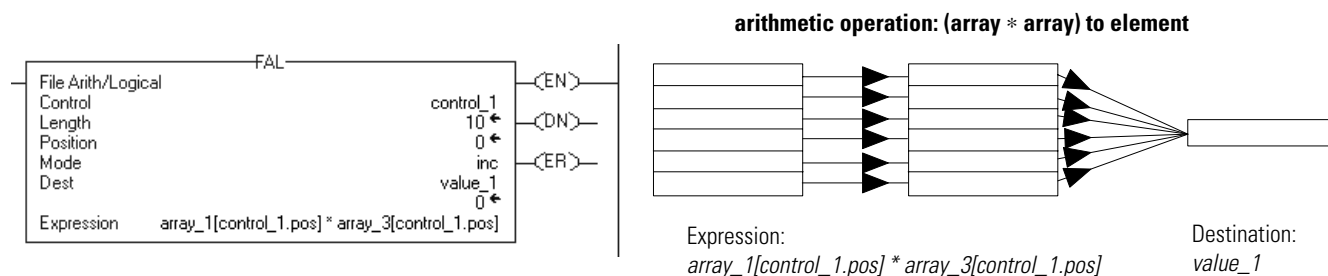
Example 6: When enabled, the FAL instruction adds the value at the current position in *array_1* to *value_1* and stores the result in the current position in *array_3*. The instruction must execute 10 times for the entire *array_1* and *array_3* to be manipulated.



Example 7: Each time the FAL instruction is enabled, it adds *value_1* to the current value of *array_1* and stores the result in *value_2*. The FAL instruction uses incremental mode, so only one array value is added to *value_1* each time the instruction is enabled. The next time the instruction is enabled, the instruction overwrites *value_2*.



Example 8: When enabled, the FAL instruction multiplies the current value of *array_1* by the current value of *array_3* and stores the result in *value_1*. The FAL instruction uses incremental mode, so only one pair of array values is multiplied each time the instruction is enabled. The next time the instruction is enabled, the instruction overwrites *value_1*.



FAL expressions

You program expressions in FAL instructions the same as expressions in CPT instructions. Use the following sections for information on valid operators, format, and order of operation, which are common to both instructions.

Valid operators

Operator:	Description:	Optimal:
+	add	DINT, REAL
-	subtract/negate	DINT, REAL
*	multiply	DINT, REAL
/	divide	DINT, REAL
**	exponent (x to y)	DINT, REAL
ABS	absolute value	DINT, REAL
ACS	arc cosine	REAL
AND	bitwise AND	DINT
ASN	arc sine	REAL
ATN	arc tangent	REAL
COS	cosine	REAL
DEG	radians to degrees	DINT, REAL
FRD	BCD to integer	DINT
LN	natural log	REAL

Operator:	Description:	Optimal:
LOG	log base 10	REAL
MOD	modulo-divide	DINT, REAL
NOT	bitwise complement	DINT
OR	bitwise OR	DINT
RAD	degrees to radians	DINT, REAL
SIN	sine	REAL
SQR	square root	DINT, REAL
TAN	tangent	REAL
TOD	integer to BCD	DINT
TRN	truncate	DINT, REAL
XOR	bitwise exclusive OR	DINT

Formatting expressions

For each operator that you use in an expression, you have to provide one or two operands (tags or immediate values). Use the following table to format operators and operands within an expression:

For operators that operate on:	Use this format:	Examples:
one operand	operator(operand)	ABS(<i>tag_a</i>)
two operands	operand_a operator operand_b	<ul style="list-style-type: none"> <i>tag_b</i> + 5 <i>tag_c</i> AND <i>tag_d</i> (<i>tag_e</i> ** 2) MOD (<i>tag_f</i> / <i>tag_g</i>)

Determining the order of operation

The operations you write into the expression are performed by the instruction in a prescribed order, not necessarily the order you write them. You can override the order of operation by grouping terms within parentheses, forcing the instruction to perform an operation within the parentheses ahead of other operations.

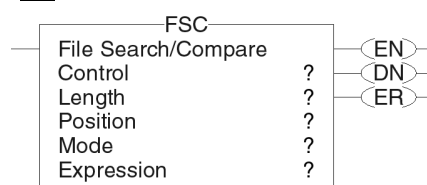
Operations of equal order are performed from left to right.

Order:	Operation:
1.	()
2.	ABS, ACS, ASN, ATN, COS, DEG, FRD, LN, LOG, RAD, SIN, SQR, TAN, TOD, TRN
3.	**
4.	– (negate), NOT
5.	*, /, MOD
6.	– (subtract), +
7.	AND
8.	XOR
9.	OR

File Search and Compare (FSC)

The FSC instruction compares values in an array, element by element.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Control	CONTROL	tag	control structure for the operation
Length	DINT	immediate	number of elements in the array to be manipulated
Position	DINT	immediate	offset into array initial value is typically 0

CONTROL Structure

Mnemonic:	Data Type:	Description:
.EN	BOOL	The enable bit indicates that the FSC instruction is enabled.
.DN	BOOL	The done bit is set when the instruction has operated on the last element (.POS = .LEN).
.ER	BOOL	The error bit is not modified.
.IN	BOOL	The inhibit bit indicates that the FSC instruction detected a true comparison. You must clear this bit to continue the search operation.
.FD	BOOL	The found bit indicates that the FSC instruction detected a true comparison.
.LEN	DINT	The length specifies the number of elements in the array on which the instruction operates.
.POS	DINT	The position contains the position of the current element that the instruction is accessing.

Description: When the FSC instruction is enabled and the comparison is true, the instruction sets the .FD bit and the .POS bit reflects the array position where the instruction found the true comparison. The instruction sets the .IN bit to prevent further searching.

Arithmetic Status Flags: Arithmetic status flags are affected.

Fault Conditions:

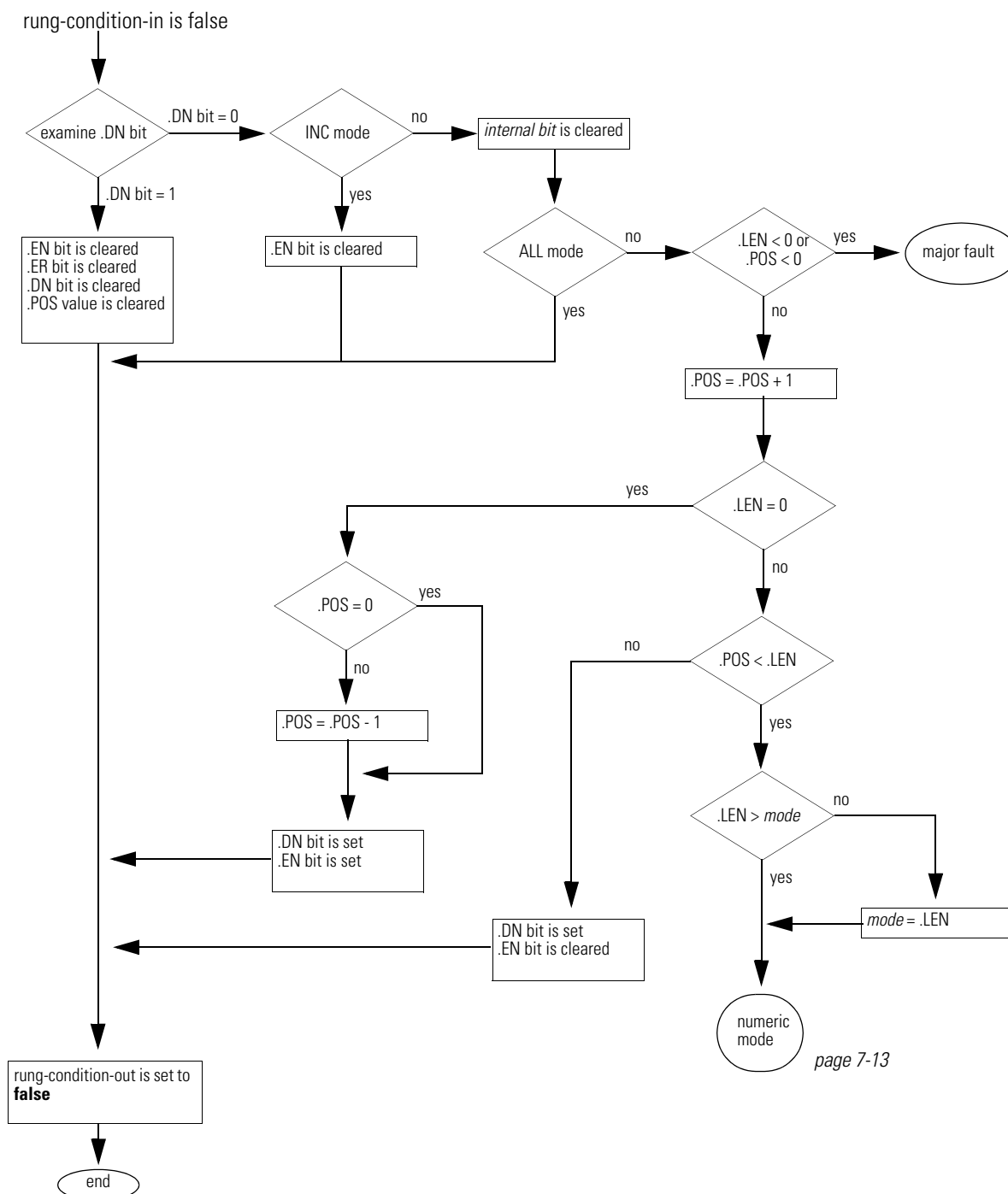
A major fault will occur if:	Fault type:	Fault code:
.POS < 0 or .LEN < 0	4	21

Execution:**Condition:**

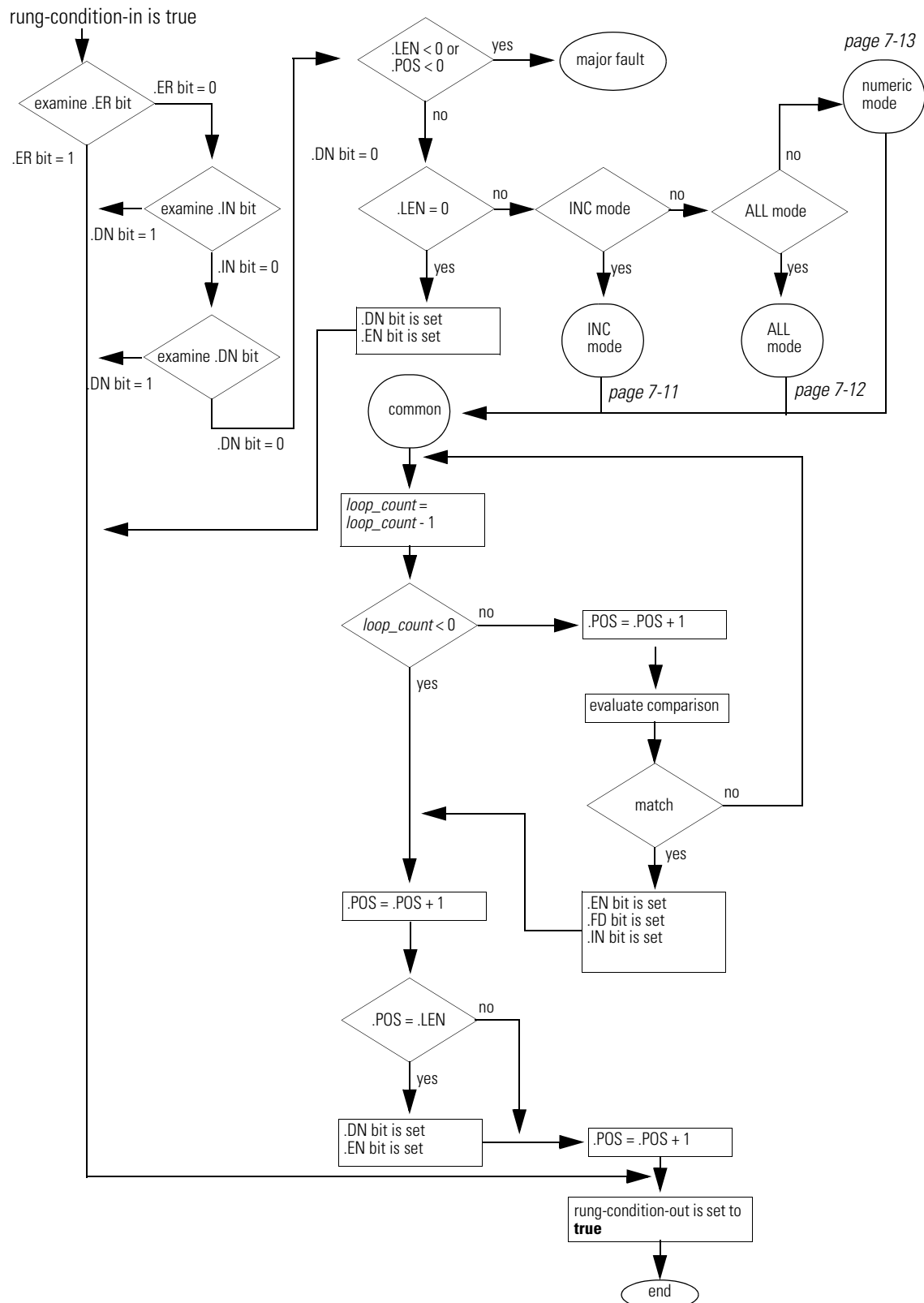
prescan

Relay Ladder Action:

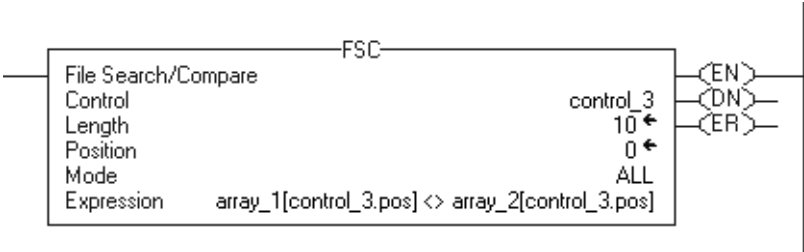
The rung-condition-out is set to false.



page 7-13

Condition:**Relay Ladder Action:**

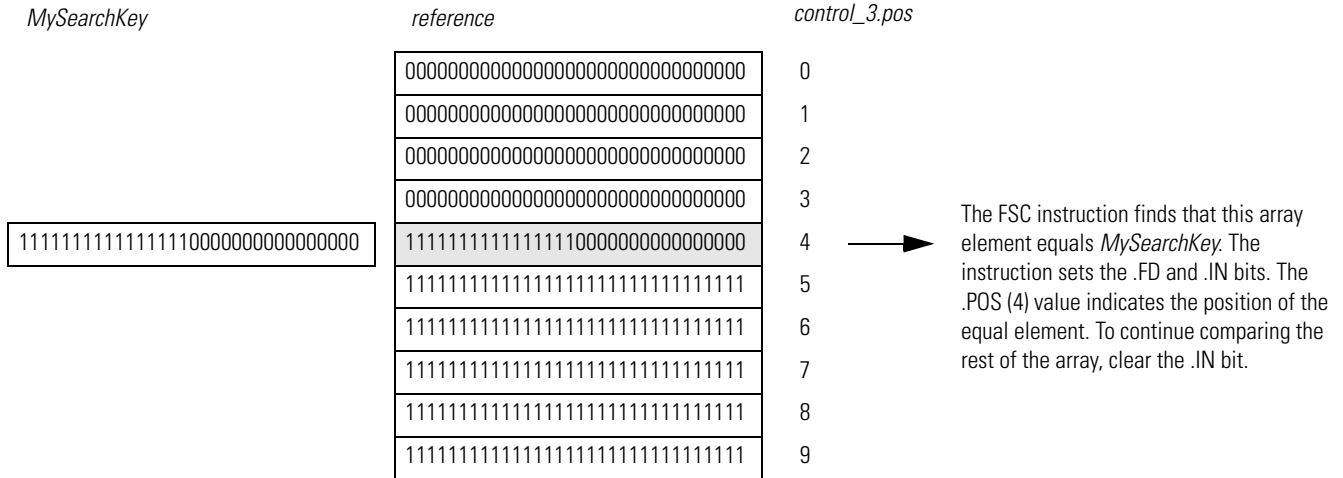
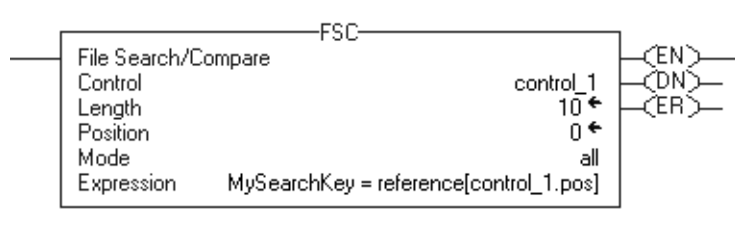
Example 1: Search for a match between two arrays. When enabled, the FSC instruction compares each of the first 10 elements in *array_1* to the corresponding elements in *array_2*.



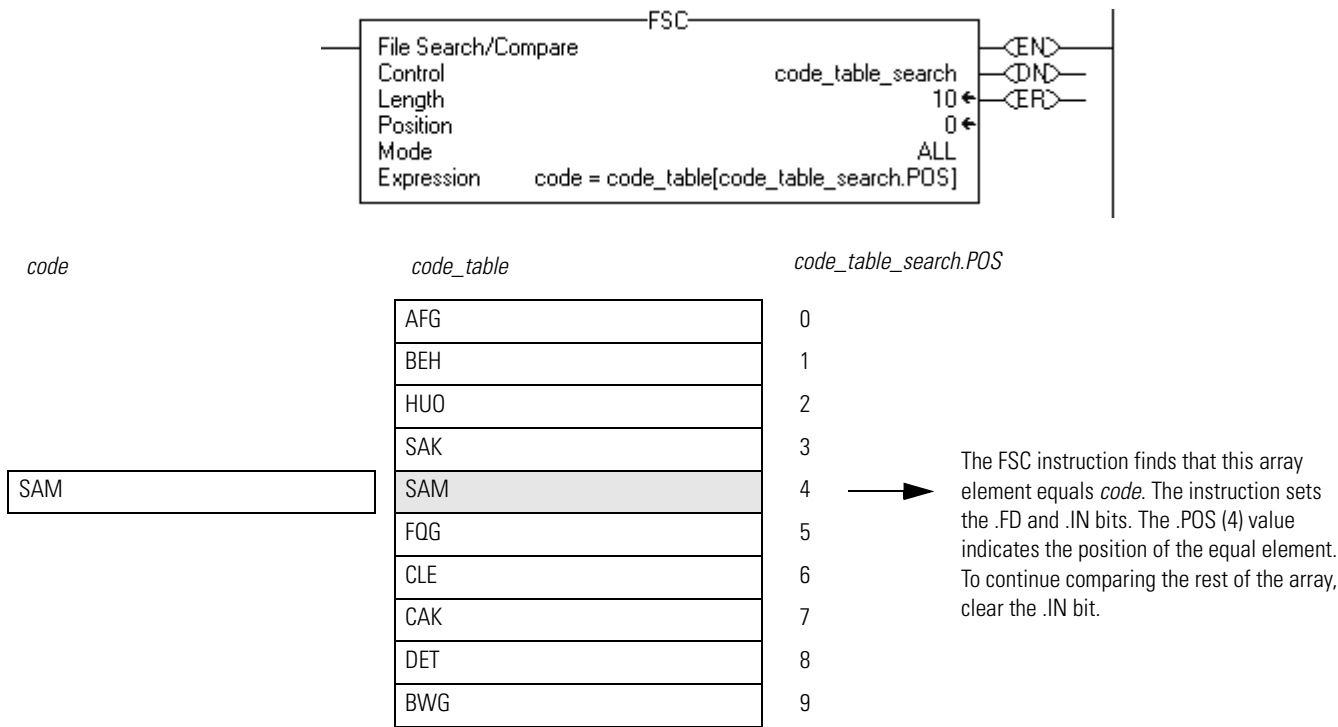
array_1	array_2	control_3.pos
00000000000000000000000000000000	00000000000000000000000000000000	0
00000000000000000000000000000000	00000000000000000000000000000000	1
00000000000000000000000000000000	00000000000000000000000000000000	2
00000000000000000000000000000000	00000000000000000000000000000000	3
00000000000000000000000000000000	11111111111111111111111111111111	4
11111111111111111111111111111111	11111111111111111111111111111111	5
11111111111111111111111111111111	11111111111111111111111111111111	6
11111111111111111111111111111111	11111111111111111111111111111111	7
11111111111111111111111111111111	11111111111111111111111111111111	8
11111111111111111111111111111111	11111111111111111111111111111111	9

→ The FSC instruction finds that these elements are not equal. The instruction sets the .FD and .IN bits. The .POS value (4) indicates the position of the elements that are not equal. To continue comparing the rest of the array, clear the .IN bit.

Example 2: Search for a match in an array. When enabled, the FSC instruction compares the *MySearchKey* to 10 elements in *array_1*.



Example 3: Search for a string in an array of strings. When enabled, the FSC instruction compares the characters in *code* to 10 elements in *code_table*.



FSC expressions

You program expressions in FSC instructions the same as expressions in CMP instructions. Use the following sections for information on valid operators, format, and order of operation, which are common to both instructions.

Valid operators

Operator:	Description:	Optimal:
+	add	DINT, REAL
-	subtract/negate	DINT, REAL
*	multiply	DINT, REAL
/	divide	DINT, REAL
=	equal	DINT, REAL
<	less than	DINT, REAL
<=	less than or equal	DINT, REAL
>	greater than	DINT, REAL
>=	greater than or equal	DINT, REAL
<>	not equal	DINT, REAL
**	exponent (x to y)	DINT, REAL
ABS	absolute value	DINT, REAL
ACS	arc cosine	REAL
AND	bitwise AND	DINT
ASN	arc sine	REAL
ATN	arc tangent	REAL
COS	cosine	REAL

Operator:	Description:	Optimal:
DEG	radians to degrees	DINT, REAL
FRD	BCD to integer	DINT
LN	natural log	REAL
LOG	log base 10	REAL
MOD	modulo-divide	DINT, REAL
NOT	bitwise complement	DINT
OR	bitwise OR	DINT
RAD	degrees to radians	DINT, REAL
SIN	sine	REAL
SQR	square root	DINT, REAL
TAN	tangent	REAL
TOD	integer to BCD	DINT
TRN	truncate	DINT, REAL
XOR	bitwise exclusive OR	DINT

Formatting expressions

For each operator that you use in an expression, you have to provide one or two operands (tags or immediate values). Use the following table to format operators and operands within an expression:

For operators that operate on:	Use this format:	Examples:
one operand	operator(operand)	<i>ABS(tag_a)</i>
two operands	operand_a operator operand_b	<ul style="list-style-type: none"> <i>tag_b + 5</i> <i>tag_c AND tag_d</i> <i>(tag_e ** 2) MOD (tag_f / tag_g)</i>

Determining the order of operation

The operations you write into the expression are performed by the instruction in a prescribed order, not necessarily the order you write them. You can override the order of operation by grouping terms within parentheses, forcing the instruction to perform an operation within the parentheses ahead of other operations.

Operations of equal order are performed from left to right.

Order:	Operation:
1.	()
2.	ABS, ACS, ASN, ATN, COS, DEG, FRD, LN, LOG, RAD, SIN, SQR, TAN, TOD, TRN
3.	**
4.	– (negate), NOT
5.	*, /, MOD
6.	<, <=, >, >=, =
7.	– (subtract), +
8.	AND
9.	XOR
10.	OR

Using strings in an expression

To use strings of ASCII characters in an expression, follow these guidelines:

- An expression lets you compare two string tags.
- You *cannot* enter ASCII characters directly into the expression.
- Only the following operators are permitted

Operator:	Description:
=	equal
<	less than
<=	less than or equal
>	greater than
>=	greater than or equal
<>	not equal

- Strings are equal if their characters match.
- ASCII characters are case sensitive. Upper case “A” (\$41) is *not* equal to lower case “a” (\$61).
- The hexadecimal values of the characters determine if one string is less than or greater than another string. For the hex code of a character, see the back cover of this manual.
- When the two strings are sorted as in a telephone directory, the order of the strings determines which one is greater.

ASCII Characters	Hex Codes
1ab	\$31\$61\$62
1b	\$31\$62
A	\$41
AB	\$41\$42
B	\$42
a	\$61
ab	\$61\$62

l
e
s
s
e
r

↑

g
r
e
a
t
e
r

↓

—

AB < B

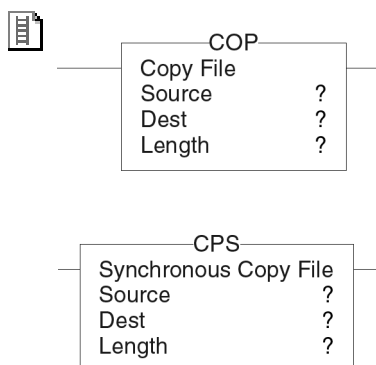
—

a > B

Copy File (COP) Synchronous Copy File (CPS)

The COP and CPS instructions copy the value(s) in the Source to the Destination. The Source remains unchanged.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Source	SINT INT DINT REAL string structure	tag	initial element to copy Important: the Source and Destination operands should be the same data type, or unexpected results may occur
Destination	SINT INT DINT REAL string structure	tag	initial element to be overwritten by the Source Important: the Source and Destination operands should be the same data type, or unexpected results may occur
Length	DINT	immediate tag	number of Destination elements to copy



COP (Source, Dest, Length) ;
CPS (Source, Dest, Length) ;

Structured Text

The operands are the same as those for the relay ladder COP and CPS instructions.

Description: During execution of the COP and CPS instructions, other controller actions may try to interrupt the copy operation and change the source or destination data:

If the source or destination is:	And you want to:	Then select:	Notes:
<ul style="list-style-type: none"> produced tag consumed tag I/O data data that another task can overwrite 	prevent the data from changing during the copy operation	CPS	<ul style="list-style-type: none"> Tasks that attempt to interrupt a CPS instruction are delayed until the instruction is done. To estimate the execution time of the CPS instruction, see <i>ControlLogix System User Manual</i>, publication 1756-UM001.
	allow the data to change during the copy operation	COP	
none of the above	→	COP	

The number of bytes copied is:

Byte Count = Length * (number of bytes in the Destination data type)

ATTENTION

If the byte count is greater than the length of the Source, unpredictable data is copied for the remaining elements.

The COP and CPS instructions operate on contiguous data memory and perform a straight byte-to-byte memory copy, which requires an understanding of the controller's memory layout.

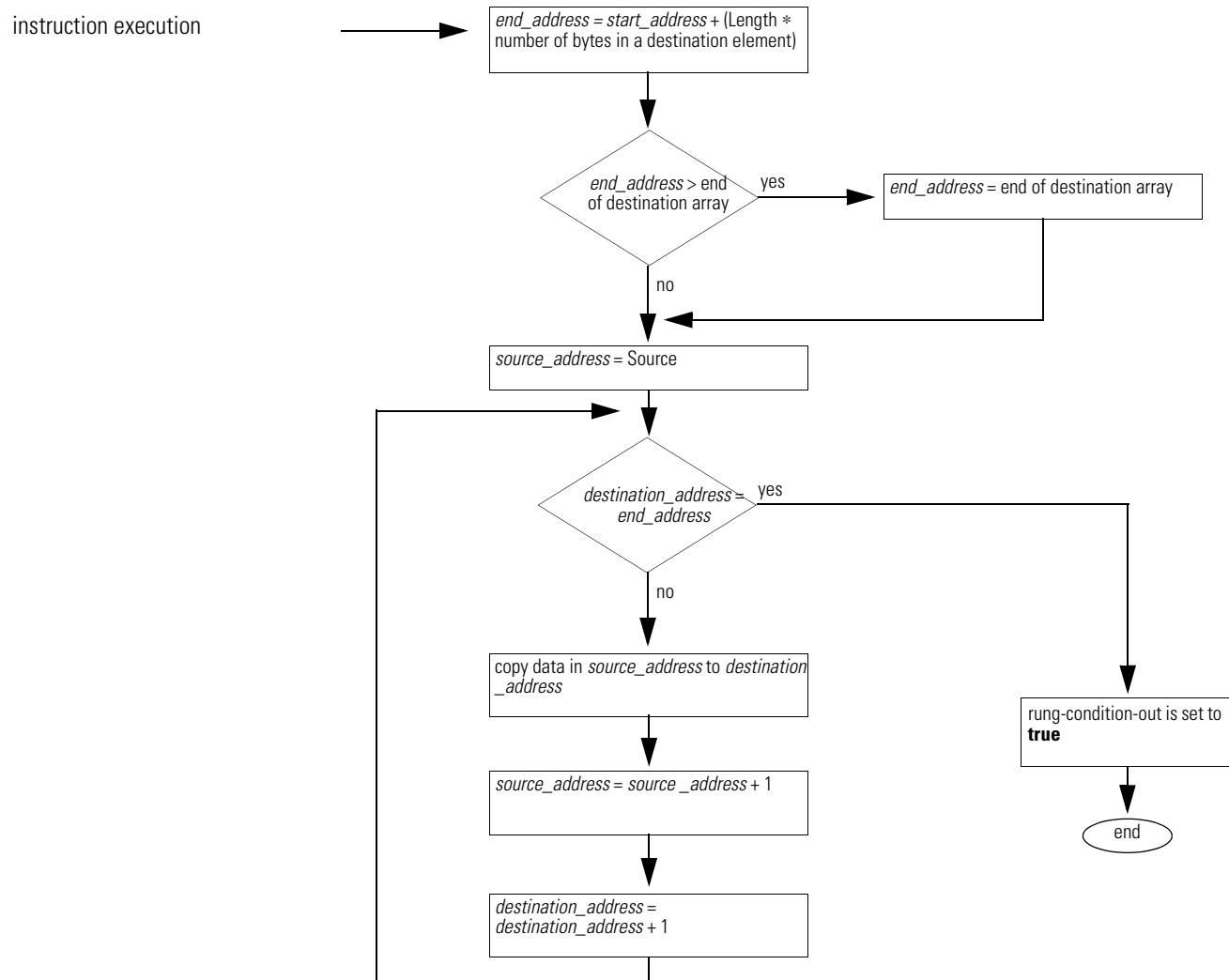
The COP and CPS instructions do not write past the end of the array. If the Length is greater than the total number of elements in the Destination array, the COP and CPS instructions stop at the end of the array. No major fault is generated.

Arithmetic Status Flags: not affected

Fault Conditions: none

Execution:

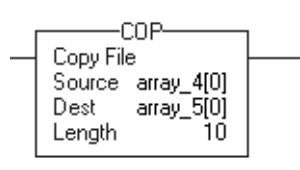
Condition:	Relay Ladder Action:	Structured Text Action:
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	na
EnableIn is set	na	EnableIn is always set. The instruction executes.



postscan	The rung-condition-out is set to false.	No action taken.
----------	---	------------------

Example 1: Both *array_4* and *array_5* are the same data type. When enabled, the COP instruction copies the first 10 elements of *array_4* into the first 10 elements of *array_5*.

Relay Ladder

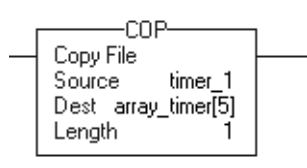


Structured Text

```
COP(array_4[0], array_5[0], 10);
```

Example 2: When enabled, the COP instruction copies the structure *timer_1* into element 5 of *array_timer*. The instruction copies only one structure to one array element.

Relay Ladder



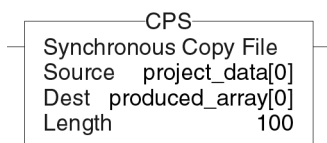
Structured Text

```
COP(timer_1, array_timer[5], 1);
```

Example 3: The *project_data* array (100 elements) stores a variety of values that change at different times in the application. To send a complete image of *project_data* at one instance in time to another controller, the CPS instruction copies *project_data* to *produced_array*.

- While the CPS instruction copies the data, no I/O updates or other tasks can change the data.
- The *produced_array* tag produces the data on a ControlNet network for consumption by other controllers.
- To use the same image of data (i.e., a synchronized copy of the data), the consuming controller (s) uses a CPS instruction to copy the data from the consumed tag to another tag for use in the application.

Relay Ladder



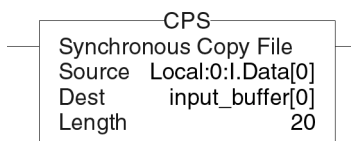
Structured Text

```
CPS (project_data [0] , produced_array [0] , 100) ;
```

Example 4: *Local:0:I.Data* stores the input data for the DeviceNet network that is connected to the 1756-DNB module in slot 0. To synchronize the inputs with the application, the CPS instruction copies the input data to *input_buffer*.

- While the CPS instruction copies the data, no I/O updates can change the data.
- As the application executes, it uses for its inputs the input data in *input_buffer*.

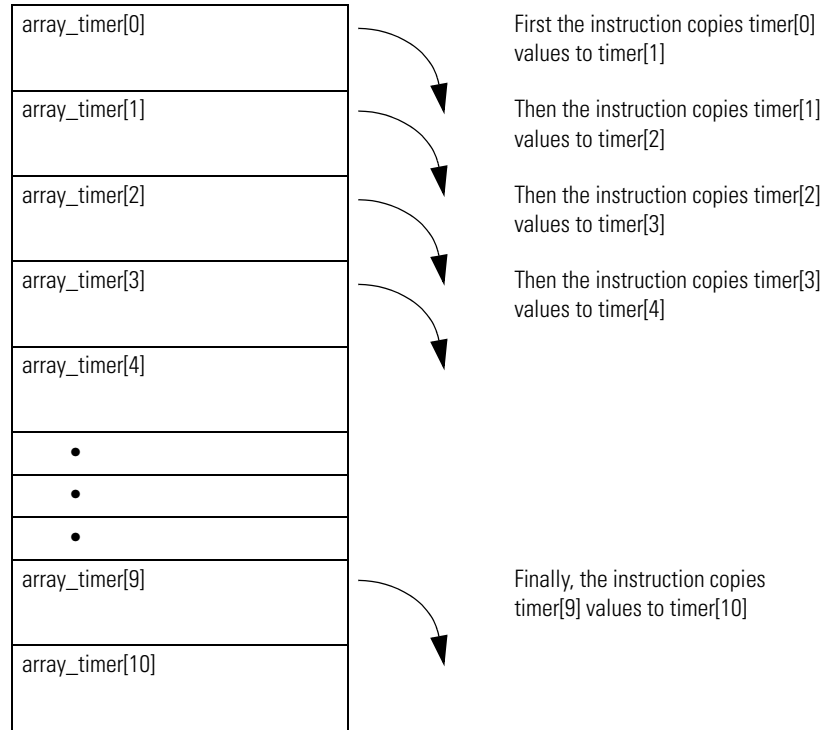
Relay Ladder



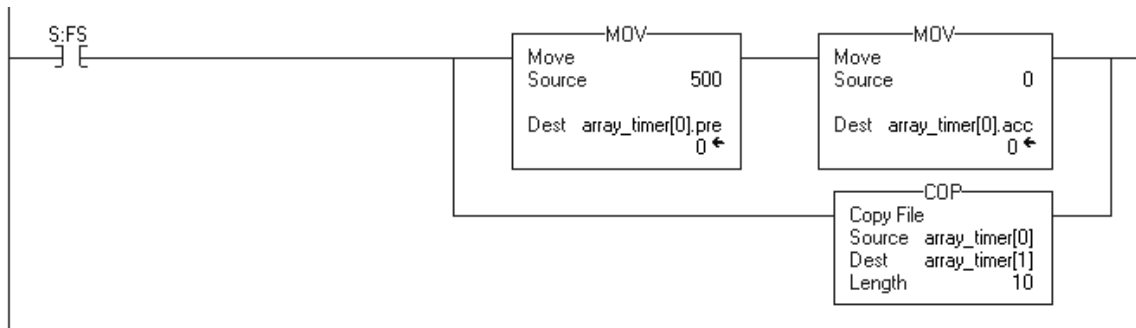
Structured Text

```
CPS (Local:0:I.Data [0] , input_buffer [0] , 20) ;
```

Example 5: This example initializes an array of timer structures. When enabled, the MOV instructions initialize the .PRE and .ACC values of the first *array_timer* element. When enabled, the COP instruction copies a contiguous block of bytes, starting at *array_timer[0]*. The length is nine timer structures.



Relay Ladder



Structured Text

```
IF S:FS THEN

    array_timer[0].pre := 500;

    array_timer[0].acc := 0;

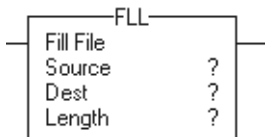
    COP(array_timer[0], array_timer[1], 10);

END_IF;
```

File Fill (FLL)

The FLL instruction fills elements of an array with the Source value. The Source remains unchanged.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Source	SINT INT DINT REAL	immediate tag	element to copy Important: the Source and Destination operands should be the same data type, or unexpected results may occur
Destination	SINT INT DINT REAL structure	tag	initial element to be overwritten by the Source Important: the Source and Destination operands should be the same data type, or unexpected results may occur The preferred way to initialize a structure is to use the COP instruction.
Length	DINT	immediate	number of elements to fill



Structured Text

Structured text does not have an FLL instruction, but you can achieve the same results using a SIZE instruction and a FOR...DO or other loop construct.

```
SIZE(destination,0,length);
FOR position = 0 TO length-1 DO
    destination[position] := source;
END_FOR;
```

See Appendix C for information on the syntax of constructs within structured text.

Description: The number of bytes filled is:

Byte count = Length * (number of bytes in the Destination data type)

The FLL instruction operates on contiguous data memory.

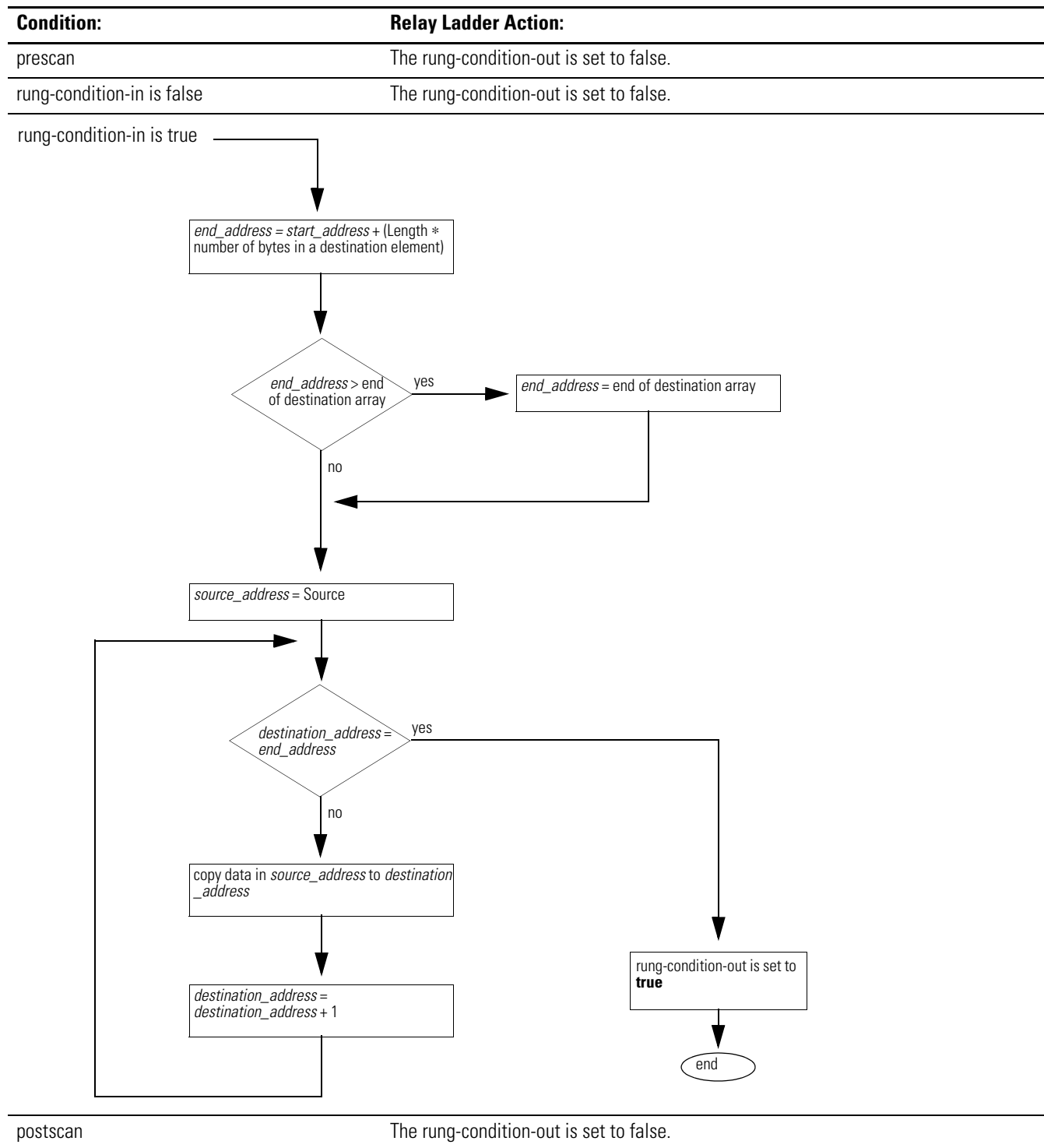
The FLL instruction will not write past the end of an array. If the Length is greater than the total number of elements in the Destination array, the FLL instruction stops at the end of the array. No major fault is generated.

For best results, the Source and Destination should be the same type. If you want to fill a structure, use the COP instruction (see example 3 on page 7-32). If you mix data types for the Source and Destination, the Destination elements are filled with converted Source values.

If the Source is:	And the Destination is:	The Source is converted to:
SINT, INT, DINT, or REAL	SINT	SINT
SINT, INT, DINT, or REAL	INT	INT
SINT, INT, DINT, or REAL	DINT	DINT
SINT, INT, DINT, or REAL	REAL	REAL
SINT	structure	SINT (not converted)
INT	structure	INT (not converted)
DINT	structure	DINT (not converted)
REAL	structure	REAL (not converted)

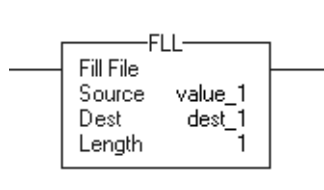
Arithmetic Status Flags: not affected

Fault Conditions: none

Execution:

Example: The FLL instruction copies the value in *value_1* into *dest_1*

Relay Ladder



Source (<i>value_1</i>) data type:	Source (<i>value_1</i>) value:	Destination (<i>dest_1</i>) data type:	Destination (<i>dest_1</i>) value after FLL:
SINT	16#80 (-128)	DINT	16#FFFF FF80 (-128)
DINT	16#1234 5678	SINT	16#78
SINT	16#01	REAL	1.0
REAL	2.0	INT	16#0002
SINT	16#01	TIMER	16#0101 0101 16#0101 0101 16#0101 0101
INT	16#0001	TIMER	16#0001 0001 16#0001 0001 16#0001 0001
DINT	16#0000 0001	TIMER	16#0000 0001 16#0000 0001 16#0000 0001

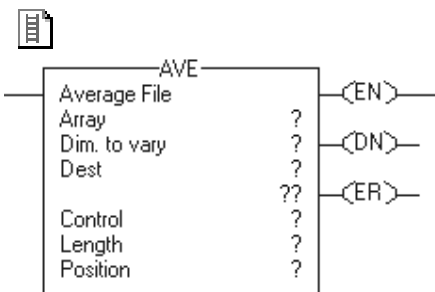
Structured Text

```
dest_1 := value_1;
```

File Average (AVE)

The AVE instruction calculates the average of a set of values.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Array	SINT INT DINT REAL	array tag	find the average of the values in this array specify the first element of the group of elements to average do not use CONTROL.POS in the subscript
Dimension to vary	DINT	immediate (0, 1, 2)	which dimension to use depending on the number of dimensions, the order is array[dim_0,dim_1,dim_2] array[dim_0,dim_1] array[dim_0]
Destination	SINT INT DINT REAL	tag	result of the operation
Control	CONTROL	tag	control structure for the operation
Length	DINT	immediate	number of elements of the array to average
Position	DINT	immediate	current element in the array initial value is typically 0



Structured Text

Structured text does not have an AVE instruction, but you can achieve the same results using a SIZE instruction and a FOR...DO or other loop construct.

```
SIZE(array,0,length);  
sum := 0;  
FOR position = 0 TO length-1 DO  
    sum := sum + array[position];  
END_FOR;  
destination := sum / length;
```

See Appendix C for information on the syntax of constructs within structured text.

CONTROL Structure

Mnemonic:	Data Type:	Description:
.EN	BOOL	The enable bit indicates that the AVE instruction is enabled.
.DN	BOOL	The done bit is set when the instruction has operated on the last element in the Array (.POS = .LEN).
.ER	BOOL	The error bit is set if the instruction generates an overflow. The instruction stops executing until the program clears the .ER bit. The position of the element that caused the overflow is stored in the .POS value.
.LEN	DINT	The length specifies the number of elements in the array on which the instruction operates.
.POS	DINT	The position contains the position of the current element that the instruction is accessing.

Description: The AVE instruction calculates the average of a set of values.

IMPORTANT

Make sure the Length does not cause the instruction to exceed the specified Dimension to vary. If this happens, the Destination will be incorrect.

Arithmetic Status Flags: Arithmetic status flags are affected.

Fault Conditions:

A major fault will occur if:	Fault type:	Fault code:
.POS < 0 or .LEN < 0	4	21
Dimension to vary does not exist for the specified array	4	20

Execution:

Condition:	Relay Ladder Action:
prescan	The .EN bit is cleared. The .DN bit is cleared. The .ER bit is cleared. The rung-condition-out is set to false.
rung-condition-in is false <div data-bbox="337 531 781 1308" data-label="Diagram"> <pre> graph TD Start(()) --> Decision{examine .DN bit} Decision -- ".DN bit = 0" --> Decision Decision -- ".DN bit = 1" --> Action1[.EN bit is cleared .ER bit is cleared .DN bit is cleared .POS value is cleared] Action1 --> Action2[rung-condition-out is set to false] Action2 --> End((end)) </pre> </div>	
rung-condition-in is true	The AVE instruction calculates the average by adding all the specified elements in the array and dividing by the number of elements. Internally, the instruction uses a FAL instruction to calculate the average: Expression = average calculation Mode = ALL For details on how the FAL instruction executes, see page 7-9.
postscan	The rung-condition-out is set to false.

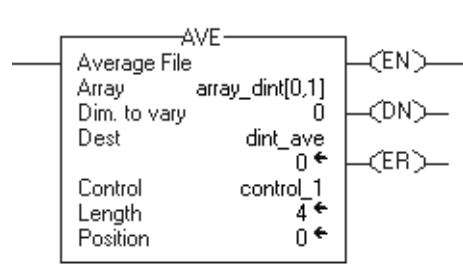
Example 1: Average *array_dint*, which is DINT[4,5].

dimension 0	subscripts	dimension 1				
		0	1	2	3	4
	0	20	19	18	17	16
	1	15	14	13	12	11
	2	10	9	8	7	6
	3	5	4	3	2	1

$$AVE = \frac{19 + 14 + 9 + 4}{4} = \frac{46}{4} = 11.5$$

$$dint_ave = 12$$

Relay Ladder



Structured Text

```

SIZE(array_dint,0,length);
sum := 0;
FOR position = 0 TO (length-1) DO
    sum := sum + array_dint[position];
END_FOR;
dint_ave := sum / length;

```

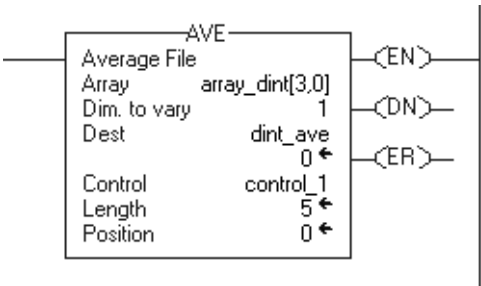
Example 2: Average *array_dint*, which is DINT[4,5].

dimension 0	subscripts	dimension 1				
		0	1	2	3	4
	0	20	19	18	17	16
	1	15	14	13	12	11
	2	10	9	8	7	6
	3	5	4	3	2	1

$$AVE = \frac{5 + 4 + 3 + 2 + 1}{5} = \frac{15}{5} = 3$$

$$dint_ave = 3$$

Relay Ladder



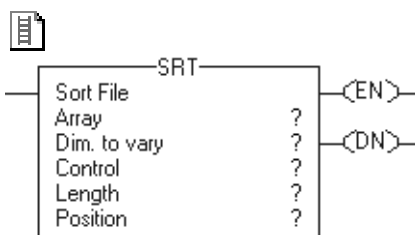
Structured Text

```
SIZE(array_dint,1,length);
sum := 0;
FOR position = 0 TO (length-1) DO
    sum := sum + array_dint[position];
END_FOR;
dint_ave := sum / length;
```


File Sort (SRT)

The SRT instruction sorts a set of values in one dimension (Dim to vary) of the Array into ascending order.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Array	SINT INT DINT REAL	array tag	array to sort specify the first element of the group of elements to sort do not use CONTROL.POS in the subscript
Dimension to vary	DINT	immediate (0, 1, 2)	which dimension to use depending on the number of dimensions, the order is array[dim_0,dim_1,dim_2] array[dim_0,dim_1] array[dim_0]
Control	CONTROL	tag	control structure for the operation
Length	DINT	immediate	number of elements of the array to sort
Position	DINT	immediate	current element in the array initial value is typically 0



```
SRT (Array,Dimtovary,  
Control);
```

Structured Text

The operands are the same as those for the relay ladder SRT instruction. However, you specify the Length and Position values by accessing the .LEN and .POS members of the CONTROL structure, rather than by including values in the operand list.

CONTROL Structure

Mnemonic:	Data Type:	Description:
.EN	BOOL	The enable bit indicates that the SRT instruction is enabled.
.DN	BOOL	The done bit is set when the specified elements have been sorted.
.ER	BOOL	The error bit is set when either .LEN < 0 or .POS < 0. Either of these conditions also generates a major fault.
.LEN	DINT	The length specifies the number of elements in the array on which the instruction
.POS	DINT	The position contains the position of the current element that the instruction is accessing.

Description: The SRT instruction sorts a set of values in one dimension (Dim to vary) of the Array into ascending order.

IMPORTANT

Make sure the Length does not cause the instruction to exceed the specified Dimension to vary. If this happens, unexpected results will occur.

This is a transitional instruction:

- In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.
- In structured text, condition the instruction so that it only executes on a transition. See Appendix C.

Arithmetic Status Flags: Arithmetic status flags are affected.

Fault Conditions:

A major fault will occur if:	Fault type:	Fault code:
.POS < 0 or .LEN < 0	4	21
Dimension to vary does not exist for the specified array	4	20
Instruction tries to access data outside of the array boundaries	4	20

Execution:

Condition:	Relay Ladder Action:	Structured Text Action:
prescan	The .EN bit is cleared. The .DN bit is cleared. The .ER bit is cleared. The rung-condition-out is set to false.	The .EN bit is cleared. The .DN bit is cleared. The .ER bit is cleared.
rung-condition-in is false		na
<pre> graph TD A{examine .DN bit} -- ".DN bit = 0" --> C[rung-condition-out is set to false] A -- ".DN bit = 1" --> B[.EN bit is cleared .ER bit is cleared .DN bit is cleared .POS value is cleared] B --> C C --> D([end]) </pre>		
rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	na
EnableIn is set	na	EnableIn is always set. The instruction executes.
instruction execution	The instruction sorts the specified elements of the array into ascending order.	The instruction sorts the specified elements of the array into ascending order.
postscan	The rung-condition-out is set to false.	No action taken.

Example 1: Sort *int _array*, which is DINT[4,5].

Before

subscripts

dimension 1

01234

02019181716

11514131211

2109876

354321

dimension 0

After

subscripts

dimension 1

01234

0201931716

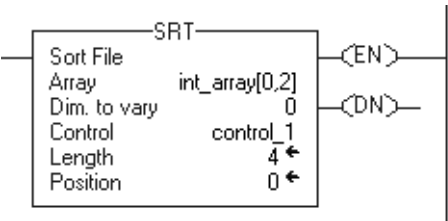
1151481211

21091376

3541821

dimension 0

Relay Ladder



Structured Text

```
control_1.LEN := 4;

control_1.POS := 0;

SRT(int_array[0,2], 0, control_1);
```

Example 2: Sort *int _array*, which is DINT[4,5].

Before

subscripts

dimension 1

dimension 0

	0	1	2	3	4
0	20	19	18	17	16
1	15	14	13	12	11
2	10	9	8	7	6
3	5	4	3	2	1

After

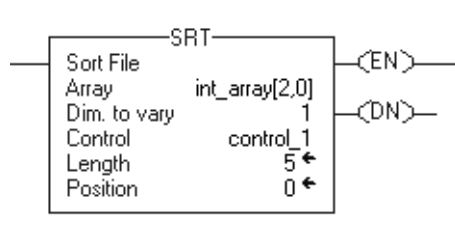
subscripts

dimension 1

dimension 0

	0	1	2	3	4
0	20	19	18	17	16
1	15	14	13	12	11
2	6	7	8	9	10
3	5	4	3	2	1

Relay Ladder



Structured Text

```

control_1.LEN := 5;

control_1.POS := 0;

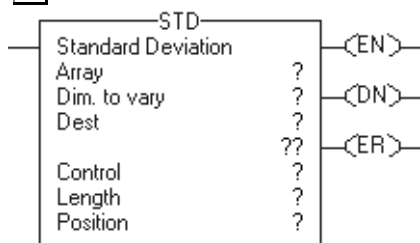
SRT(int_array[2,0],1,control_1);

```

File Standard Deviation (STD)

The STD instruction calculates the standard deviation of a set of values in one dimension of the Array and stores the result in the Destination.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Array	SINT INT DINT REAL	array tag	find the standard deviation of the values in this array specify the first element of the group of elements to use in calculating the standard deviation do not use CONTROL.POS in the subscript A SINT or INT tag converts to a DINT value by sign-extension.
Dimension to vary	DINT	immediate (0, 1, 2)	which dimension to use depending on the number of dimensions, the order is array[dim_0,dim_1,dim_2] array[dim_0,dim_1] array[dim_0]
Destination	REAL	tag	result of the operation
Control	CONTROL	tag	control structure for the operation
Length	DINT	immediate	number of elements of the array to use in calculating the standard deviation
Position	DINT	immediate	current element in the array initial value is typically 0

CONTROL Structure

Mnemonic:	Data Type:	Description:
.EN	BOOL	The enable bit indicates that the STD instruction is enabled.
.DN	BOOL	The done bit is set when the calculation is complete.
.ER	BOOL	The error bit is set when the instruction generates an overflow. The instruction stops executing until the program clears the .ER bit. The position of the element that caused the overflow is stored in the .POS value.
.LEN	DINT	The length specifies the number of elements in the array on which the instruction operates.
.POS	DINT	The position contains the position of the current element that the instruction is accessing.



Structured Text

Structured text does not have an STD instruction, but you can achieve the same results using a SIZE instruction and a FOR...DO or other loop construct.

```
SIZE(array,0,length);
sum := 0;
FOR position = 0 TO length-1 DO
    sum := sum + array[position];
END_FOR;
average := sum / length;
sum := 0;
FOR position = 0 TO length-1 DO
    sum := sum + ((array[position] - average)**2);
END_FOR;
destination := SQRT(sum / (length-1));
```

See Appendix C for information on the syntax of constructs within structured text.

Description: The standard deviation is calculated according to this formula:

$$\text{Standard Deviation} = \sqrt{\frac{\sum_{i=1}^N [X_{(start+i)} - AVE]^2}{(N-1)}}$$

Where:

- start = dimension-to-vary subscript of the array operand
- x_i = variable element in the array
- N = number of specified elements in the array

$$\bullet \text{ AVE} = \frac{\sum_{i=1}^N x_{(start+i)}}{N}$$

IMPORTANT

Make sure the Length does not cause the instruction to exceed the specified Dimension to vary. If this happens, the Destination will be incorrect.

Arithmetic Status Flags: Arithmetic status flags are affected.

Fault Conditions:

A major fault will occur if:	Fault type:	Fault code:
.POS < 0 or .LEN < 0	4	21
Dimension to vary does not exist for the specified array	4	20

Execution:

Condition:	Relay Ladder Action:
prescan	The .EN bit is cleared. The .DN bit is cleared. The .ER bit is cleared. The rung-condition-out is set to false.
<p>rung-condition-in is false</p> <pre>graph TD Start([rung-condition-in is false]) --> Decision{examine .DN bit} Decision -- ".DN bit = 0" --> SetFalse[rung-condition-out is set to false] Decision -- ".DN bit = 1" --> ClearBits[.EN bit is cleared .ER bit is cleared .DN bit is cleared .POS value is cleared] ClearBits --> SetFalse SetFalse --> End([end])</pre>	
rung-condition-in is true	The STD instruction calculates the standard deviation of the specified elements. Internally, the instruction uses a FAL instruction to calculate the average: Expression = standard deviation calculation Mode = ALL For details on how the FAL instruction executes, see page 7-9.
postscan	The rung-condition-out is set to false.

Example 1: Calculate the standard deviation of *dint_array*, which is DINT[4,5].

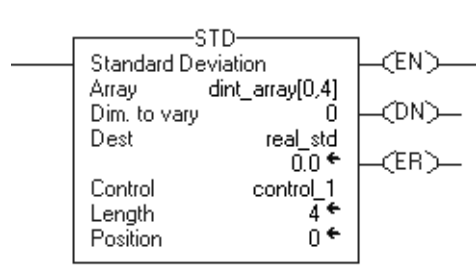
$$AVE = \frac{16 + 11 + 6 + 1}{4} = \frac{34}{4} = 8.5$$

$$STD = \sqrt{\frac{\langle 16 - 8.5 \rangle^2 + \langle 11 - 8.5 \rangle^2 + \langle 6 - 8.5 \rangle^2 + \langle 1 - 8.5 \rangle^2}{\langle 4 - 1 \rangle}} = 6.454972$$

$$real_std = 6.454972$$

		dimension 1				
		0	1	2	3	4
dimension 0	subscripts	0	1	2	3	4
	0	20	19	18	17	16
	1	15	14	13	12	11
	2	10	9	8	7	6
	3	5	4	3	2	1

Relay Ladder



Structured Text

```

SIZE(dint_array,0,length);
sum := 0;
FOR position = 0 TO (length-1) DO
    sum := sum + dint_array[position];
END_FOR;
average := sum / length;
sum := 0;
FOR position = 0 TO (length-1) DO
    sum := sum + ((dint_array[position] - average)**2);
END_FOR;
real_std := SQRT(sum / (length-1));

```

Example 2: Calculate the standard deviation of `dint_array`, which is DINT[4,5].

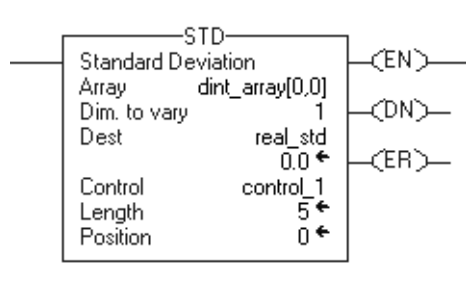
$$AVE = \frac{20 + 19 + 18 + 17 + 16}{5} = \frac{90}{5} = 18$$

$$STD = \sqrt{\frac{\langle 20 - 18 \rangle^2 + \langle 19 - 18 \rangle^2 + \langle 18 - 18 \rangle^2 + \langle 17 - 18 \rangle^2 + \langle 16 - 18 \rangle^2}{\langle 5 - 1 \rangle}} = 1.581139$$

$$real_std = 1.581139$$

		dimension 1				
		0	1	2	3	4
dimension 0	0	20	19	18	17	16
	1	15	14	13	12	11
	2	10	9	8	7	6
	3	5	4	3	2	1

Relay Ladder



Structured Text

```

SIZE(dint_array,1,length);
sum := 0;
FOR position = 0 TO (length-1) DO
    sum := sum + dint_array[position];
END_FOR;
average := sum / length;
sum := 0;
FOR position = 0 TO (length-1) DO
    sum := sum + ((dint_array[position] - average)**2);
END_FOR;
real_std := SQRT(sum / (length-1));
  
```

Size In Elements (SIZE)

The SIZE instruction finds the size of a dimension of an array.

Operands:



SIZE	
Size in Elements	?
Source	??
Dim. To Vary	?
Size	?
	??

Relay Ladder

Operand:	Type:	Format:	Description:								
Source	SINT INT DINT REAL structure string	array tag	array on which the instruction is to operate								
Dimension to Vary	DINT	immediate (0, 1, 2)	dimension to use: <table><tr><th>For the size of:</th><th>Enter:</th></tr><tr><td>first dimension</td><td>0</td></tr><tr><td>second dimension</td><td>1</td></tr><tr><td>third dimension</td><td>2</td></tr></table>	For the size of:	Enter:	first dimension	0	second dimension	1	third dimension	2
For the size of:	Enter:										
first dimension	0										
second dimension	1										
third dimension	2										
Size	SINT INT DINT REAL	tag	tag to store the number of elements in the specified dimension of the array								



SIZE(Source,Dimtovary,Size);

Structured Text

The operands are the same as those for the relay ladder SIZE instruction.

Description: The SIZE instruction finds the number of elements (size) in the designated dimension of the Source array and places the result in the Size operand.

- The instruction finds the size of one dimension of an array.
- The instruction operates on an:
 - array
 - array in a structure
 - array that is part of a larger array

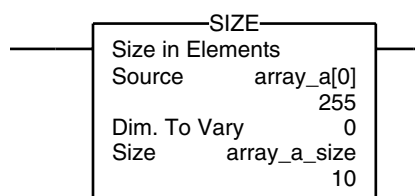
Arithmetic Status Flags: not affected

Fault Conditions: none.

Execution:

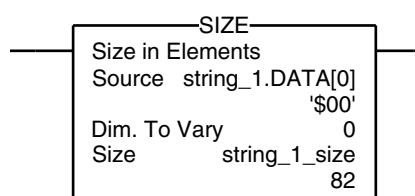
Condition:	Relay Ladder Action:	Structured Text Action:
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	na
EnableIn is set	na	EnableIn is always set. The instruction executes.
instruction execution	The instruction finds the size of a dimension.	The instruction finds the size of a dimension.
postscan	The rung-condition-out is set to false.	No action taken.

Example 1: Find the number of elements in dimension 0 (first dimension) of *array_a*. Store the size in *array_a_size*. In this example, dimension 0 of *array_a* has 10 elements.

Relay Ladder**Structured Text**

```
SIZE(array_a,0,array_a_size);
```

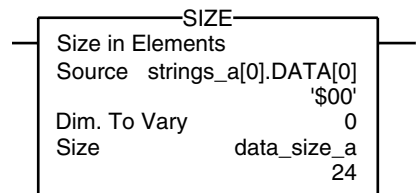
Example 2: Find the number of elements in the DATA member of *string_1*, which is a string. Store the size in *string_1_size*. In this example, the DATA member of *string_1* has 82 elements. (The string uses the default STRING data type.) Since each element holds one character, *string_1* can contain up to 82 characters.

Relay Ladder**Structured Text**

```
SIZE(string_1.DATA[0],0,string_1_size);
```

Example 3: *Strings_a* is an array of string structures. The SIZE instruction finds the number of elements in the DATA member of the string structure and stores the size in *data_size_a*. In this example, the DATA member has 24 elements. (The string structure has a user-specified length of 24.)

Relay Ladder



Structured Text

```
SIZE(strings_a[0].DATA[0], 0, data_size_a);
```

Notes:

Array (File)/Shift Instructions

(BSL, BSR, FFL, FFU, LFL, LFU)

Introduction

Use the array (file)/shift instructions to modify the location of data within arrays.

If you want to:	Use this instruction:	Available in these languages:	See page:
Load bits into, shift bits through, and unload bits from a bit array one bit at a time.	BSL	relay ladder	8-2
	BSR	relay ladder	8-5
Load and unload values in the same order.	FFL	relay ladder	8-8
	FFU	relay ladder	8-14
Load and unload values in reverse order.	LFL	relay ladder	8-20
	LFU	relay ladder	8-26

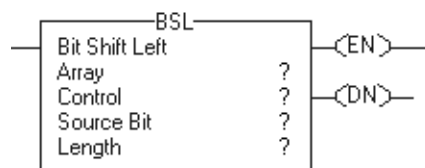
You can mix data types, but loss of accuracy and rounding error might occur.

For relay ladder instructions, **bold** data types indicate optimal data types. An instruction executes faster and requires less memory if all the operands of the instruction use the same optimal data type, typically DINT or REAL.

Bit Shift Left (BSL)

The BSL instruction shifts the specified bits within the Array one position left.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Array	DINT	array tag	array to modify specify the first element of the group of elements do not use CONTROL.POS in the subscript
Control	CONTROL	tag	control structure for the operation
Source bit	BOOL	tag	bit to shift
Length	DINT	immediate	number of bits in the array to shift

CONTROL Structure

Mnemonic:	Data Type:	Description:
.EN	BOOL	The enable bit indicates that the BSL instruction is enabled.
.DN	BOOL	The done bit is set to indicate that bits shifted one position to the left.
.UL	BOOL	The unload bit is the instruction's output. The .UL bit stores the status of the bit that was shifted out of the range of bits.
.ER	BOOL	The error bit is set when .LEN < 0.
.LEN	DINT	The length specifies the number of array bits to shift.

Description: When enabled, the instruction unloads the uppermost bit of the specified bits to the .UL bit, shifts the remaining bits one position left, and loads Source bit into bit 0 of Array.

The BSL instruction operates on contiguous memory.

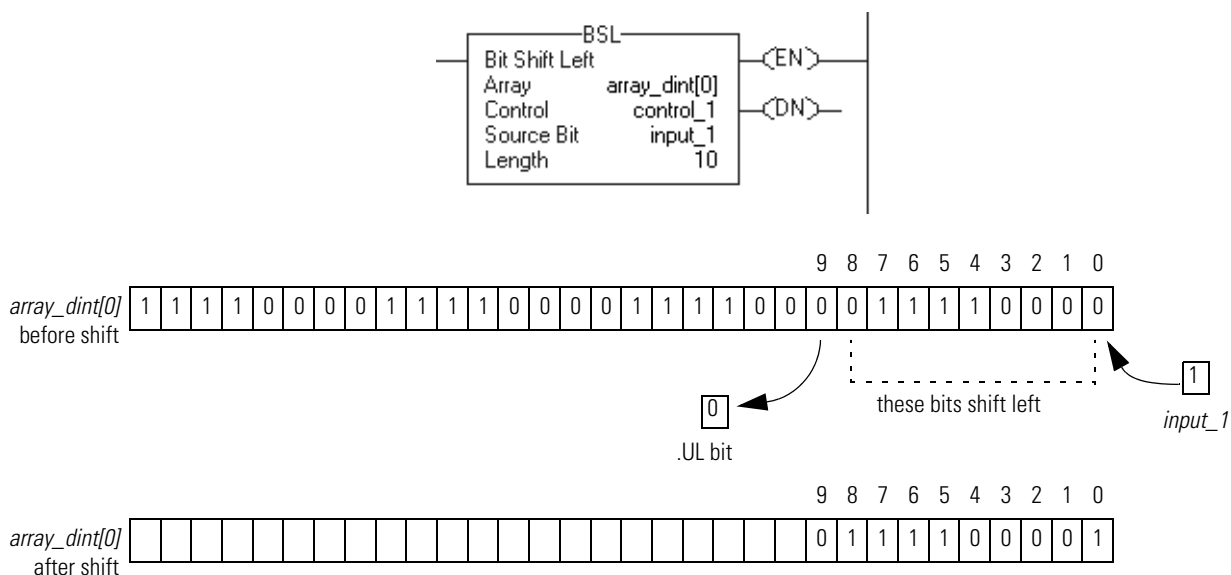
Arithmetic Status Flags: not affected

Fault Conditions: none

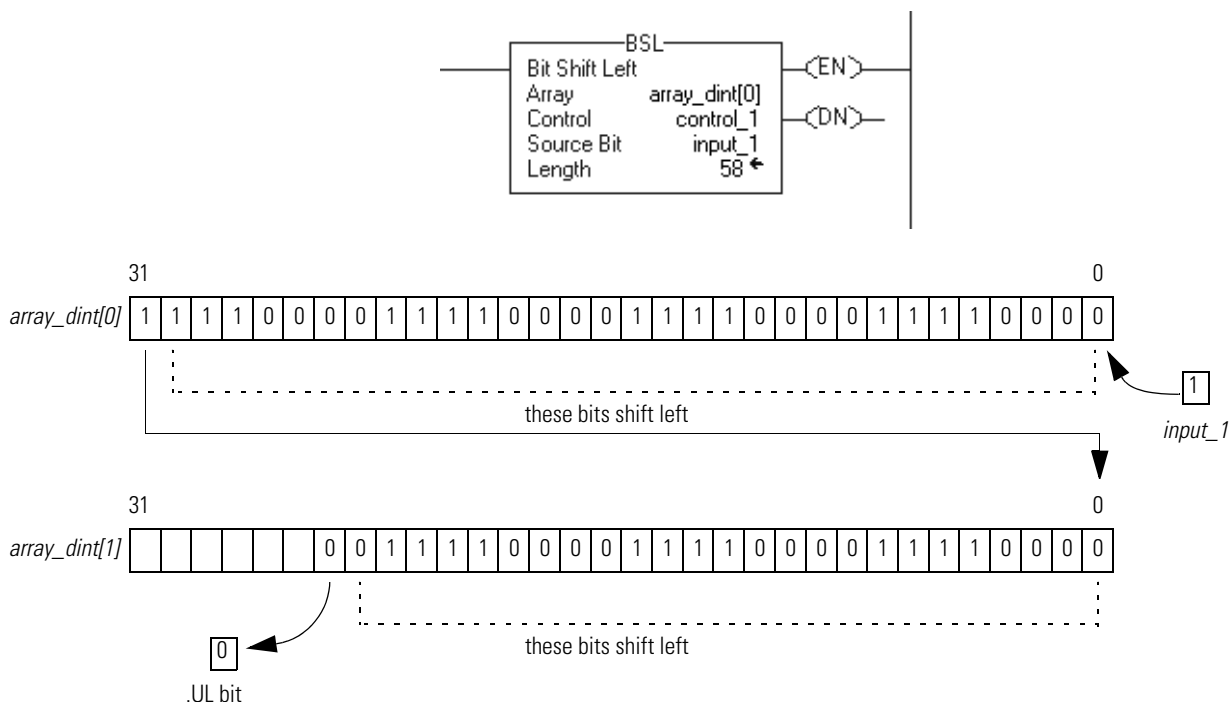
Execution:

Condition:	Relay Ladder Action:
prescan	The .EN bit is cleared. The .DN bit is cleared. The .ER bit is cleared. The .POS value is cleared. The rung-condition-out is set to false.
rung-condition-in is false	The .EN bit is cleared. The .DN bit is cleared. The .ER bit is cleared. The .POS value is cleared. The rung-condition-out is set to false.
<pre> graph TD Start([rung-condition-in is true]) --> EN_1{examine .EN bit} EN_1 -- ".EN bit = 1" --> DN_Set[.DN bit is set] EN_1 -- ".EN bit = 0" --> EN_Set[.EN bit is set] EN_Set --> LEN_0{.LEN = 0} LEN_0 -- yes --> DN_Set LEN_0 -- no --> LEN_LT_0{.LEN < 0} LEN_LT_0 -- yes --> ER_Set[.ER bit is set] LEN_LT_0 -- no --> Shift[shift array left one position left] subgraph Shift_Box [shift array left one position left] direction LR UL[.UL bit] <--> array[array] <--> source[source bit] end Shift_Box --> DN_POS[.DN bit is set .POS = .LEN] DN_Set --> Source_1{examine source bit} ER_Set --> Source_1 Source_1 -- ".source bit = 1" --> UL_Remains[.UL bit remains set] Source_1 -- ".source bit = 0" --> UL_Set[.UL bit is set] UL_Remains --> Rung_Out_Set[rung-condition-out is set to true] UL_Set --> Rung_Out_Set DN_POS --> Rung_Out_Set Rung_Out_Set --> End([end]) </pre> <p>The flowchart illustrates the execution logic for the Array Shift Instruction. It begins with a condition 'rung-condition-in is true'. A decision diamond 'examine .EN bit' checks if the .EN bit is 1. If yes, it sets the .DN bit and proceeds to 'examine source bit'. If no, it sets the .EN bit and checks if .LEN is 0. If .LEN is 0, it sets the .DN bit. If not, it checks if .LEN is less than 0. If yes, it sets the .ER bit. If no, it performs a 'shift array left one position left' operation, which involves shifting the .UL bit, the array, and the source bit. After the shift, it sets the .DN bit and .POS to .LEN. Both the .DN bit set path and the .ER bit set path lead to 'examine source bit'. If the source bit is 1, the .UL bit remains set. If the source bit is 0, the .UL bit is set. Both paths lead to 'rung-condition-out is set to true', which then leads to 'end'.</p>	
postscan	The rung-condition-out is set to false.

Example 1: When enabled, the BSL instruction starts at bit 0 in *array_dint[0]*. The instruction unloads *array_dint[0].9* into the .UL bit, shifts the remaining bits, and loads *input_1* into *array_dint[0].0*. The values in the remaining bits (10-31) are invalid.



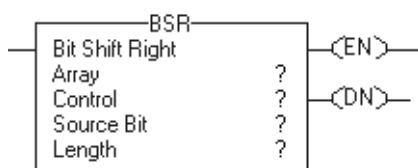
Example 2: When enabled, the BSL instruction starts at bit 0 in *array_dint[0]*. The instruction unloads *array_dint[1].25* into the .UL bit, shifts the remaining bits, and loads *input_1* into *array_dint[0].0*. The values in the remaining bits (31-26 in *array_dint[1]*) are invalid. Note how *array_dint[0].31* shifts across words to *array_dint[1].0*.



Bit Shift Right (BSR)

The BSR instruction shifts the specified bits within the Array one position right.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Array	DINT	array tag	array to modify specify the element where to begin the shift do not use CONTROL.POS in the subscript
Control	CONTROL	tag	control structure for the operation
Source bit	BOOL	tag	bit to shift
Length	DINT	immediate	number of bits in the array to shift

CONTROL Structure

Mnemonic:	Data Type:	Description:
.EN	BOOL	The enable bit indicates that the BSR instruction is enabled.
.DN	BOOL	The done bit is set to indicate that bits shifted one position to the right.
.UL	BOOL	The unload bit is the instruction's output. The .UL bit stores the status of the bit that was shifted out of the range of bits.
.ER	BOOL	The error bit is set when .LEN < 0.
.LEN	DINT	The length specifies the number of array bits to shift.

Description: When enabled, the instruction unloads the value at bit 0 of Array to the .UL bit, shifts the remaining bits one position right, and loads Source bit into the uppermost bit of the specified bits.

The BSR instruction operates on contiguous memory.

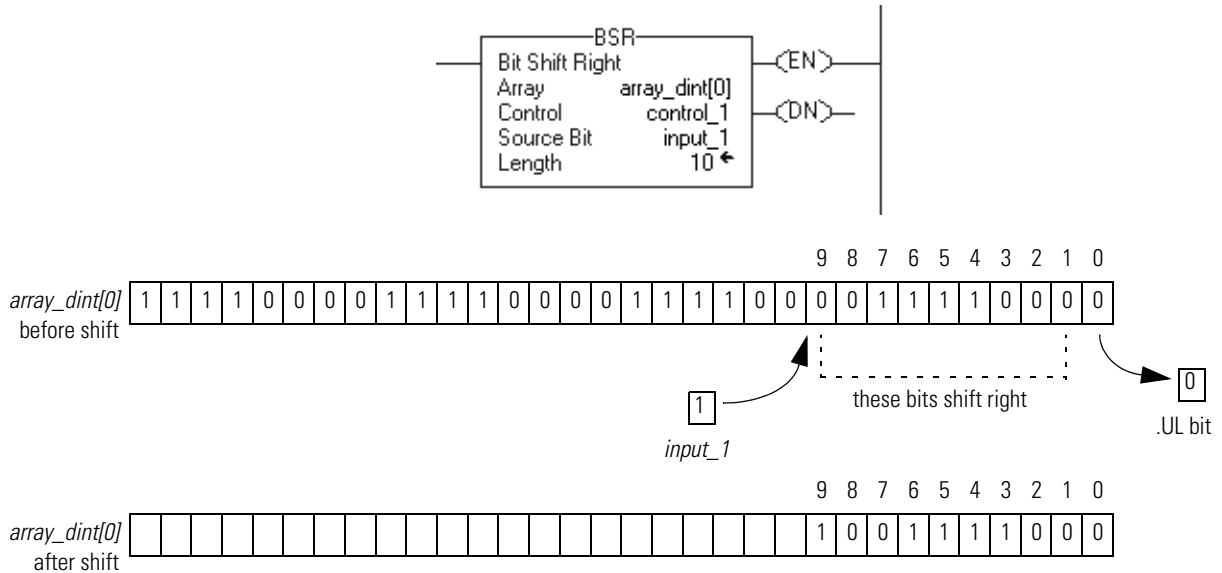
Arithmetic Status Flags: not affected

Fault Conditions: none

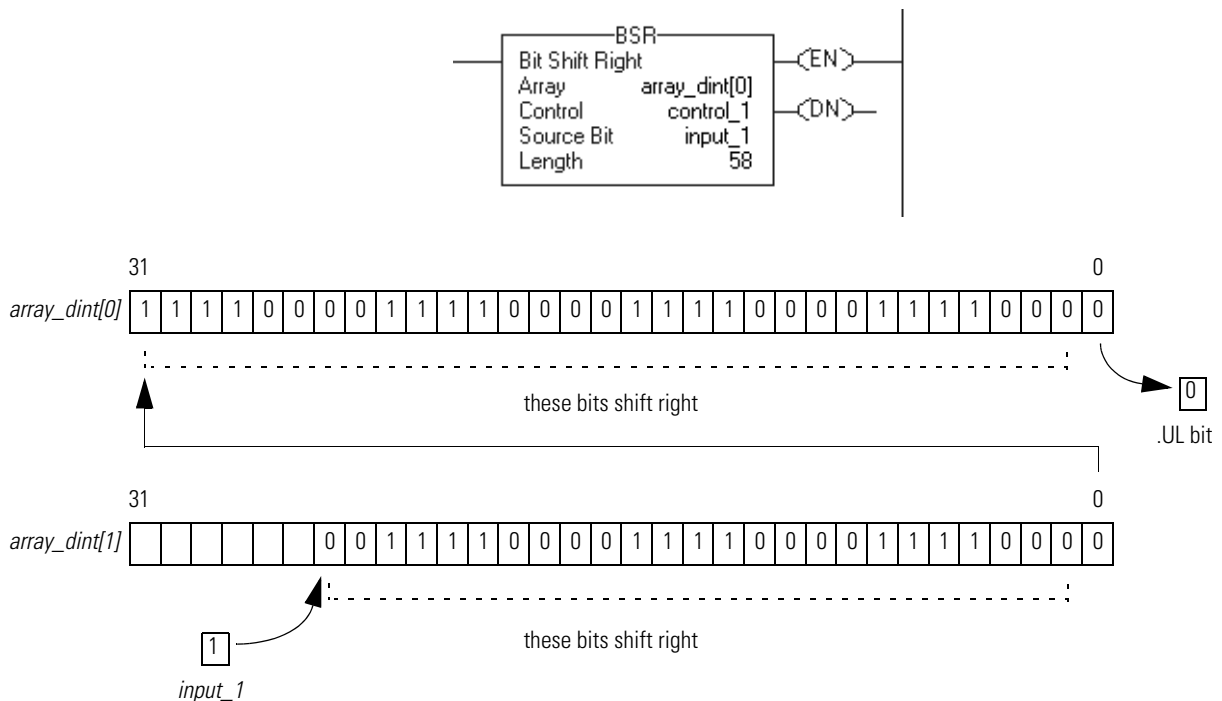
Execution:

Condition:	Relay Ladder Action:
prescan	The .EN bit is cleared. The .DN bit is cleared. The .ER bit is cleared. The .POS value is cleared. The rung-condition-out is set to false.
rung-condition-in is false	The .EN bit is cleared. The .DN bit is cleared. The .ER bit is cleared. The .POS value is cleared. The rung-condition-out is set to false.
rung-condition-in is true	<pre> graph TD Start([rung-condition-in is true]) --> EN_Dec{examine .EN bit} EN_Dec -- ".EN bit = 1" --> DN_Set1[.DN bit is set] EN_Dec -- ".EN bit = 0" --> EN_Set[.EN bit is set] EN_Set --> LEN0_Dec{.LEN = 0} LEN0_Dec -- yes --> DN_Set2[.DN bit is set] LEN0_Dec -- no --> LENLT0_Dec{.LEN < 0} LENLT0_Dec -- yes --> ER_Set[.ER bit is set] LENLT0_Dec -- no --> Shift[shift array left one position left] subgraph ShiftBlock [shift array left one position left] direction LR Source[source bit] --> Array[array] --> UL[.UL bit] end ShiftBlock --> DN_POS[.DN bit is set .POS = .LEN] DN_Set1 --> SourceBit_Dec{examine source bit} DN_Set2 --> SourceBit_Dec SourceBit_Dec -- ".source bit = 1" --> UL_Remains[.UL bit remains set] SourceBit_Dec -- ".source bit = 0" --> UL_Set[.UL bit is set] UL_Remains --> RungOut[run-condition-out is set to true] UL_Set --> RungOut ER_Set --> RungOut DN_POS --> RungOut RungOut --> End([end]) </pre>
postscan	The rung-condition-out is set to false.

Example 1: When enabled, the BSR instruction starts at bit 9 in *array_dint[0]*. The instruction unloads *array_dint[0].0* into the .UL bit, shifts the remaining bits right, and loads *input_1* into *array_dint[0].9*. The values in the remaining bits (10-31) are invalid.



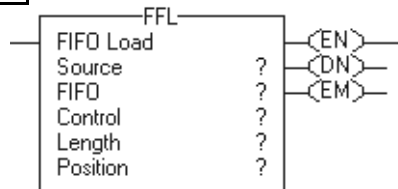
Example 2: When enabled, the BSR instruction starts at bit 25 in *array_dint[1]*. The instruction unloads *array_dint[0].0* into the .UL bit, shifts the remaining bits right, and loads *input_1* into *array_dint[1].25*. The values in the remaining bits (31-26 in *dint_array[1]*) are invalid. Note how *array_dint[1].0* shifts across words into *array_dint[0].31*.



FIFO Load (FFL)

The FFL instruction copies the Source value to the FIFO.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Source	SINT INT DINT REAL string structure	immediate tag	data to be stored in the FIFO The Source converts to the data type of the array tag. A smaller integer converts to a larger integer by sign-extension.
FIFO	SINT INT DINT REAL string structure	array tag	FIFO to modify specify the first element of the FIFO do not use CONTROL.POS in the subscript
Control	CONTROL	tag	control structure for the operation typically use the same CONTROL as the associated FFU
Length	DINT	immediate	maximum number of elements the FIFO can hold at one time
Position	DINT	immediate	next location in the FIFO where the instruction loads data initial value is typically 0

If you use a user-defined structure as the data type for the Source or FIFO operand, use the same structure for both operands.

CONTROL Structure

Mnemonic:	Data Type:	Description:
.EN	BOOL	The enable bit indicates that the FFL instruction is enabled.
.DN	BOOL	The done bit is set to indicate that the FIFO is full (.POS = .LEN). The .DN bit inhibits loading the FIFO until .POS < .LEN.
.EM	BOOL	The empty bit indicates that the FIFO is empty. If .LEN ≤ 0 or .POS < 0, both the .EM bit and .DN bit are set.
.LEN	DINT	The length specifies the maximum number of elements the FIFO can hold at one time.
.POS	DINT	The position identifies the location in the FIFO where the instruction will load the next value.

Description: Use the FFL instruction with the FFU instruction to store and retrieve data in a first-in/first-out order. When used in pairs, the FFL and FFU instructions establish an asynchronous shift register.

Typically, the Source and the FIFO are the same data type.

When enabled, the FFL instruction loads the Source value into the position in the FIFO identified by the .POS value. The instruction loads one value each time the instruction is enabled, until the FIFO is full.

The FFL instruction operates on contiguous memory.

Arithmetic Status Flags: not affected

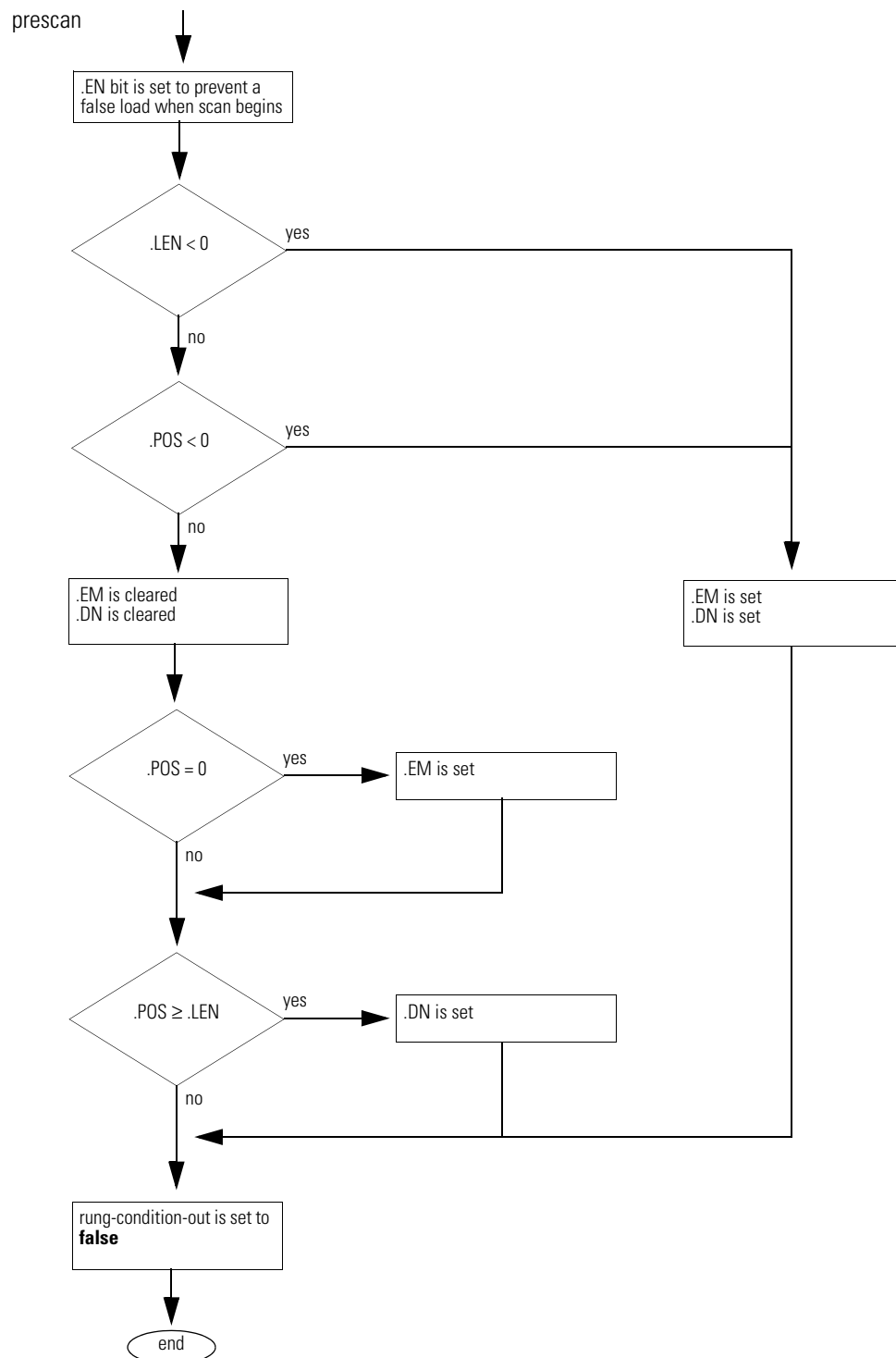
Fault Conditions:

A major fault will occur if:	Fault type:	Fault code:
(starting element + .POS) > FIFO array size	4	20

Execution:

Condition:

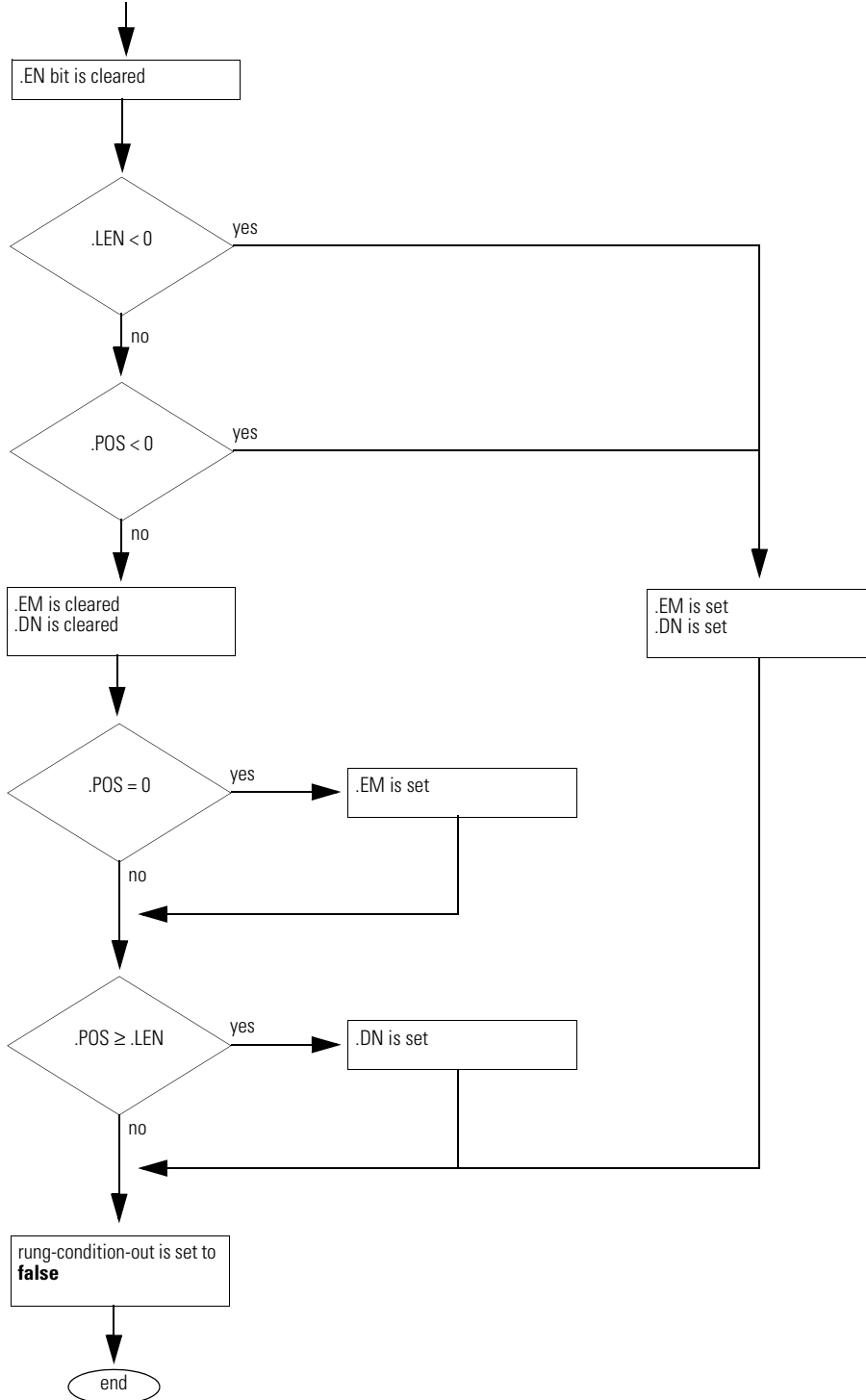
Relay Ladder Action:



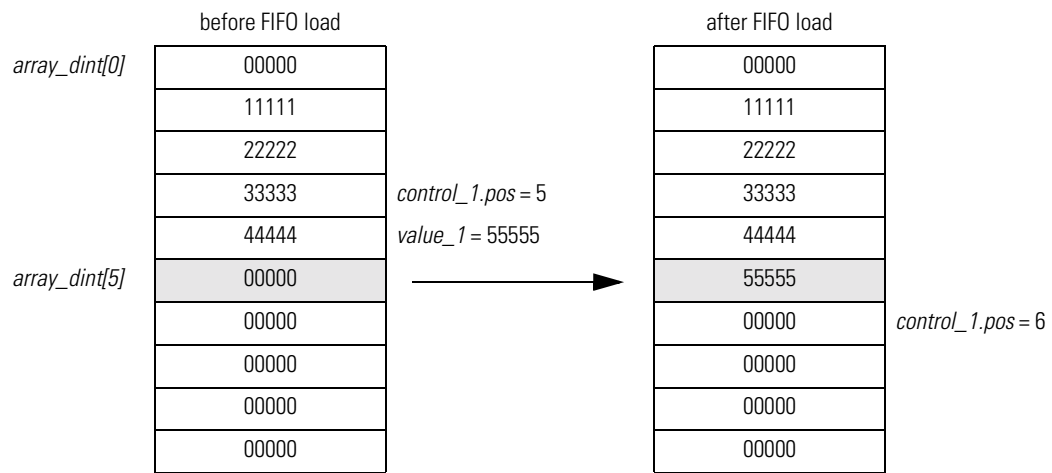
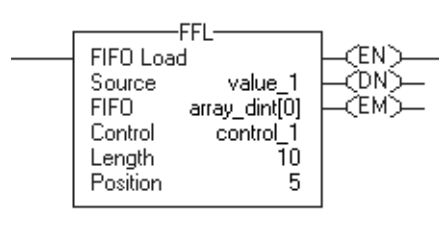
Condition:

Relay Ladder Action:

rung-condition-in is false



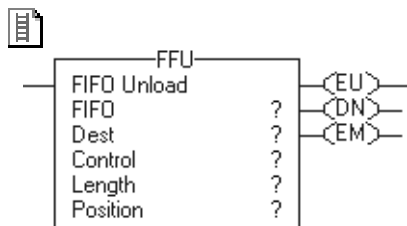
Example: When enabled, the FFL instruction loads *value_1* into the next position in the FIFO, which is *array_dint[5]* in this example.



FIFO Unload (FFU)

The FFU instruction unloads the value from position 0 (first position) of the FIFO and stores that value in the Destination. The remaining data in the FIFO shifts down one position.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
FIFO	SINT INT DINT REAL string structure	array tag	FIFO to modify specify the first element of the FIFO do not use CONTROL.POS in the subscript
Destination	SINT INT DINT REAL string structure	tag	value that exits the FIFO The Destination value converts to the data type of the Destination tag. A smaller integer converts to a larger integer by sign-extension.
Control	CONTROL	tag	control structure for the operation typically use the same CONTROL as the associated FFL
Length	DINT	immediate	maximum number of elements the FIFO can hold at one time
Position	DINT	immediate	next location in the FIFO where the instruction unloads data initial value is typically 0

If you use a user-defined structure as the data type for the FIFO or Destination operand, use the same structure for both operands.

CONTROL Structure

Mnemonic:	Data Type:	Description:
.EU	BOOL	The enable unload bit indicates that the FFU instruction is enabled. The .EU bit is set to preset a false unload when the program scan begins.
.DN	BOOL	The done bit is set to indicate that the FIFO is full (.POS = .LEN).
.EM	BOOL	The empty bit indicates that the FIFO is empty. If .LEN ≤ 0 or .POS < 0, the .EM bit and .DN bits are set.
.LEN	DINT	The length specifies the maximum number of elements in the FIFO.
.POS	DINT	The position identifies the end of the data that has been loaded into the FIFO.

Description: Use the FFU instruction with the FFL instruction to store and retrieve data in a first-in/first-out order.

When enabled, the FFU instruction unloads data from the first element of the FIFO and places that value in the Destination. The instruction unloads one value each time the instruction is enabled, until the FIFO is empty. If the FIFO is empty, the FFU returns 0 to the Destination.

The FFU instruction operates on contiguous memory.

Arithmetic Status Flags: not affected

Fault Conditions:

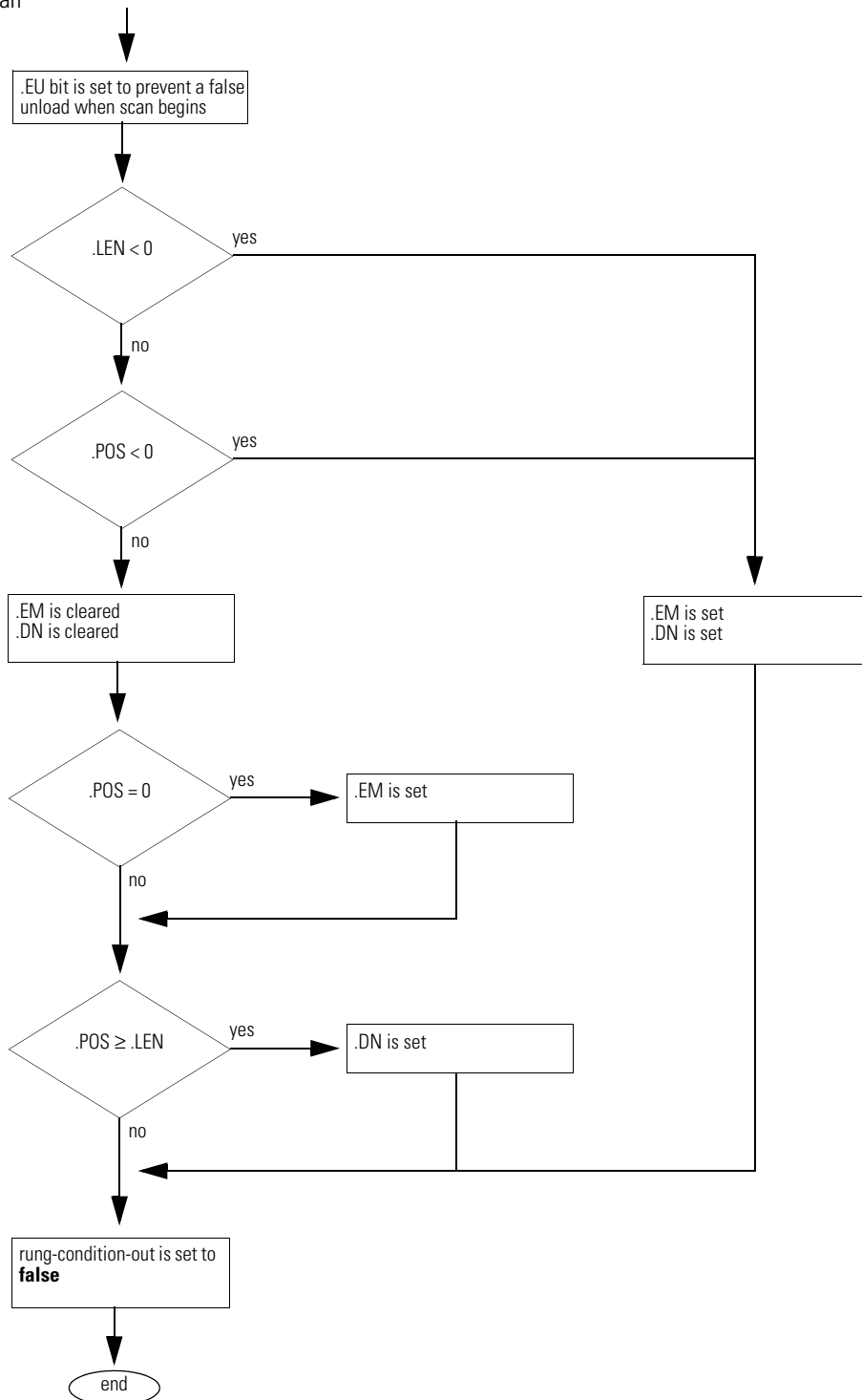
A major fault will occur if:	Fault type:	Fault code:
Length > FIFO array size	4	20

Execution:

Condition:

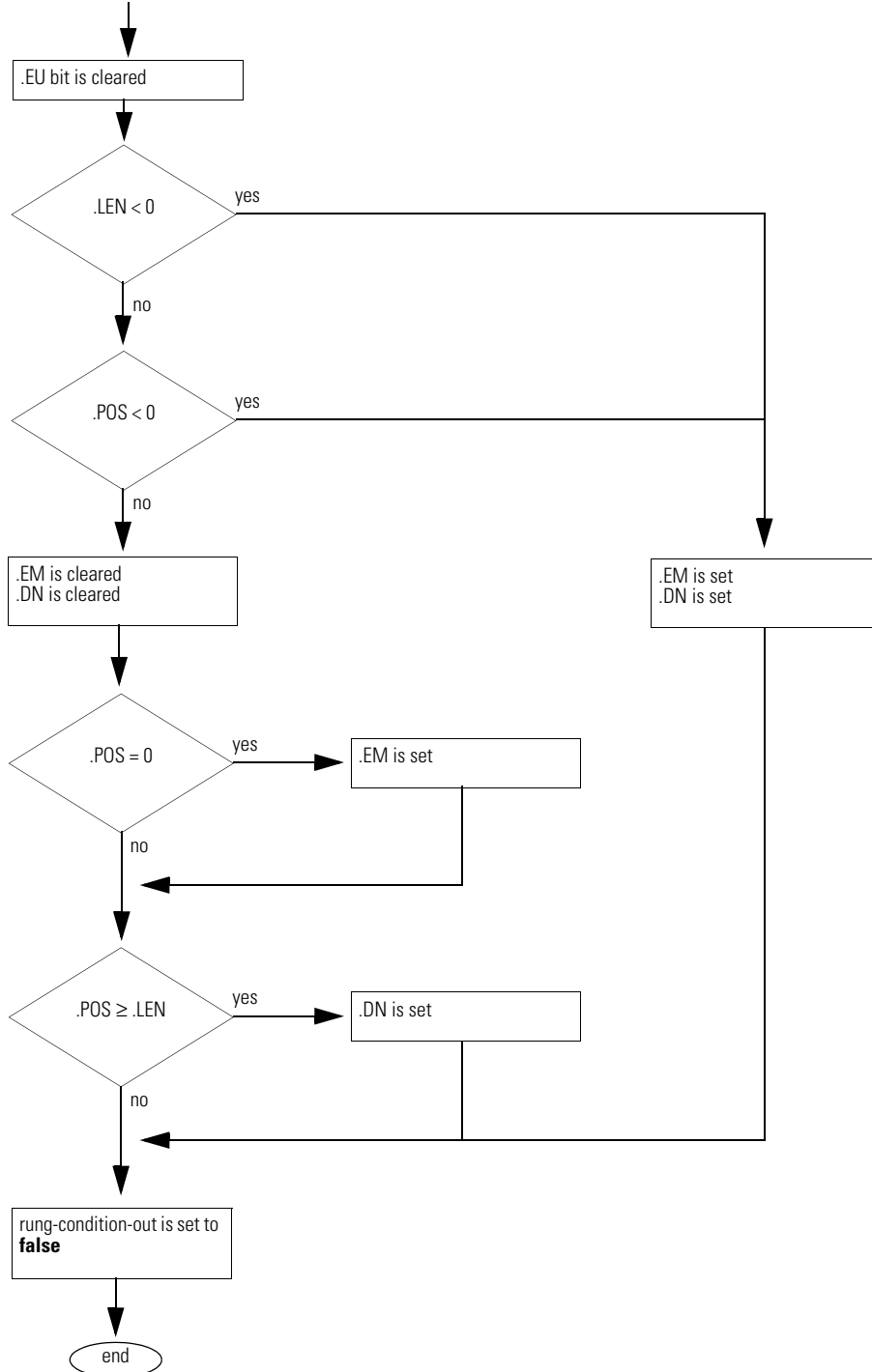
Relay Ladder Action:

prescan



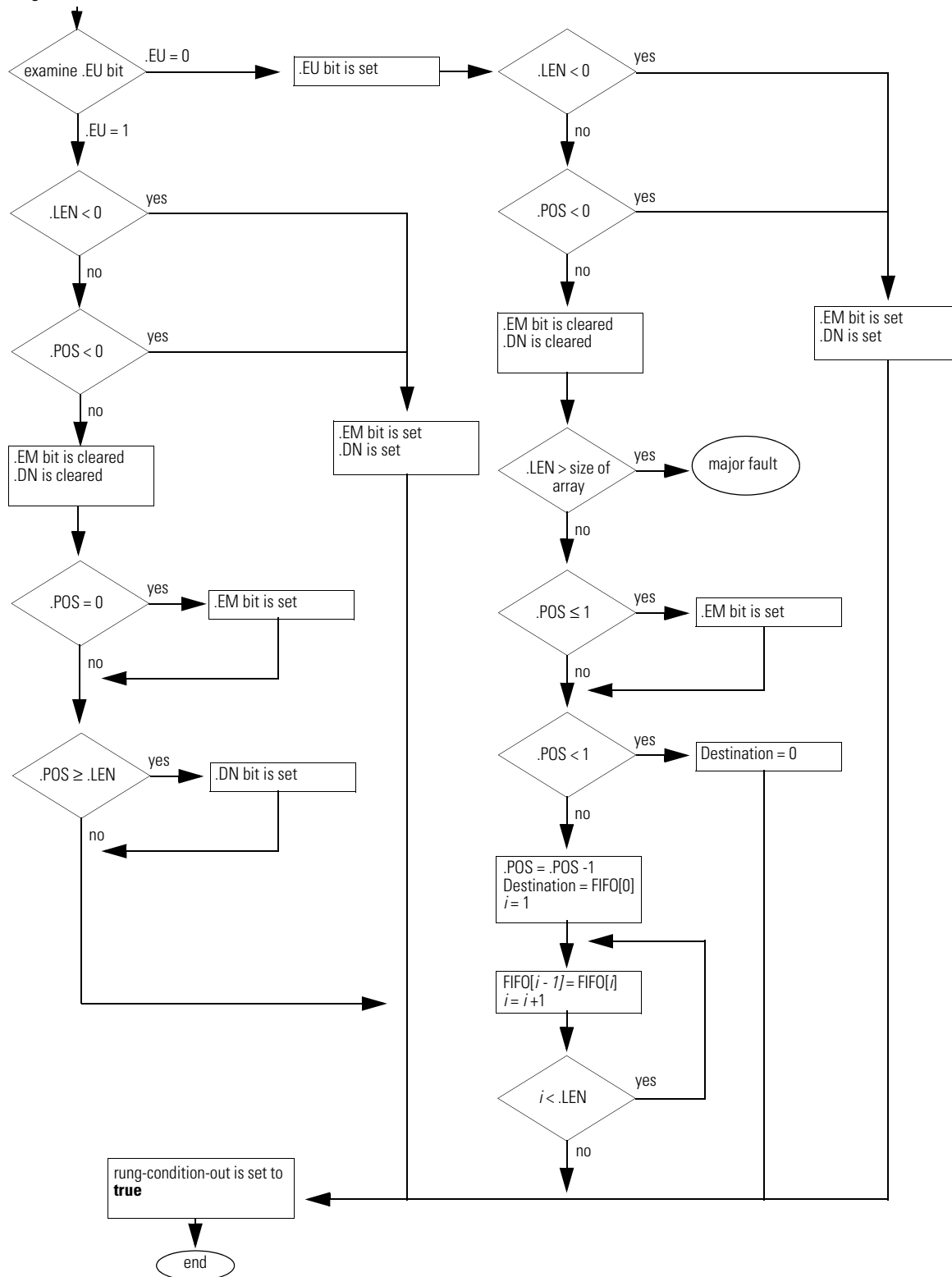
Condition:**Relay Ladder Action:**

rung-condition-in is false



Condition:**Relay Ladder Action:**

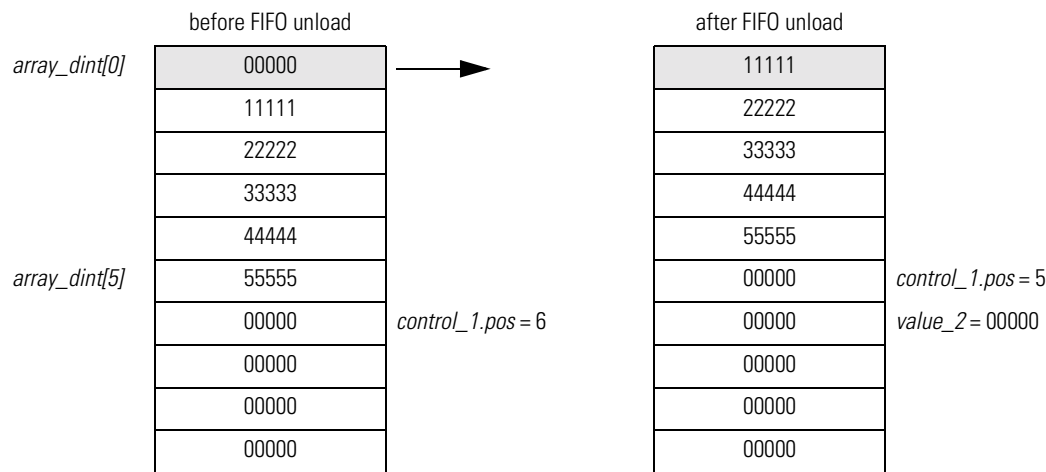
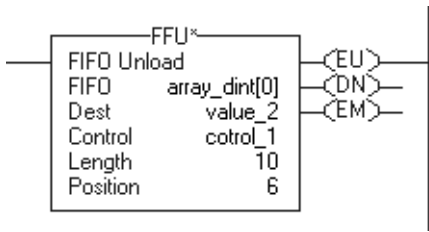
rung-condition-in is true



postscan

The rung-condition-out is set to false.

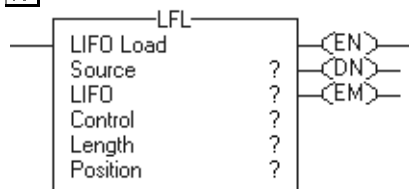
Example: When enabled, the FFU instruction unloads *array_dint[0]* into *value_2* and shifts the remaining elements in *array_dint*.



LIFO Load (LFL)

The LFL instruction copies the Source value to the LIFO.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Source	SINT INT DINT REAL string structure	immediate tag	data to be stored in the LIFO The Source converts to the data type of the array tag. A smaller integer converts to a larger integer by sign-extension.
LIFO	SINT INT DINT REAL string structure	array tag	LIFO to modify specify the first element of the LIFO do not use CONTROL.POS in the subscript
Control	CONTROL	tag	control structure for the operation typically use the same CONTROL as the associated LFU
Length	DINT	immediate	maximum number of elements the LIFO can hold at one time
Position	DINT	immediate	next location in the LIFO where the instruction loads data initial value is typically 0

If you use a user-defined structure as the data type for the Source or LIFO operand, use the same structure for both operands.

CONTROL Structure

Mnemonic:	Data Type:	Description:
.EN	BOOL	The enable bit indicates that the LFL instruction is enabled.
.DN	BOOL	The done bit is set to indicate that the LIFO is full (.POS = .LEN). The .DN bit inhibits loading the LIFO until .POS < .LEN.
.EM	BOOL	The empty bit indicates that the LIFO is empty. If .LEN ≤ 0 or .POS < 0, both the .EM bit and .DN bit are set.
.LEN	DINT	The length specifies the maximum number of elements the LIFO can hold at one time.
.POS	DINT	The position identifies the location in the LIFO where the instruction will load the next value.

Description: Use the LFL instruction with the LFU instruction to store and retrieve data in a last-in/first-out order. When used in pairs, the LFL and LFU instructions establish an asynchronous shift register.

Typically, the Source and the LIFO are the same data type.

When enabled, the LFL instruction loads the Source value into the position in the LIFO identified by the .POS value. The instruction loads one value each time the instruction is enabled, until the LIFO is full.

The LFL instruction operates on contiguous memory.

Arithmetic Status Flags: not affected

Fault Conditions:

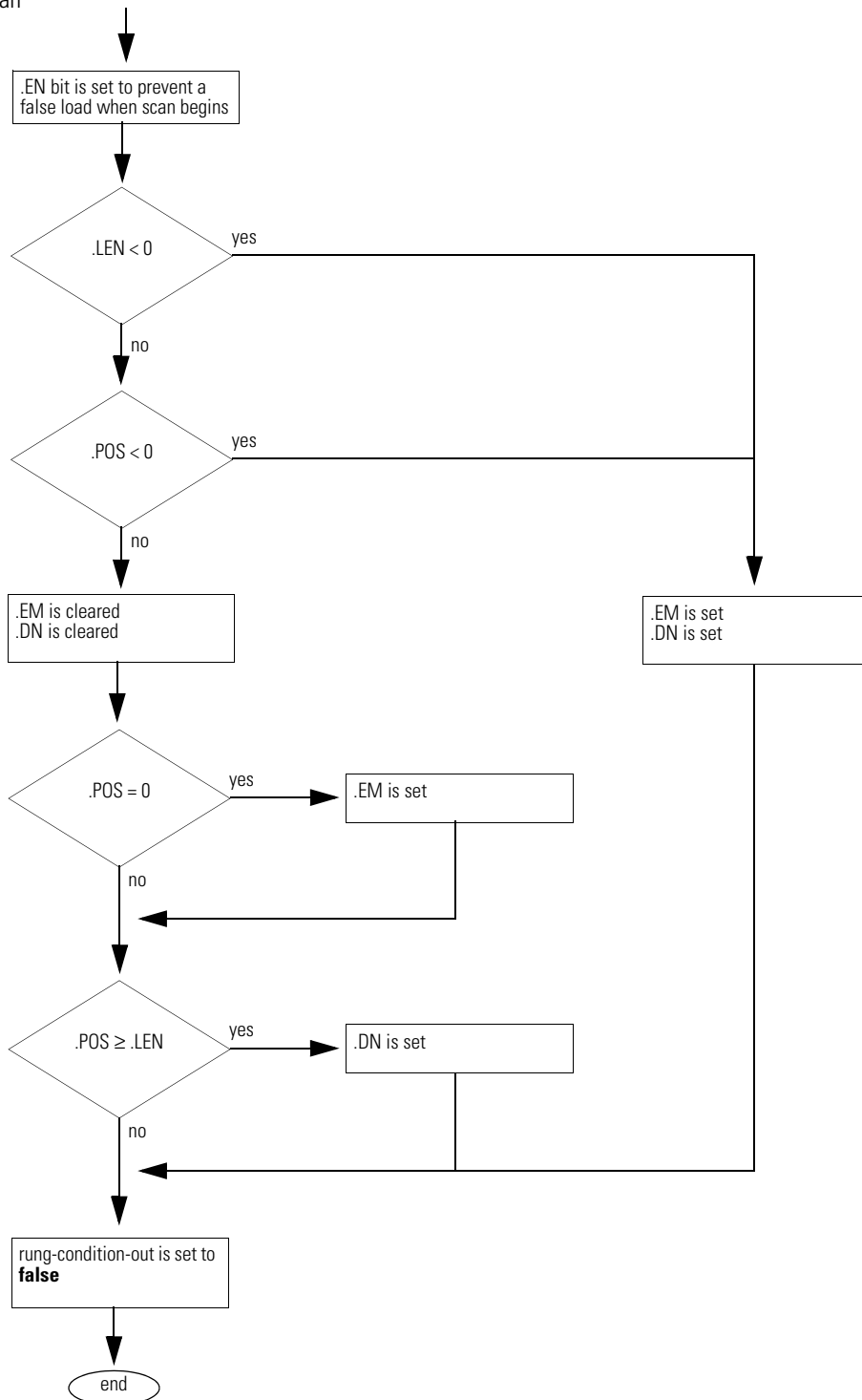
A major fault will occur if:	Fault type:	Fault code:
(starting element + .POS) > LIFO array size	4	20

Execution:

Condition:

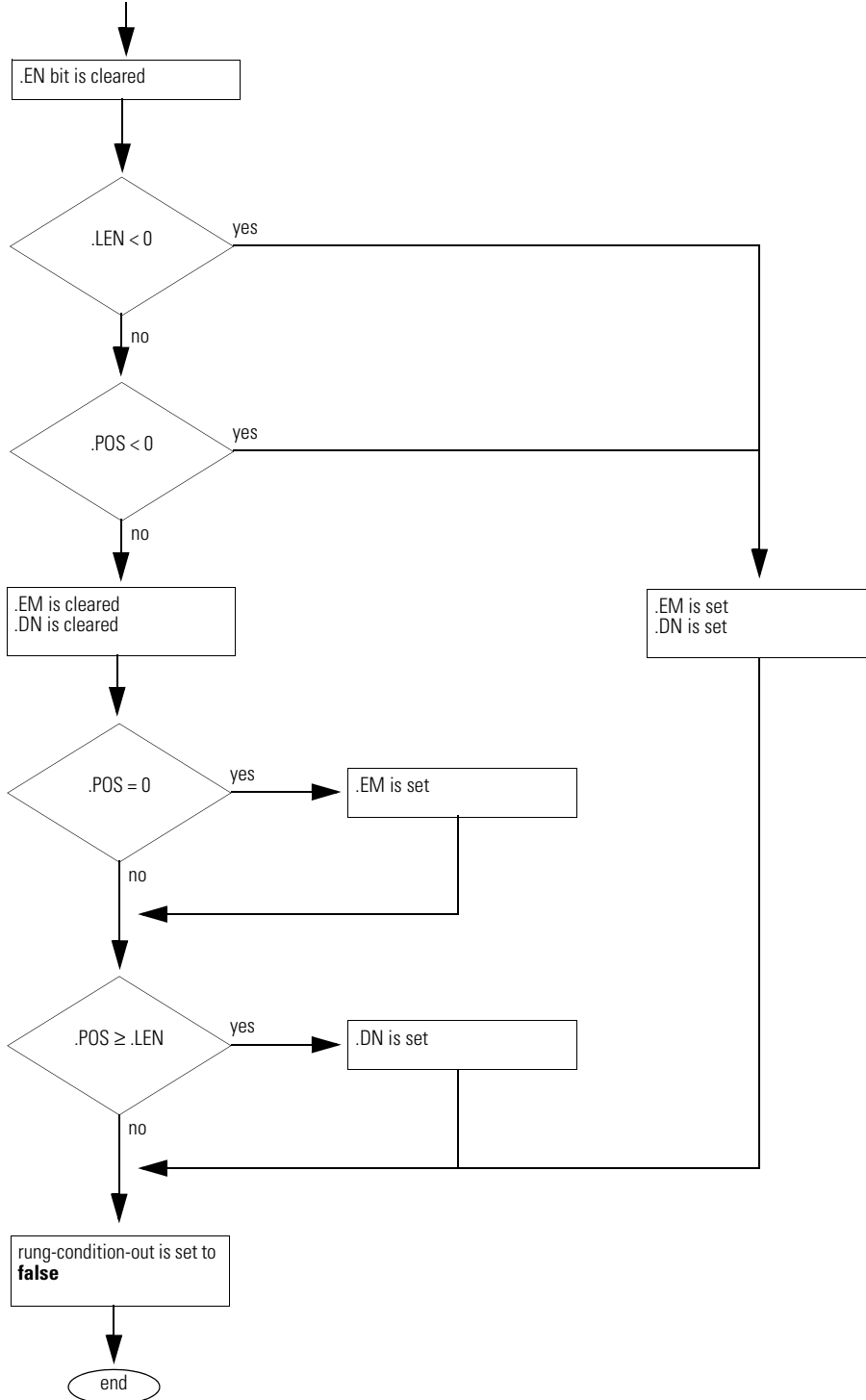
Relay Ladder Action:

prescan



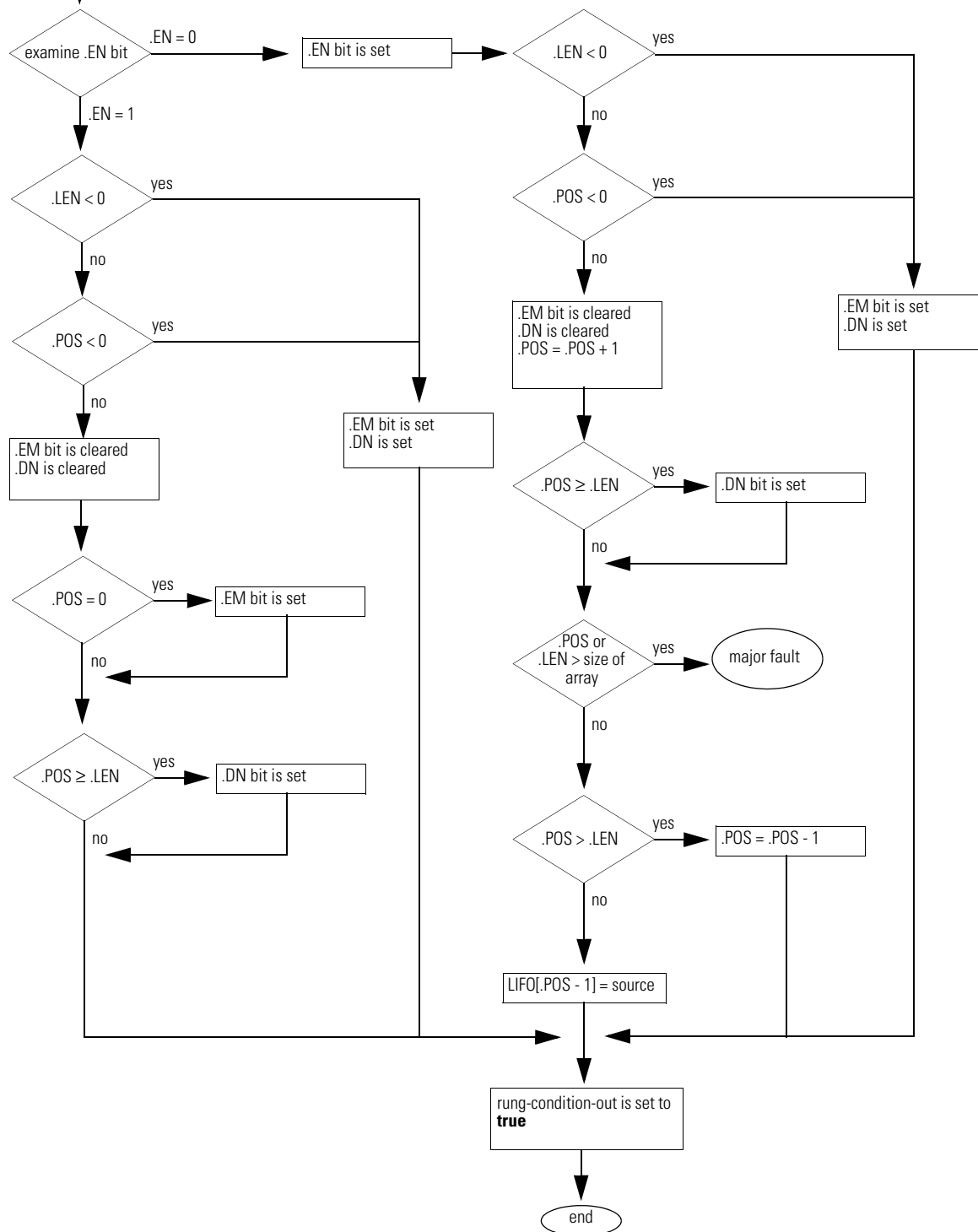
Condition:**Relay Ladder Action:**

rung-condition-in is false



Condition:**Relay Ladder Action:**

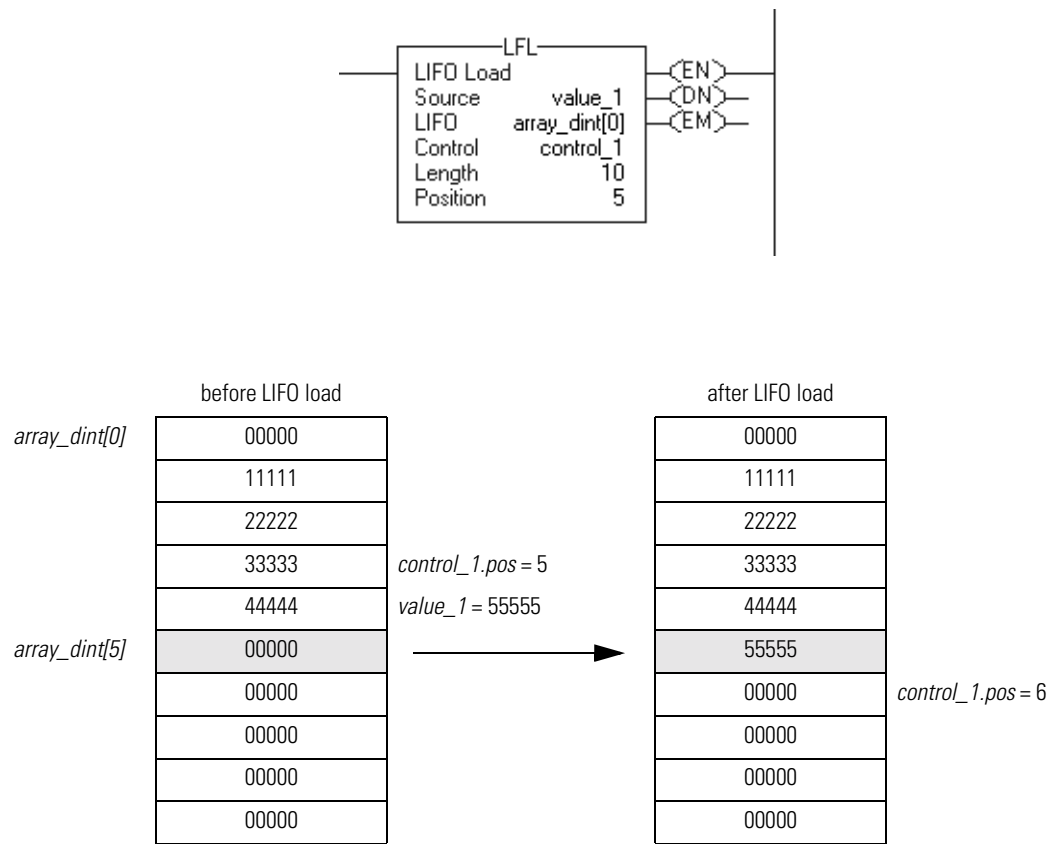
rung-condition-in is true



postscan

The rung-condition-out is set to false.

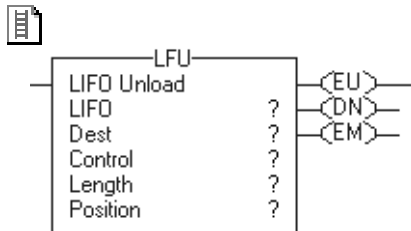
Example: When enabled, the LFL instruction loads *value_1* into the next position in the LIFO, which is *array_dint[5]* in this example.



LIFO Unload (LFU)

The LFU instruction unloads the value at .POS of the LIFO and stores 0 in that location.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
LIFO	SINT INT DINT REAL string structure	array tag	LIFO to modify specify the first element of the LIFO do not use CONTROL.POS in the subscript
Destination	SINT INT DINT REAL string structure	tag	value that exits the LIFO The Destination value converts to the data type of the Destination tag. A smaller integer converts to a larger integer by sign-extension.
Control	CONTROL	tag	control structure for the operation typically use the same CONTROL as the associated LFL
Length	DINT	immediate	maximum number of elements the LIFO can hold at one time
Position	DINT	immediate	next location in the LIFO where the instruction unloads data initial value is typically 0

If you use a user-defined structure as the data type for the LIFO or Destination operand, use the same structure for both operands.

CONTROL Structure

Mnemonic:	Data Type:	Description:
.EU	BOOL	The enable unload bit indicates that the LFU instruction is enabled. The .EU bit is set to preset a false unload when the program scan begins.
.DN	BOOL	The done bit is set to indicate that the LIFO is full (.POS = .LEN).
.EM	BOOL	The empty bit indicates that the LIFO is empty. If .LEN ≤ 0 or .POS < 0, both the .EM bit and .DN bit are set.
.LEN	DINT	The length specifies the maximum number of elements the LIFO can hold at one time.
.POS	DINT	The position identifies the end of the data that has been loaded into the LIFO.

Description: Use the LFU instruction with the LFL instruction to store and retrieve data in a last-in/first-out order.

When enabled, the LFU instruction unloads the value at .POS of the LIFO and places that value in the Destination. The instruction unloads one value and replaces it with 0 each time the instruction is enabled, until the LIFO is empty. If the LIFO is empty, the LFU returns 0 to the Destination.

The LFU instruction operates on contiguous memory.

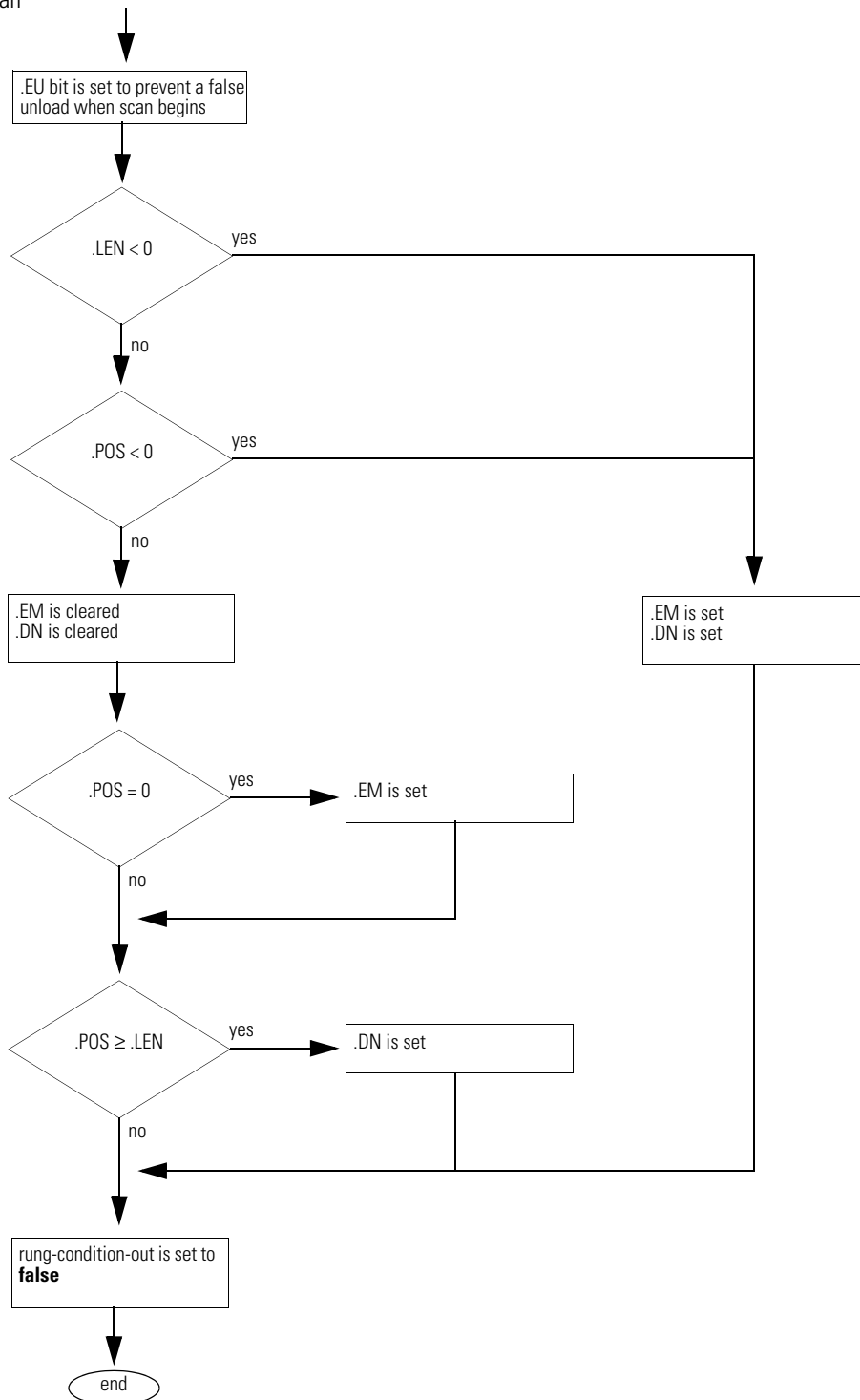
Arithmetic Status Flags: not affected

Fault Conditions:

A major fault will occur if:	Fault type:	Fault code:
Length > LIFO array size	4	20

Execution:**Condition:****Relay Ladder Action:**

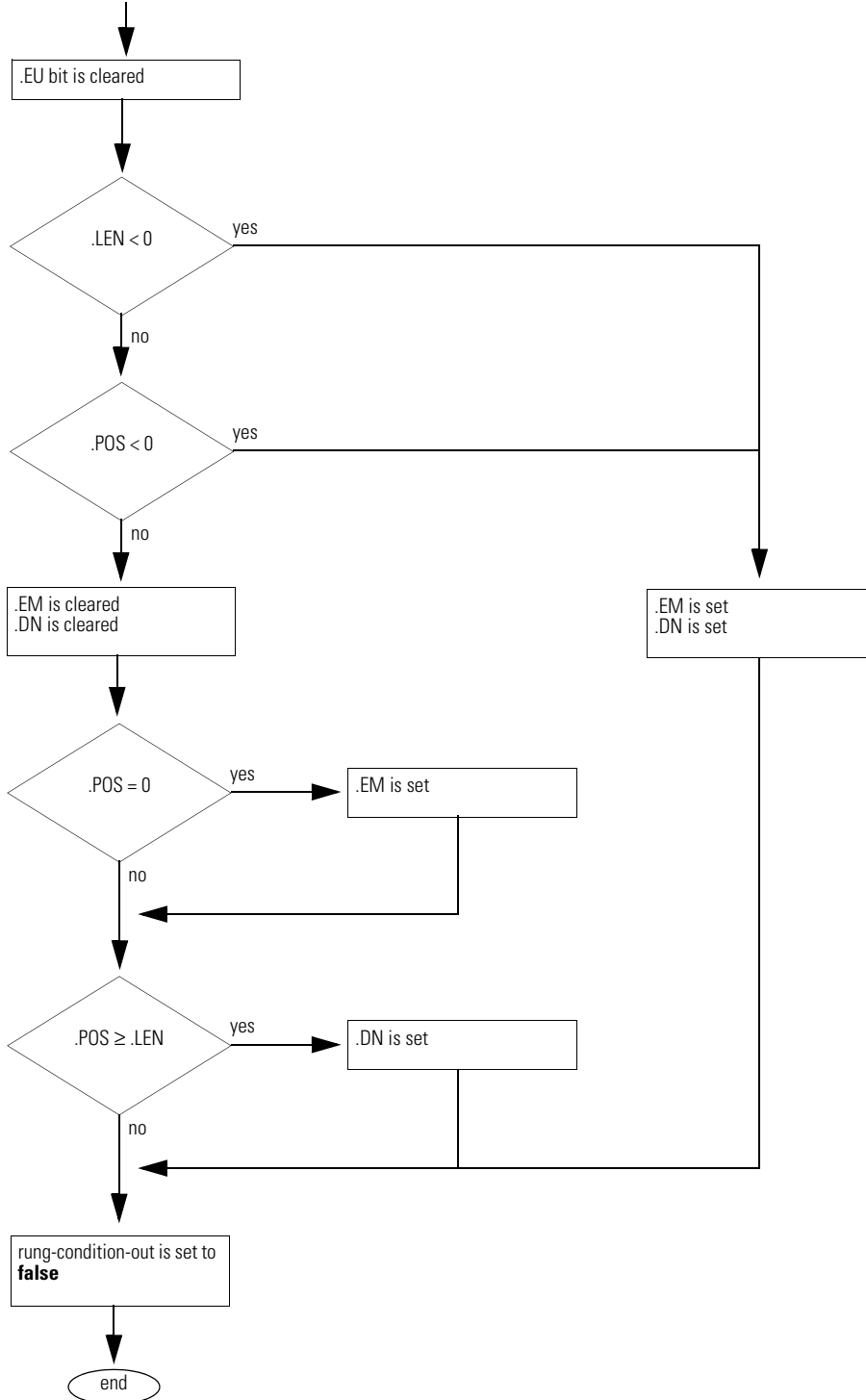
prescan



Condition:

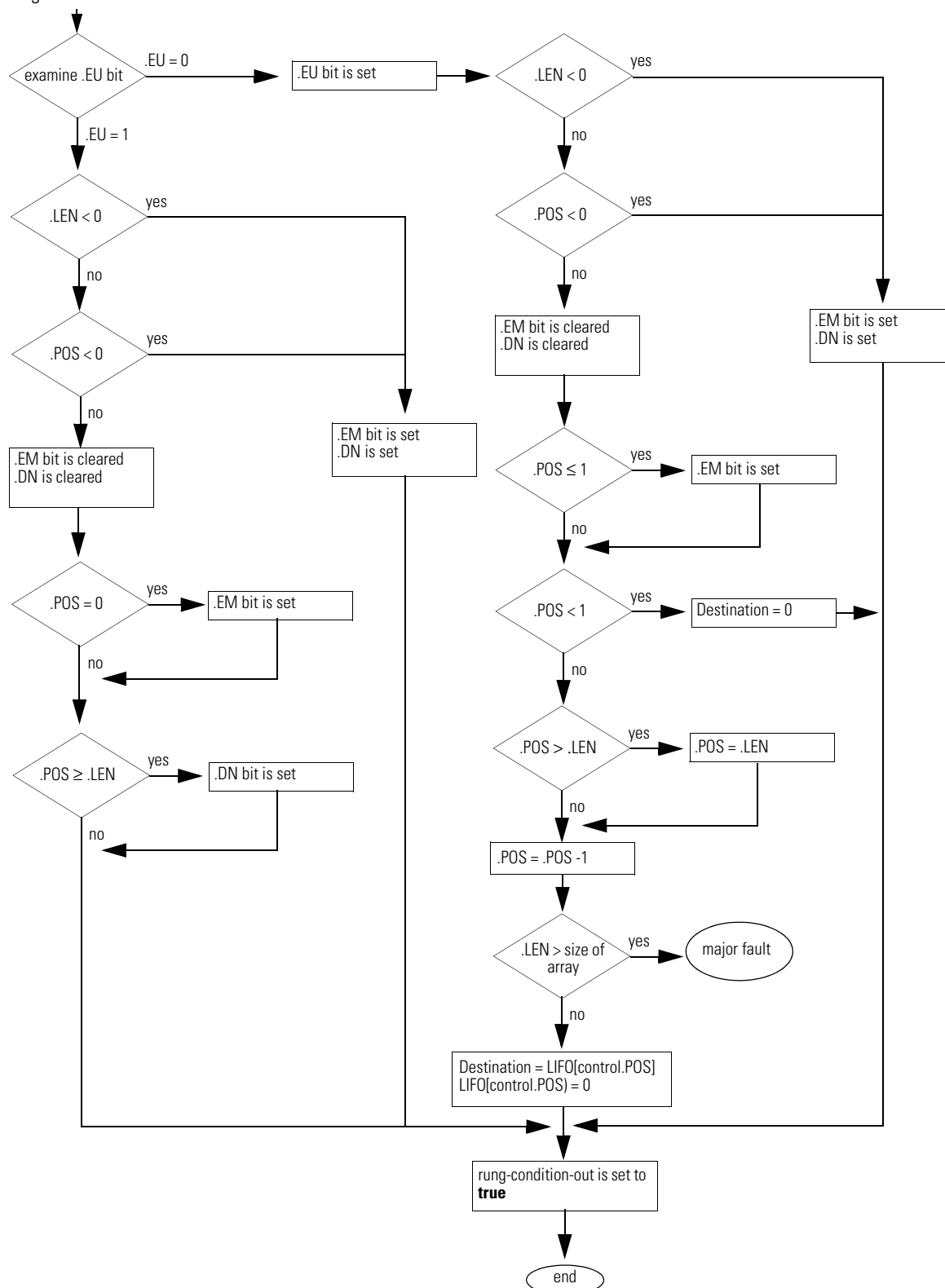
Relay Ladder Action:

rung-condition-in is false



Condition:**Relay Ladder Action:**

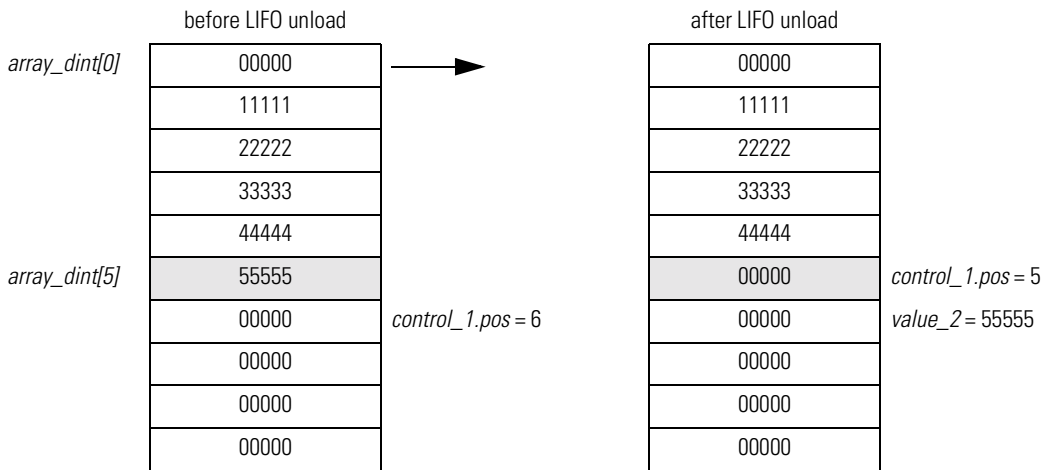
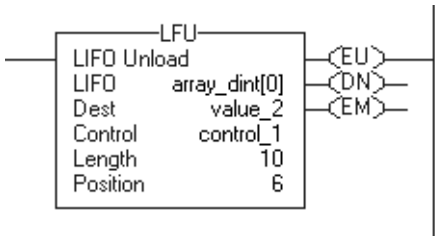
rung-condition-in is true



postscan

The rung-condition-out is set to false.

Example: When enabled, the LFU instruction unloads *array_dint[5]* into *value_2*.



Notes:

Sequencer Instructions

(SQI, SQO, SQL)

Introduction

No action taken. Sequencer instructions monitor consistent and repeatable operations.

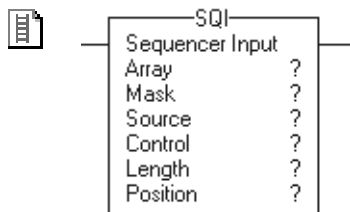
If you want to:	Use this instruction:	Available in these languages:	See page:
Detect when a step is complete.	SQI	relay ladder	9-2
Set output conditions for the next step.	SQO	relay ladder	9-6
Load reference conditions into sequencer arrays	SQL	relay ladder	9-10

For relay ladder instructions, **bold** data types indicate optimal data types. An instruction executes faster and requires less memory if all the operands of the instruction use the same optimal data type, typically DINT or REAL.

Sequencer Input (SQI)

The SQI instruction detects when a step is complete in a sequence pair of SQO/SQI instructions.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Array	DINT	array tag	sequencer array specify the first element of the sequencer array do not use CONTROL.POS in the subscript
Mask	SINT INT DINT	tag immediate	which bits to block or pass A SINT or INT tag converts to a DINT value by sign-extension.
Source	SINT INT DINT	tag	input data for the sequencer array A SINT or INT tag converts to a DINT value by sign-extension.
Control	CONTROL	tag	control structure for the operation typically use the same CONTROL as the SQO and SQL instructions
Length	DINT	immediate	number of elements in the Array (sequencer table) to compare
Position	DINT	immediate	current position in the array initial value is typically 0

CONTROL Structure

Mnemonic:	Data Type:	Description:
.ER	BOOL	The error bit is set when .LEN \leq 0, .POS < 0, or .POS > .LEN.
.LEN	DINT	The length specifies the number of steps in the sequencer array.
.POS	DINT	The position identifies the element that the instruction is currently comparing.

Description: When enabled, the SQI instruction compares a Source element through a Mask to an Array element for equality.

Typically use the same CONTROL structure as the SQO and SQL instructions.

The SQI instruction operates on contiguous memory.

Entering an immediate mask value

When you enter a mask, the programming software defaults to decimal values. If you want to enter a mask using another format, precede the value with the correct prefix.

Prefix:	Description:
16#	hexadecimal for example; 16#0F0F
8#	octal for example; 8#16
2#	binary for example; 2#00110011

Arithmetic Status Flags: not affected

Fault Conditions: none

Execution:

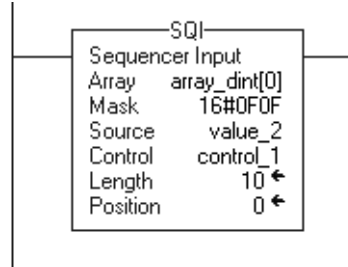
Condition:	Relay Ladder Action:
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	


```

graph TD
    Start([rung-condition-in is true]) --> D1{'.LEN <= 0  
.POS < 0  
or  
.POS > .LEN'}
    D1 -- no --> A1[.ER bit is cleared]
    D1 -- yes --> A2[.ER bit is set]
    A1 --> D2{masked Source =  
masked Array[.POS]}
    D2 -- yes --> A3[rung-condition-out is set to  
true]
    D2 -- no --> A2
    A2 --> A4[rung-condition-out is set to  
false]
    A3 --> End([end])
    A4 --> End
  
```


postscan	The rung-condition-out is set to false.
----------	---

Example: When enabled, the SQI instruction passes *value_2* through the mask to determine whether the result is equal to the current element in *array_dint*. The masked comparison is true, so the rung-condition-out goes true.



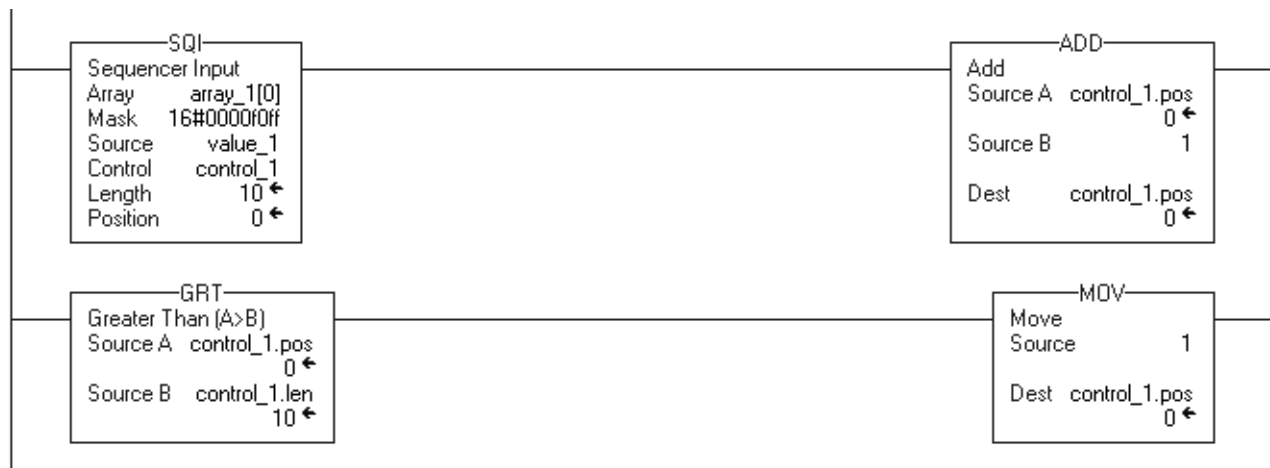
SQI operand:	Example values (DINTs displayed in binary):
Source	xxxxxxxx xxxxxxxx xxxx0101 xxxx1010
Mask	00000000 00000000 00001111 00001111
Array	xxxxxxxx xxxxxxxx xxxx0101 xxxx1010

A 0 in the mask means the bit is not compared (designated by xxxx in this example).

Using SQI without SQO

If you use the SQI instruction without a paired SQO instruction, you have to externally increment the sequencer array.

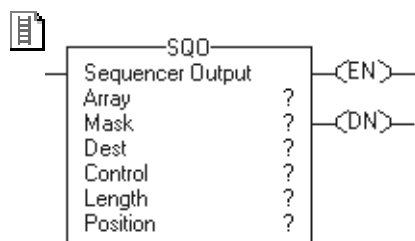
The SQI instruction compares the source value. The ADD instruction increments the sequencer array. The GRT determined whether another value is available to check in the sequencer array. The MOV instruction resets the position value after completely stepping through the sequencer array one time.



Sequencer Output (SQO)

The SQO instruction sets output conditions for the next step of a sequence pair of SQO/SQI instructions.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Array	DINT	array tag	sequencer array specify the first element of the sequencer array do not use CONTROL.POS in the subscript
Mask	SINT INT DINT	tag immediate	which bits to block or pass A SINT or INT tag converts to a DINT value by sign-extension.
Destination	DINT	tag	output data from the sequencer array
Control	CONTROL	tag	control structure for the operation typically use the same CONTROL as the SQI and SQL instructions
Length	DINT	immediate	number of elements in the Array (sequencer table) to output
Position	DINT	immediate	current position in the array initial value is typically 0

CONTROL Structure

Mnemonic:	Data Type:	Description:
.EN	BOOL	The enable bit indicates that the SQO instruction is enabled.
.DN	BOOL	The done bit is set when all the specified elements have been moved to the Destination.
.ER	BOOL	The error bit is set when .LEN ≤ 0, .POS < 0, or .POS > .LEN.
.LEN	DINT	The length specifies the number of steps in the sequencer array.
.POS	DINT	The position identifies the element that the controller is currently manipulating.

Description: When enabled, the SQO instruction increments the position, moves the data at the position through the Mask, and stores the result in the Destination. If .POS > .LEN, the instruction wraps around to the beginning of the sequencer array and continues with .POS = 1.

Typically, use the same CONTROL structure as the SQI and SQL instructions.

The SQO instruction operates on contiguous memory.

Entering an immediate mask value

When you enter a mask, the programming software defaults to decimal values. If you want to enter a mask using another format, precede the value with the correct prefix.

Prefix:	Description:
16#	hexadecimal for example; 16#0F0F
8#	octal for example; 8#16
2#	binary for example; 2#00110011

Arithmetic Status Flags: not affected

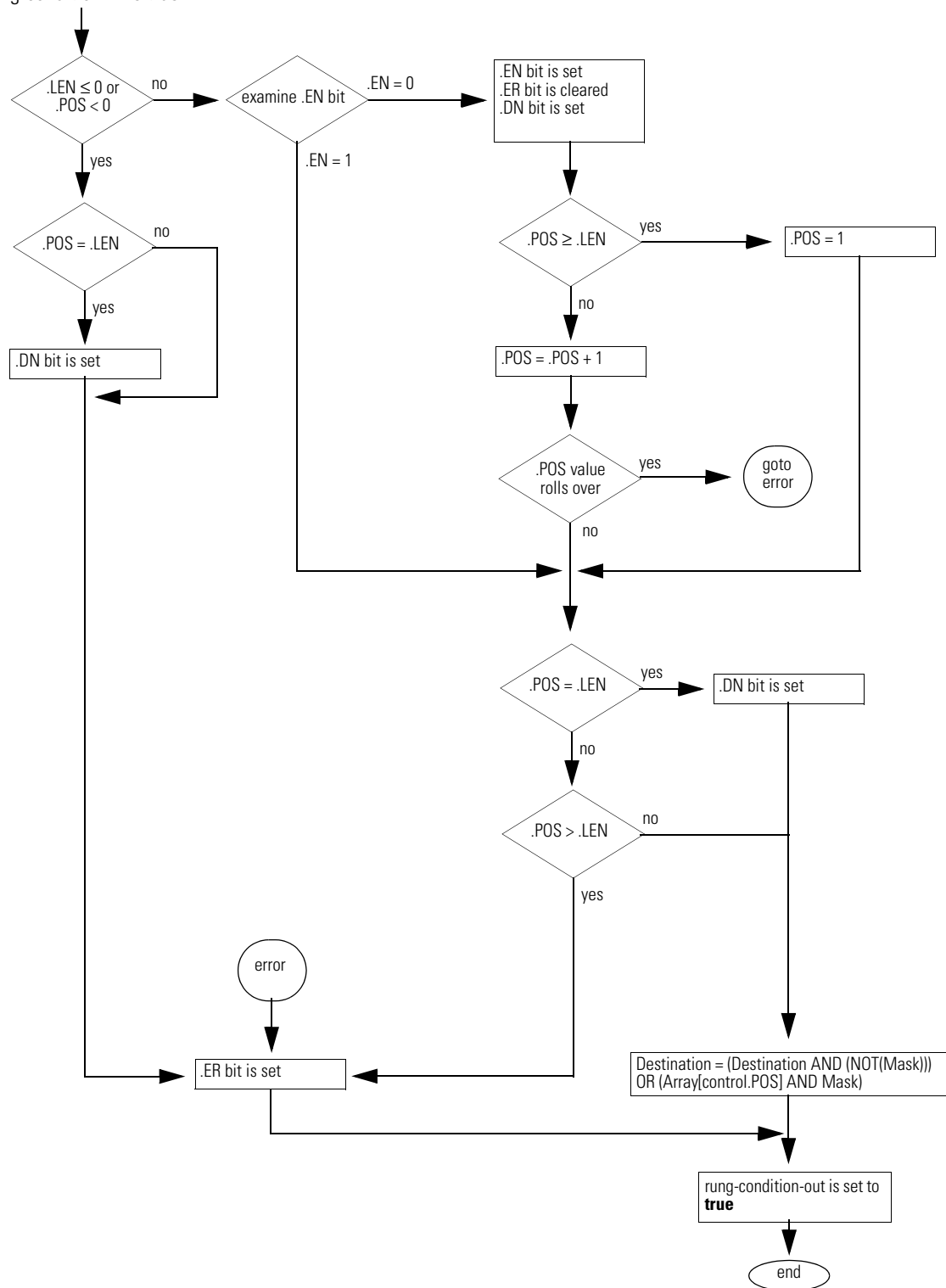
Fault Conditions: none

Execution:

Condition:	Relay Ladder Action:
prescan	The .EN bit is set to prevent a false load when the program scan begins. The rung-condition-out is set to false.
rung-condition-in is false	The .EN bit is cleared. The rung-condition-out is set to false.

Condition:
Relay Ladder Action:

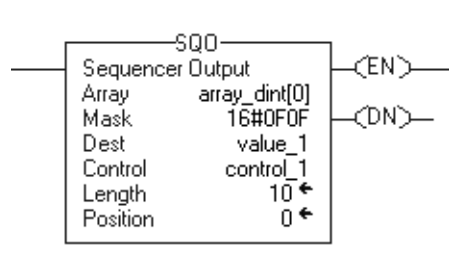
rung-condition-in is true



postscan

The rung-condition-out is set to false.

Example: When enabled, the SQO instruction increments the position, passes the data at that position in *array_dint* through the mask, and stores the result in *value_1*.



SQO operand:	Example values (using INTs displayed in binary):
Array	xxxxxxxx xxxxxxxx xxxx0101 xxxx1010
Mask	00000000 00000000 00001111 00001111
Destination	xxxxxxxx xxxxxxxx xxxx0101 xxxx1010

A 0 in the mask means the bit is not compared (designated by xxxx in this example).

Using SQI with SQO

If you pair an SQI instruction with an SQO instruction, make sure that both instructions use the same Control, Length, and Position values,.



Resetting the position of SQO

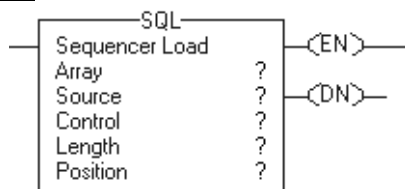
Each time the controller goes from Program to Run mode, the SQO instruction clears (initializes) the .POS value. To reset .POS to the initialization value (.POS = 0), use a RES instruction to clear the position value. This example uses the status of the first-scan bit to clear the .POS value.



Sequencer Load (SQL)

The SQL instruction loads reference conditions into a sequencer array.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Array	DINT	array tag	sequencer array specify the first element of the sequencer array do not use CONTROL.POS in the subscript
Source	SINT INT DINT	tag immediate	input data to load into the sequencer array A SINT or INT tag converts to a DINT value by sign-extension.
Control	CONTROL	tag	control structure for the operation typically use the same CONTROL as the SQI and SQO instructions
Length	DINT	immediate	number of elements in the Array (sequencer table) to load
Position	DINT	immediate	current position in the array initial value is typically 0

CONTROL Structure

Mnemonic:	Data Type:	Description:
.EN	BOOL	The enable bit indicates that the SQL instruction is enabled.
.DN	BOOL	The done bit is set when all the specified elements have been loaded into Array.
.ER	BOOL	The error bit is set when .LEN ≤ 0, .POS < 0, or .POS > .LEN.
.LEN	DINT	The length specifies the number of steps in the sequencer array.
.POS	DINT	The position identifies the element that the controller is currently manipulating.

Description: When enabled, the SQL instruction increments to the next position in the sequencer array and loads the Source value into that position. If the .DN bit is set or if .POS ≥ .LEN, the instruction sets .POS=1.

Typically use the same CONTROL structure as the SQI and SQO instructions.

The SQL instruction operates on contiguous memory.

Arithmetic Status Flags: not affected

Fault Conditions:

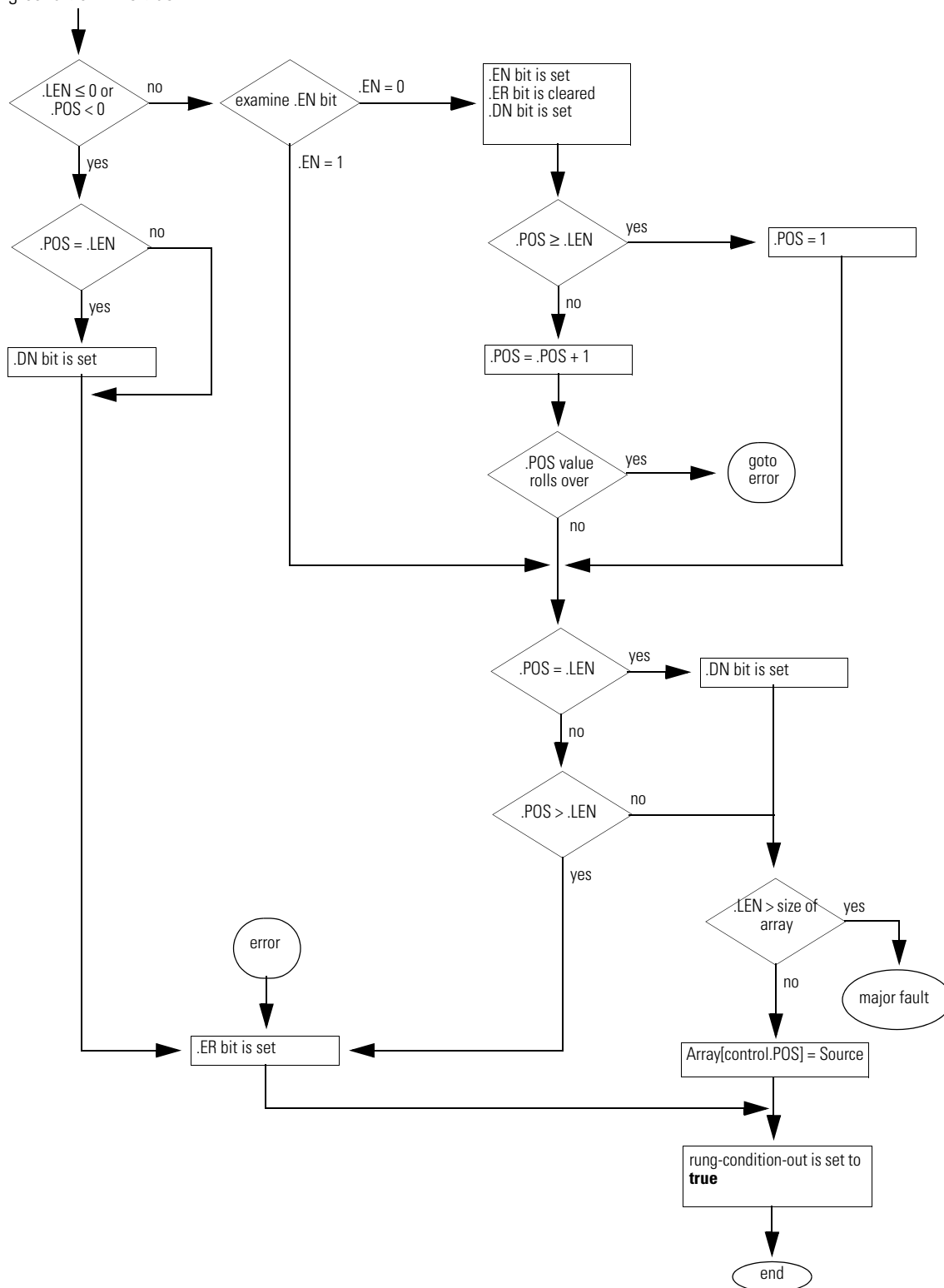
A major fault will occur if:	Fault type:	Fault code:
Length > size of Array	4	20

Execution:

Condition:	Relay Ladder Action:
prescan	The .EN bit is set to prevent a false load when the program scan begins. The rung-condition-out is set to false.
rung-condition-in is false	The .EN bit is cleared. The rung-condition-out is set to false.

Condition:**Relay Ladder Action:**

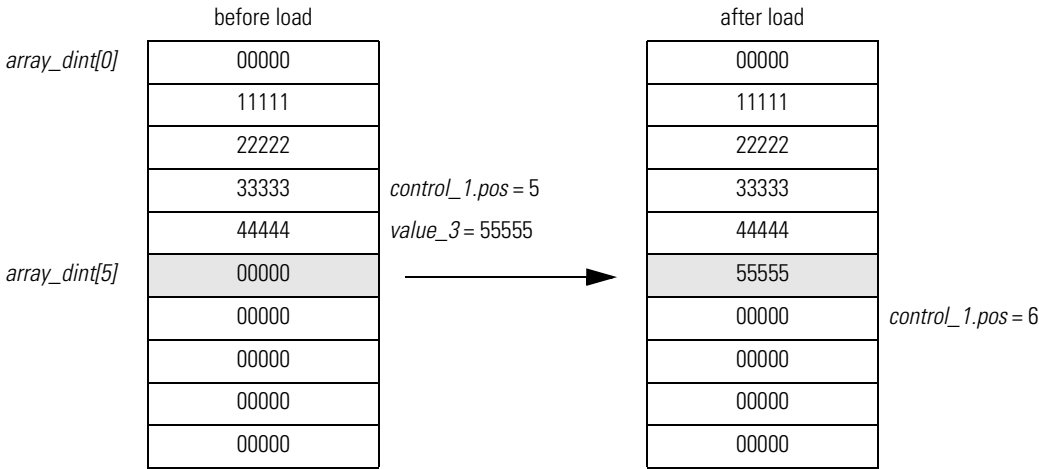
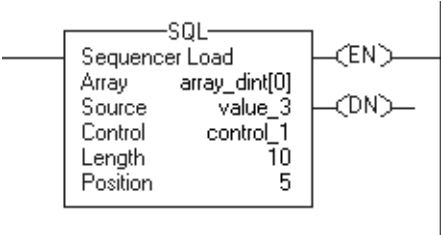
rung-condition-in is true



postscan

The rung-condition-out is set to false.

Example: When enabled, the SQL instruction loads *value_3* into the next position in the sequencer array, which is *array_dint[5]* in this example.



Notes:

Program Control Instructions

(JMP, LBL, JSR, RET, SBR, JXR, TND, MCR, UID, UIE, AFI,
NOP, EOT, SFP, SFR, EVENT)

Introduction

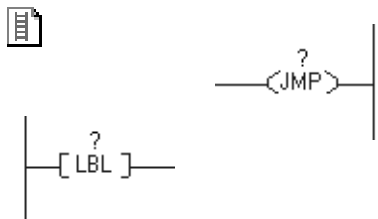
Use the program control instructions to change the flow of logic.

If you want to:	Use this instruction:	Available in these languages:	See page:
Jump over a section of logic that does not always need to be executed.	JMP LBL	relay ladder	10-2
Jump to a separate routine, pass data to the routine, execute the routine, and return results.	JSR SBR RET	relay ladder function block structured text	10-4
Jump to an external routine (SoftLogix5800 controller only)	JXR	relay ladder	10-14
Mark a temporary end that halts routine execution.	TND	relay ladder structured text	10-17
Disable all the rungs in a section of logic.	MCR	relay ladder	10-19
Disable user tasks.	UID	relay ladder structured text	10-21
Enable user tasks.	UIE	relay ladder structured text	10-21
Disable a rung.	AFI	relay ladder	10-23
Insert a placeholder in the logic.	NOP	relay ladder	10-24
End a transition for a sequential function chart.	EOT	relay ladder structured text	10-25
Pause a sequential function chart.	SFP	relay ladder structured text	10-27
Reset a sequential function chart.	SFR	relay ladder structured text	10-29
Trigger the execution of an event task	EVENT	relay ladder structured text	10-31

Jump to Label (JMP) Label (LBL)

The JMP and LBL instructions skip portions of ladder logic.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
JMP instruction			
Label name		label name	enter name for associated LBL instruction
LBL instruction			
Label name		label name	execution jumps to LBL instruction with referenced label name

Description: When enabled, the JMP instruction skips to the referenced LBL instruction and the controller continues executing from there. When disabled, the JMP instruction does not affect ladder execution.

The JMP instruction can move ladder execution forward or backward. Jumping forward to a label saves program scan time by omitting a logic segment until it's needed. Jumping backward lets the controller repeat iterations of logic.

Be careful not to jump backward an excessive number of times. The watchdog timer could time out because the controller never reaches the end of the logic, which in turn faults the controller.

ATTENTION

Jumped logic is not scanned. Place critical logic outside the jumped zone.



The LBL instruction is the target of the JMP instruction that has the same label name. **Make sure the LBL instruction is the first instruction on its rung.**

A label name must be unique within a routine. The name can:

- have as many as 40 characters
- contain letters, numbers, and underscores (_)

Arithmetic Status Flags: not affected

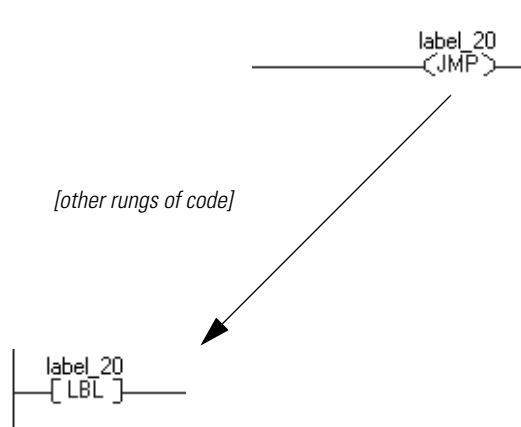
Fault Conditions:

A major fault will occur if:	Fault type:	Fault code:
label does not exist	4	42

Execution:

Condition:	Relay Ladder Action:
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	The rung-condition-out is set to true. Execution jumps to the rung that contains the LBL instruction with the referenced label name.
postscan	The rung-condition-out is set to false.

Example: When the JMP instruction is enabled, execution jumps over successive rungs of logic until it reaches the rung that contains the LBL instruction with *label_20*.



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

The JSR instruction jumps execution to a different routine. The SBR and RET instructions are optional instructions that exchange data with the JSR instruction.

JSR Operands:



JSR	
Jump to Subroutine	
Routine name	?
Input par	?
Return par	?

Relay Ladder

Operand:	Type:	Format:	Description:
Routine name	ROUTINE	name	routine to execute (i.e., subroutine)
Input parameter	BOOL SINT INT DINT REAL structure	immediate tag array tag	data from this routine that you want to copy to a tag in the subroutine <ul style="list-style-type: none"> Input parameters are optional. Enter multiple input parameters, if needed.
Return parameter	BOOL SINT INT DINT REAL structure	tag array tag	tag in this routine to which you want to copy a result of the subroutine <ul style="list-style-type: none"> Return parameters are optional. Enter multiple return parameters, if needed.



```
JSR (RoutineName, InputCount,
    InputPar, ReturnPar) ;
```

Structured Text

Operand:	Type:	Format:	Description:
Routine name	ROUTINE	name	routine to execute (i.e., subroutine)
Input count	SINT INT DINT REAL	immediate	number of input parameters
Input parameter	BOOL SINT INT DINT REAL structure	immediate tag array tag	data from this routine that you want to copy to a tag in the subroutine <ul style="list-style-type: none"> Input parameters are optional. Enter multiple input parameters, if needed.
Return parameter	BOOL SINT INT DINT REAL structure	tag array tag	tag in this routine to which you want to copy a result of the subroutine <ul style="list-style-type: none"> Return parameters are optional. Enter multiple return parameters, if needed.

(JSR operands continued on next page)

JSR Operands (continued)



JSR

Jump to Subroutine

Routine: ?

Function Block

JSR

Jump to Subroutine

Routine: ?

Input Parameters		Return Parameters	
*		*	

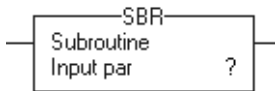
The operands are the same as those for the relay ladder JSR instruction.

ATTENTION



For each parameter in a SBR or RET instruction, use the same data type (including any array dimensions) as the corresponding parameter in the JSR instruction. Using different data types may produce unexpected results.

SBR Operands: The SBR instruction must be the first instruction in a relay ladder or structured text routine.



Relay Ladder

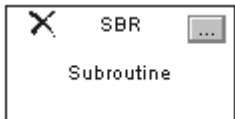
Operand:	Type:	Format:	Description:
Input	BOOL	tag	tag in this routine into which you want to copy the corresponding input parameter from the JSR instruction
parameter	SINT	array tag	
	INT		
	DINT		
	REAL		
	structure		



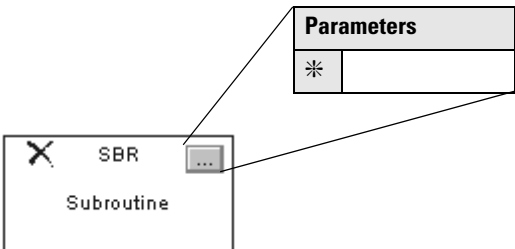
SBR (InputPar) ;

Structured Text

The operands are the same as those for the relay ladder SBR instruction.

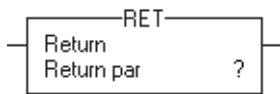


Function Block



The operands are the same as those for the relay ladder SBR instruction.

RET Operands:



Relay Ladder

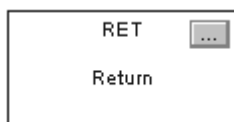
Operand:	Type:	Format:	Description:
Return parameter	BOOL SINT INT DINT REAL structure	immediate tag array tag	data from this routine that you want to copy to the corresponding return parameter in the JSR instruction



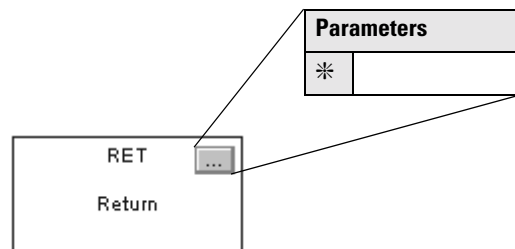
```
RET (ReturnPar) ;
```

Structured Text

The operands are the same as those for the relay ladder RET instruction.



Function Block



The operands are the same as those for the relay ladder RET instruction.

Description: The JSR instruction initiates the execution of the specified routine, which is referred to as a subroutine:

- The subroutine executes one time.
- After the subroutine executes, logic execution returns to the routine that contains the JSR instruction.

To program a jump to a subroutine, follow these guidelines:

IMPORTANT

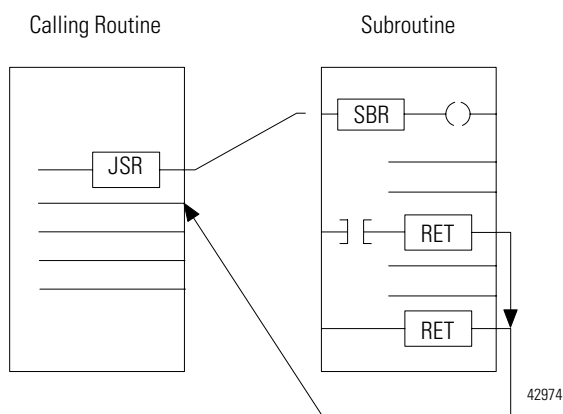
Do not use a JSR instruction to call (execute) the main routine.

- You can put a JSR instruction in the main routine or any other routine.
- If you use a JSR instruction to call the main routine and then put a RET instruction in the main routine, a major fault occurs (type 4, code 31).

The following diagram illustrates how the instructions operate.

JSR

1. If you want to copy data to a tag in the subroutine, enter an input parameter.
2. If you want to copy a result of the subroutine to a tag in this routine, enter a return parameter.
3. Enter as many input and return parameters as you need.



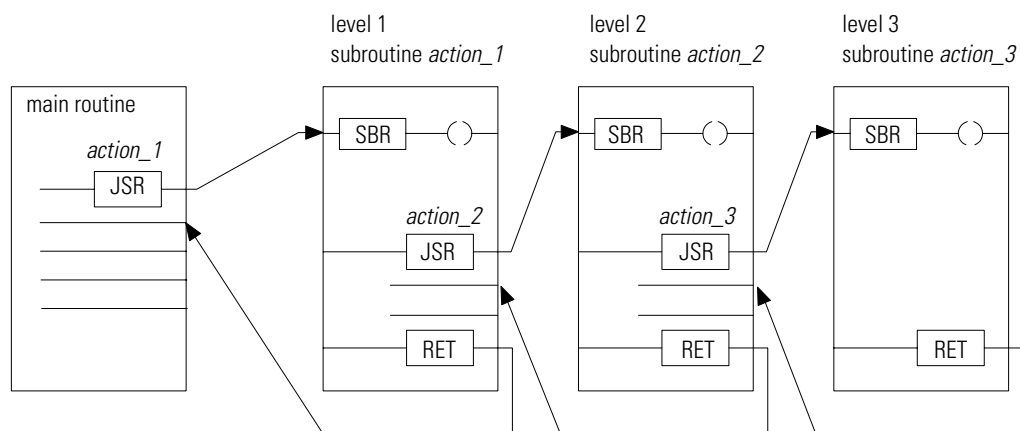
SBR

1. If the JSR instruction has an input parameter, enter an SBR instruction.
2. Place the SBR instruction as the first instruction in the routine.
3. For each input parameter in the JSR instruction, enter the tag into which you want to copy the data.

RET

1. If the JSR instruction has a return parameter, enter an RET instruction.
2. Place the RET instruction as the last instruction in the routine.
3. For each return parameter in the JSR instruction, enter a return parameter to send to the JSR instruction.
4. In a ladder routine, place additional RET instructions to exit the subroutine based on different input conditions, if required. (Function block routines only permit one RET instruction.)

There are no restrictions, other than controller memory, on the number of nested routines you can have or the number of parameters you pass or return.



Arithmetic Status Flags: Arithmetic status flags are affected.

Fault Conditions:

A major fault will occur if:	Fault type:	Fault code:
JSR instruction has fewer input parameters than SBR instruction	4	31
JSR instruction jumps to a fault routine	4 or user-supplied	0 or user-supplied
RET instruction has fewer return parameters than JSR instruction	4	31
main routine contains a RET instruction	4	31

Execution:

Relay Ladder and Structured Text



Condition:	Relay Ladder Action:	Structured Text Action:
prescan	<p>The controller executes all subroutines regardless of rung condition. To ensure that all rungs in the subroutine are prescanned, the controller ignores RET instructions. (I.e., RET instructions do not exit the subroutine.)</p> <ul style="list-style-type: none"> Release 6.x and earlier, input and return parameters are passed. Release 7.x and later, input and return parameters are not passed. <p>If recursive calls exist to the same subroutine, the subroutine is prescanned only the first time. If multiple calls exist (non-recursive) to the same subroutine, the subroutine is prescanned each time. The rung-condition-out is set to false (relay ladder only).</p>	
rung-condition-in is false to the JSR instruction	<p>The subroutine <i>does not</i> execute. Outputs in the subroutine remain in their last state. The rung-condition-out is set to false.</p>	na
rung-condition-in is true	<p>The instruction executes. The rung-condition-out is set to true.</p>	na
EnableIn is set	na	<p>EnableIn is always set. The instruction executes.</p>

Condition:	Relay Ladder Action:	Structured Text Action:
instruction execution	<pre>graph TD A{input parameters} -- yes --> B[JSR copies input parameters to appropriate SBR tags] A -- no --> C[logic execution begins in routine identified by JSR] B --> C C --> D{RET instruction} D -- yes --> E{return parameters} D -- no --> F{end of subroutine} E -- yes --> G[RET copies return parameters to appropriate JSR tags] E -- no --> F G --> F F -- yes --> H[rung-condition-out is set to false continue executing routine] F -- no --> I[rung-condition-out is set to true logic execution returns to JSR] E -- yes --> I I --> J([end])</pre>	
postscan	Same action as prescan described above.	Same action as prescan described above.



Function Block

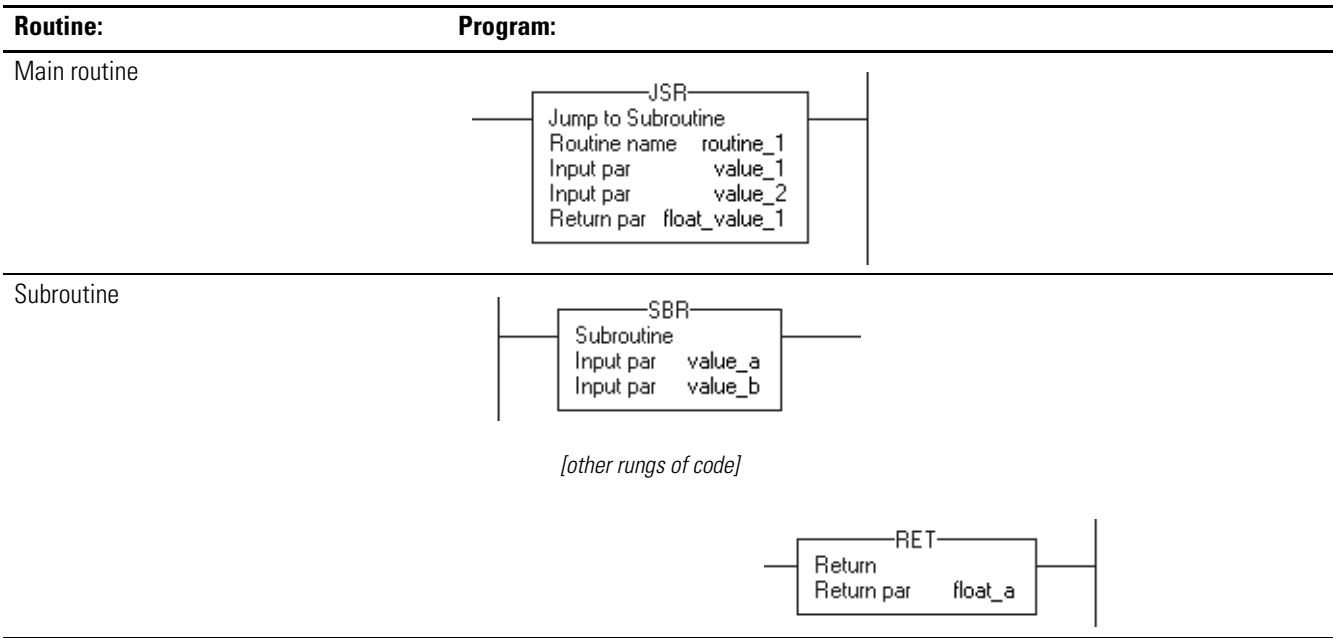
Condition:	Action:
prescan	No action taken.
instruction first scan	No action taken.
instruction first run	No action taken.
normal execution	<ol style="list-style-type: none">1. If the routine contains an SBR instruction, the controller first executes the SBR instruction.2. The controller latches all data values in IREFs.3. The controller executes the other function blocks in the order that is determined by their wiring. This includes other JSR instructions.4. The controller writes outputs in OREFs.5. If the routine contains an RET instruction, the controller executes the RET instruction last.
postscan	The subroutine is called. If the routine is an SFC routine, the routine is initialized the same as it is during prescan.

Example 1: The JSR instruction passes *value_1* and *value_2* to *routine_1*.

The SBR instruction receives *value_1* and *value_2* from the JSR instruction and copies those values to *value_a* and *value_b*, respectively. Logic execution continues in this routine.

The RET instruction sends *float_a* to the JSR instruction. The JSR instruction receives *float_a* and copies the value to *float_value_1*. Logic execution continues with the next instruction following the JSR instruction.

Relay Ladder



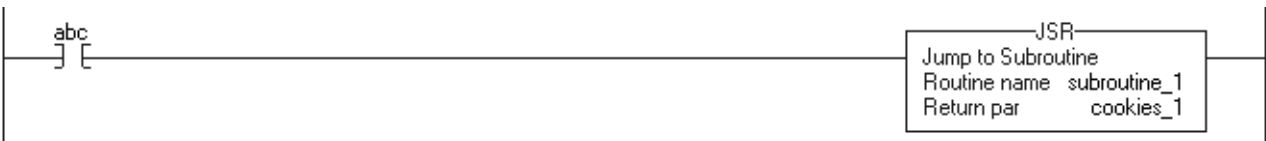
Structured Text

Routine:	Program:
Main routine	JSR(routine_1,2,value_1,value_2,float_value_1);
Subroutine	SBR(value_a,value_b); <statements>; RET(float_a);

Example 2:
Relay Ladder

MainRoutine

When *abc* is on, *subroutine_1* executes, calculates the number of cookies, and places a value in *cookies_1*.

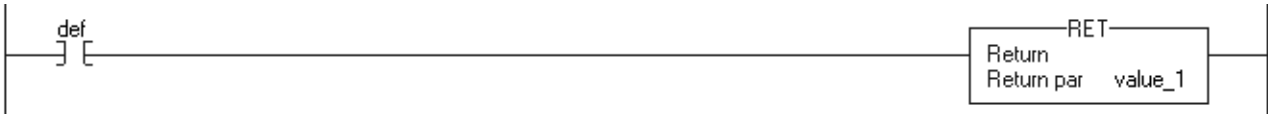


Adds the value in *cookies_1* to *cookies_2* and stores the result in *total_cookies*.

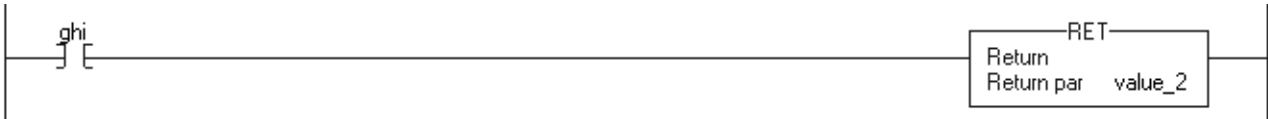


Subroutine_1

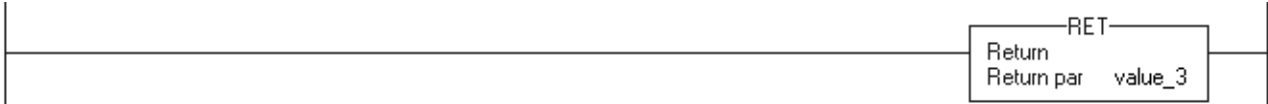
When *def* is on, the RET instruction returns *value_1* to the JSR *cookies_1* parameter and the rest of the subroutine is not scanned.

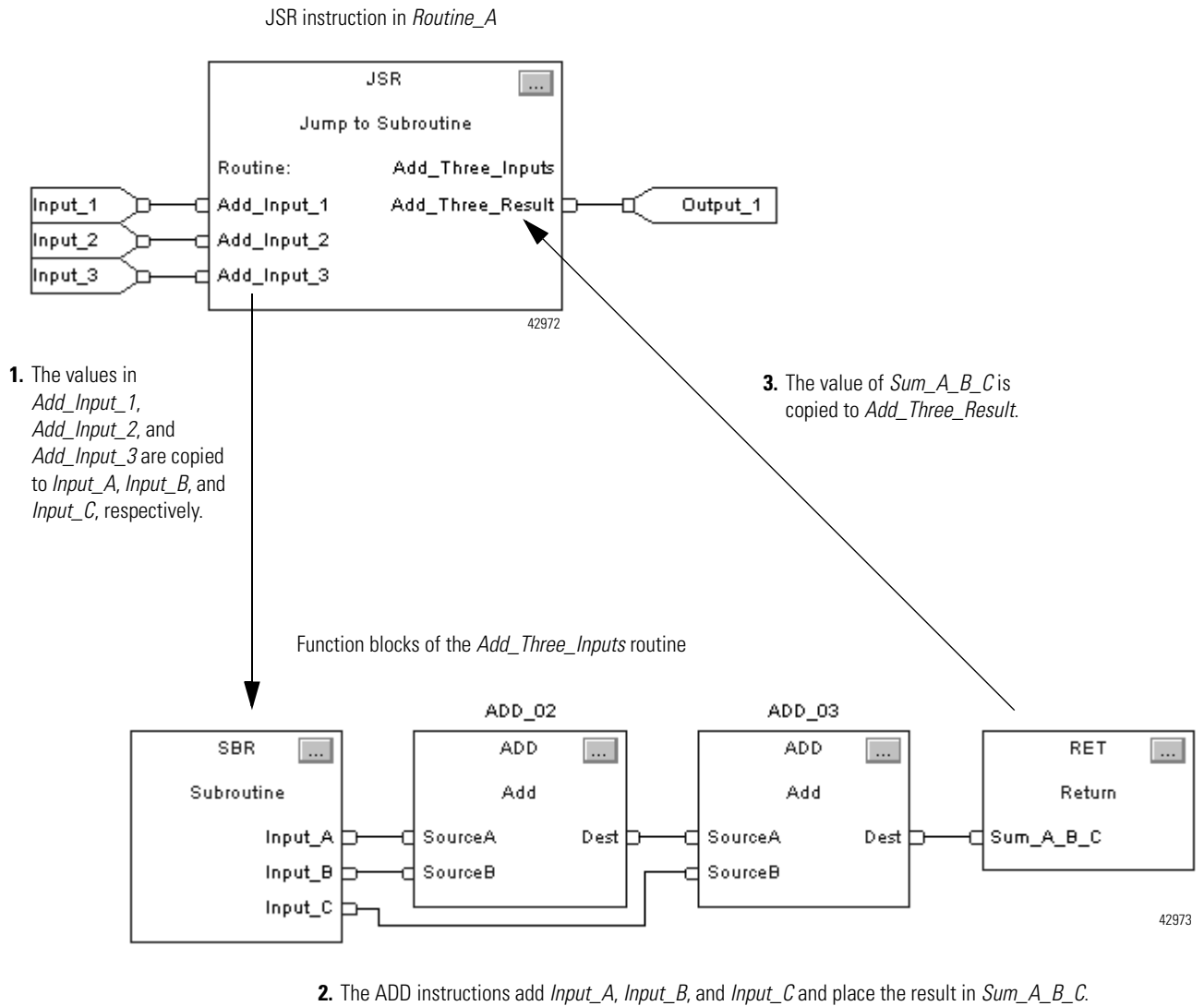


When *def* is off (previous rung) and *ghi* is on, the RET instruction returns *value_2* to the JSR *cookies_1* parameter and the rest of the subroutine is not scanned.



When both *def* and *ghi* are off (previous rungs), the RET instruction returns *value_3* to the JSR *cookies_1* parameter.



Example 3:**Function Block**

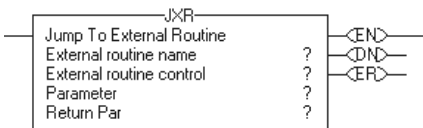
Jump to External Routine (JXR)

The JXR instruction executes an external routine. This instruction is only supported by the SoftLogix5800 controllers.

Operands:



Relay Ladder



Operand:	Type:	Format:	Description:
External routine name	ROUTINE	name	external routine to execute
External routine control	EXT_ROUTINE_CONTROL	tag	control structure (see the next page)
Parameter	BOOL SINT INT DINT REAL structure	immediate tag array tag	data from this routine that you want to copy to a variable in the external routine <ul style="list-style-type: none">Parameters are optional.Enter multiple parameters, if needed.You can have as many as 10 parameters.
Return parameter	BOOL SINT INT DINT REAL	tag	tag in this routine to which you want to copy a result of the external routine <ul style="list-style-type: none">The return parameter is optional.You can have only one return parameter

EXT_ROUTINE_CONTROL Structure

Mnemonic:	Data Type:	Description:	Implementation:
ErrorCode	SINT	If an error occurs, this value identifies the error. Valid values are from 0-255.	There are no predefined error codes. The developer of the external routine must provide the error codes.
NumParams	SINT	This value indicates the number of parameters associated with this instruction.	Display only - this information is derived from the instruction entry.
ParameterDefs	EXT_ROUTINE_PARAMETERS[10]	This array contains definitions of the parameters to pass to the external routine. The instruction can pass as many as 10 parameters.	Display only - this information is derived from the instruction entry.
ReturnParamDef	EXT_ROUTINE_PARAMETERS	This value contains definitions of the return parameter from the external routine. There is only one return parameter.	Display only - this information is derived from the instruction entry.
EN	BOOL	When set, the enable bit indicates that the JXR instruction is enabled.	The external routine sets this bit.
ReturnsValue	BOOL	If set, this bit indicates that a return parameter was entered for the instruction. If cleared, this bit indicates that no return parameter was entered for the instruction.	Display only - this information is derived from the instruction entry.
DN	BOOL	The done bit is set when the external routine has executed once to completion.	The external routine sets this bit.
ER	BOOL	The error bit is set if an error occurs. The instruction stops executing until the program clears the error bit.	The external routine sets this bit.
FirstScan	BOOL	This bit identifies whether this is the first scan after switching the controller to Run mode. Use FirstScan to initialize the external routine, if needed.	The controller sets this bit to reflect scan status.
EnableOut	BOOL	Enable output.	The external routine sets this bit.
EnableIn	BOOL	Enable input.	The controller sets this bit to reflect rung-condition-in. The instruction executes regardless of rung condition. The developer of the external routine should monitor this status and act accordingly.
User1	BOOL	These bits are available for the user. The controller does not initialize these bits.	Either the external routine or the user program can set these bits.
User0	BOOL		
ScanType1	BOOL	These bits identify the current scan type: Bit Values: Scan Type: 00 Normal 01 Pre Scan 10 Post Scan (not applicable to relay ladder programs)	The controller sets these bits to reflect scan status.
ScanType0	BOOL		

Description: Use the Jump to External Routine (JXR) instruction to call the external routine from a ladder routine in your project. The JXR instruction supports multiple parameters so you can pass values between the ladder routine and the external routine.

The JXR instruction is similar to the Jump to Subroutine (JSR) instruction. The JXR instruction initiates the execution of the specified external routine:

- The external routine executes one time.
- After the external routine executes, logic execution returns to the routine that contains the JXR instruction.

Arithmetic Status Flags: Arithmetic status flags are not affected.

Fault Conditions:

A major fault will occur if:	Fault type:	Fault code:
<ul style="list-style-type: none"> • an exception occurs in the external routine DLL • the DLL could not be loaded • the entry point was not found in the DLL 	4	88

Execution: The JXR can be synchronous or asynchronous depending on the implementation of the DLL. The code in the DLL also determines how to respond to scan status, rung-condition-in status, and rung-condition-out status.

For more information on using the JXR instruction and creating external routines, see the *SoftLogix5800 System User Manual*, publication 1789-UM002.

Temporary End (TND)

The TND instruction acts as a boundary.

Operands:



—(TND)—

Relay Ladder Operands

none



TND () ;

Structured Text

none

You must enter the parentheses () after the instruction mnemonic, even though there are no operands.

Description: When enabled, the TND instruction lets the controller execute logic only up to this instruction.

When enabled, the TND instruction acts as the end of the routine. When the controller scans a TND instruction, the controller moves to the end of the current routine. If the TND instruction is in a subroutine, control returns to the calling routine. If the TND instruction is in a main routine, control returns to the next program within the current task.

Arithmetic Status Flags: not affected

Fault Conditions: none

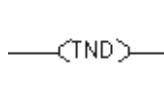
Execution:

Condition:	Relay Ladder Action:	Structured Text Action:
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	na
EnableIn is set	na	EnableIn is always set. The instruction executes.
instruction execution	The current routine terminates.	The current routine terminates.
postscan	The rung-condition-out is set to false.	No action taken.

Example: You can use the TND instruction when debugging or troubleshooting to execute logic up to a certain point. Progressively move the TND instruction through the logic as you debug each new section.

When the TND instruction is enabled, the controller stops scanning the current routine.

Relay Ladder



Structured Text

```
TND ( ) ;
```

Master Control Reset (MCR)

The MCR instruction, used in pairs, creates a program zone that can disable all rungs within the MCR instructions.

Operands:



—(MCR)—

Relay Ladder

none

Description: When the MCR zone is enabled, the rungs in the MCR zone are scanned for normal true or false conditions. When disabled, the controller still scans rungs within an MCR zone, but scan time is reduced because non-retentive outputs in the zone are disabled. The rung-condition-in is false for all the instructions inside of the disabled MCR zone.

When you program an MCR zone, note that:

- You must end the zone with an unconditional MCR instruction.
- You cannot nest one MCR zone within another.
- Do not jump into an MCR zone. If the zone is false, jumping into the zone activates the zone from the point to which you jumped to the end of the zone.
- If an MCR zone continues to the end of the routine, you do not have to program an MCR instruction to end the zone.

The MCR instruction is not a substitute for a hard-wired master control relay that provides emergency-stop capability. You should still install a hard-wired master control relay to provide emergency I/O power shutdown.

ATTENTION



Do not overlap or nest MCR zones. Each MCR zone must be separate and complete. If they overlap or nest, unpredictable machine operation could occur with possible damage to equipment or injury to personnel.

Place critical operations outside the MCR zone. If you start instructions such as timers in a MCR zone, instruction execution stops when the zone is disabled and the timer is cleared.

Arithmetic Status Flags: not affected

Fault Conditions: none

Execution:

Condition:	Relay Ladder Action:
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false. The instructions in the zone are scanned, but the rung-condition-in is false and non-retentive outputs in the zone are disabled.
rung-condition-in is true	The rung-condition-out is set to true. The instructions in the zone are scanned normally.
postscan	The rung-condition-out is set to false.

Example: When the first MCR instruction is enabled (*input_1*, *input_2*, and *input_3* are set), the controller executes the rungs in the MCR zone (between the two MCR instructions) and sets or clears outputs, depending on input conditions.

When the first MCR instruction is disabled (*input_1*, *input_2*, and *input_3* are not all set), the controller executes the rungs in the MCR zone (between the two MCR instructions) and the rung-condition-in goes false for all the rungs in the MCR zone, regardless of input conditions.



User Interrupt Disable (UID) User Interrupt Enable (UIE)

The UID instruction and the UIE instruction work together to prevent a small number of critical rungs from being interrupted by other tasks.

Operands:



—UID> <UIE—

Relay Ladder

none



```
UID ( ) ;  
UIE ( ) ;
```

Structured Text

none

You must enter the parentheses () after the instruction mnemonic, even though there are no operands.

Description: When the rung-condition-in is true, the:

- UID instruction prevents higher-priority tasks from interrupting the current task but *does not* disable execution of a fault routine or the Controller Fault Handler.
- UIE instruction enables other tasks to interrupt the current task.

To prevent a series of rungs from being interrupted:

1. Limit the number of rungs that you *do not* want interrupted to as few as possible. Disabling interrupts for a prolonged period of time can produce communication loss.
2. Above the first rung that you *do not* want interrupted, enter a rung and a UID instruction.
3. After the last rung in the series that you *do not* want interrupted, enter a rung and a UIE instruction.
4. If required, you can nest pairs of UID/UIE instructions.

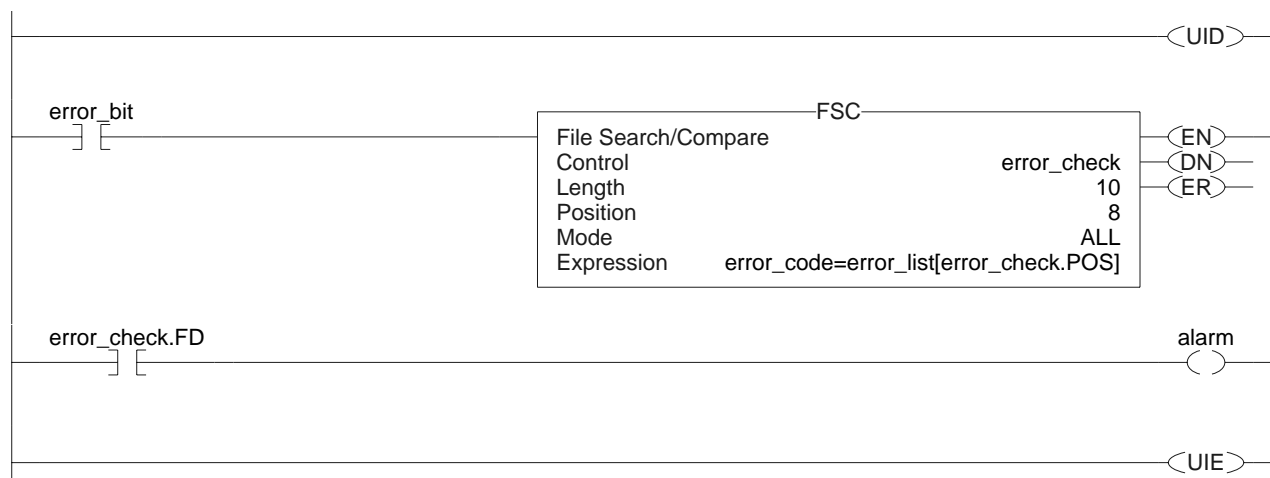
Arithmetic Status Flags: not affected

Fault Conditions: none

Execution:

Condition:	Relay Ladder Action:	Structured Text Action:
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	na
EnableIn is set	na	EnableIn is always set. The instruction executes.
instruction execution	The UID instruction prevents interruption by higher-priority tasks. The UIE instruction enables interruption by higher-priority tasks.	
postscan	The rung-condition-out is set to false.	No action taken.

Example: When an error occurs (*error_bit* is on), the FSC instruction checks the error code against a list of critical errors. If the FSC instruction finds that the error is critical (*error_check.FD* is on), an alarm is annunciated. The UID and UIE instructions prevent any other tasks from interrupting the error checking and alarming.

Relay Ladder**Structured Text**

```

UID ( ) ;

    <statements>

UIE ( ) ;

```

Always False Instruction (AFI)

The AFI instruction sets its rung-condition-out to false.

Operands:



—[AFI]—

Relay Ladder

none

Description: The AFI instruction sets its rung-condition-out to false.

Arithmetic Status Flags: not affected

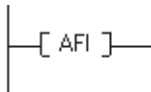
Fault Conditions: none

Execution:

Condition:	Relay Ladder Action:
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	The rung-condition-out is set to false.
postscan	The rung-condition-out is set to false.

Example: Use the AFI instruction to temporarily disable a rung while you are debugging a program.

When enabled, the AFI disables all the instructions on this rung.



No Operation (NOP)

The NOP instruction functions as a placeholder

Operands:



—[NOP]—

Relay Ladder

none

Description: You can place the NOP instruction anywhere on a rung. When enabled the NOP instruction performs no operation. When disabled, the NOP instruction performs no operation.

Arithmetic Status Flags: not affected

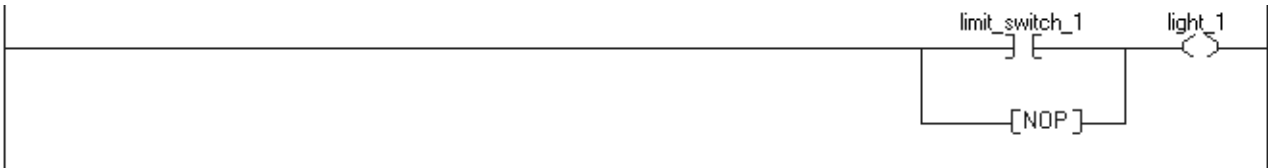
Fault Conditions: none

Execution:

Condition:	Relay Ladder Action:
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	The rung-condition-out is set to true.
postscan	The rung-condition-out is set to false.

Example: This instruction is useful for locating unconditional branches when you place the NOP instruction on the branch.

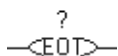
The NOP instruction bypasses the XIC instruction to enable the output.



End of Transition (EOT)

The EOT instruction returns a boolean state to an SFC transition.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
data bit	BOOL	tag	state of the transition (0=executing, 1=completed)



EOT(data_bit);

Structured Text

The operands are the same as those for the relay ladder EOT instruction.

Description: Because the EOT instruction returns a boolean state, multiple SFC routines can share the same routine that contains the EOT instruction. If the calling routine is not a transition, the EOT instruction acts as a TND instruction (see page 10-17).

The Logix implementation of the EOT instruction differs from that in a PLC-5 controller. In a PLC-5 controller, the EOT instruction has no parameters. Instead, the PLC-5 EOT instruction returns rung condition as its state. In a Logix controller, the return parameter returns the transition state since rung condition is not available in all Logix programming languages.

Arithmetic Status Flags: not affected

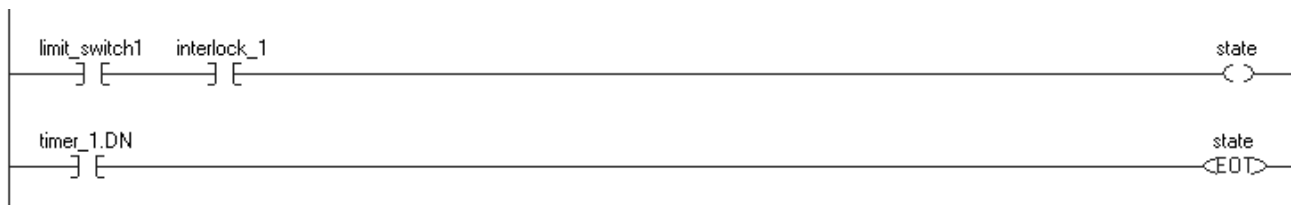
Fault Conditions: none

Execution:

Condition:	Relay Ladder Action:	Structured Text Action:
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	na
EnableIn is set	na	EnableIn is always set. The instruction executes.
instruction execution	The instruction returns the data bit value to the calling routine.	
postscan	The rung-condition-out is set to false.	No action taken.

Example: When both *limit_switch1* and *interlock_1* are set, set state. After *timer_1* completes, EOT returns the value of *state* to the calling routine.

Relay Ladder



Structured Text

```
state := limit_switch1 AND interlock_1;
```

```
IF timer_1.DN THEN
```

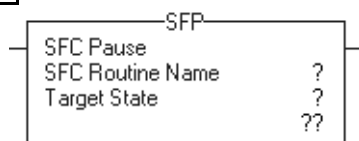
```
    EOT(state);
```

```
END_IF;
```

SFC Pause (SFP)

The SFP instruction pauses an SFC routine.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
SFCRoutine Name	ROUTINE	name	SFC routine to pause
TargetState	DINT	immediate tag	select one: executing (or enter 0) paused (or enter 1)



```
SFP(SFCRoutineName,  
    TargetState);
```

Structured Text

The operands are the same as those for the relay ladder SFP instruction.

Description: The SFP instruction lets you pause an executing SFC routine. If an SFC routine is in the paused state, use the SFP instruction again to change the state and resume execution of the routine.

Also, use the SFP instruction to resume SFC execution after using an SFR instruction (see page 10-29) to reset an SFC routine.

Arithmetic Status Flags: not affected

Fault Conditions:

A major fault will occur if:	Fault type:	Fault code:
the routine type is not an SFC routine	4	85

Execution:

Condition:	Relay Ladder Action:	Structured Text Action:
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	na
EnableIn is set	na	EnableIn is always set. The instruction executes.
instruction execution	The instruction pauses or resumes execution of the specified SFC routine.	
postscan	The rung-condition-out is set to false.	No action taken.

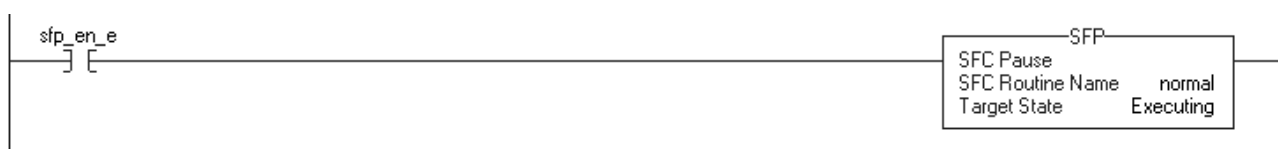
Example: If *sfc_en_p* is set, pause the SFC routine named *normal*. Restart the SFC when *sfc_en_e* is set.

Relay Ladder

Pause the SFC routine.



Resume executing the SFC routine.



Structured Text

```

Pause the SFC routine: IF (sfp_en_p) THEN
    SFP(normal, paused);
    sfp_en_p := 0;
END_IF;

```

```

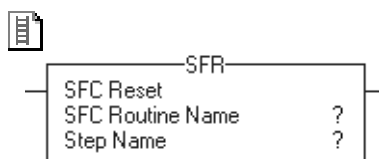
Resume executing the SFC routine: IF (sfp_en_e) THEN
    SFP(normal, executing);
    sfp_en_e := 0;
END_IF;

```


SFC Reset (SFR)

The SFR instruction resets the execution of a SFC routine at a specified step.

Operands:



Relay Ladder Operands

Operand:	Type:	Format:	Description:
SFCRoutine Name	ROUTINE	name	SFC routine to reset
Step Name	SFC_STEP	tag	target step where to resume execution



SFR (SFCRoutineName, StepName) ;

Structured Text

The operands are the same as those for the relay ladder SFR instruction.

Description: When the SFR instruction is enabled:

- In the specified SFC routine, all stored actions stop executing (reset).
- The SFC begins executing at the specified step.

If the target step is 0, the chart will be reset to its initial step

The Logix implementation of the SFR instruction differs from that in a PLC-5 controller. In the PLC-5 controller, the SFR executed when the rung condition was true. After reset, the SFC would remain paused until the rung containing the SFR became false. This allowed the execution following a reset to be delayed. This pause/un-pause feature of the PLC-5 SFR instruction was decoupled from the rung condition and moved into the SFP instruction.

Arithmetic Status Flags: not affected

Fault Conditions:

A major fault will occur if:	Fault type:	Fault code:
the routine type is not an SFC routine	4	85
specified target step does not exist in the SFC routine	4	89

Execution:

Condition:	Relay Ladder Action:	Structured Text Action:
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	na
EnableIn is set	na	EnableIn is always set. The instruction executes.
instruction execution	The instruction resets the specified SFC routine.	The instruction resets the specified SFC routine.
postscan	The rung-condition-out is set to false.	No action taken.

Example: If a specific condition occurs (*shutdown* is set), restart the SFC at step *initialize*.

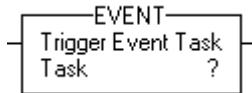
Relay Ladder**Structured Text**

```

IF shutdown THEN
    SFR(mySFC, initialize);
END_IF;
  
```

Trigger Event Task (EVENT) The EVENT instruction triggers one execution of an event task.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Task	TASK	name	event task to execute The instruction lets you choose other types of tasks, but it does not execute them.



```
EVENT ( task_name ) ;
```

Structured Text

The operands are the same as those for the relay ladder EVENT instruction.

Description: Use the EVENT instruction to programmatically execute an event task:

- Each time the instruction executes, it triggers the specified event task.
- Make sure that you give the event task enough time to complete its execution before you trigger it again. If not, an overlap occurs.
- If you execute an EVENT instruction while the event task is already executing, the controller increments the overlap counter but it does not trigger the event task.

Programmatically Determine if an EVENT Instruction Triggered a Task

To determine if an EVENT instruction triggered an event task, use a Get System Value (GSV) instruction to monitor the Status attribute of the task.

Table 10.1 Status Attribute of the TASK Object

Attribute:	Data Type:	Instruction:	Description:								
Status	DINT	GSV SSV	Provides status information about the task. Once the controller sets a bit, you must manually clear the bit to determine if another fault of that type occurred.								
			<table><tr><th>To determine if:</th><th>Examine this bit:</th></tr><tr><td>An EVENT instruction triggered the task (event task only).</td><td>0</td></tr><tr><td>A timeout triggered the task (event task only).</td><td>1</td></tr><tr><td>An overlap occurred for this task.</td><td>2</td></tr></table>	To determine if:	Examine this bit:	An EVENT instruction triggered the task (event task only).	0	A timeout triggered the task (event task only).	1	An overlap occurred for this task.	2
To determine if:	Examine this bit:										
An EVENT instruction triggered the task (event task only).	0										
A timeout triggered the task (event task only).	1										
An overlap occurred for this task.	2										

The controller does not clear the bits of the Status attribute once they are set.

- To use a bit for new status information, you must manually clear the bit.
- Use a Set System Value (SSV) instruction to set the attribute to a different value.

Arithmetic Status Flags: not affected

Fault Conditions: none

Execution:

Condition:	Relay Ladder Action:	Structured Text Action:
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes.	na
	The rung-condition-out is set to true.	
EnableIn is set	na	EnableIn is always set.
		The instruction executes.
instruction execution	The instruction triggers one execution of the specified event task	
postscan	The rung-condition-out is set to false.	No action taken.

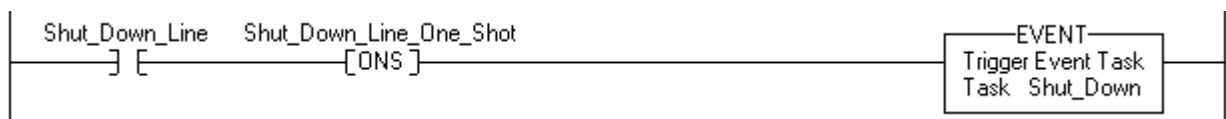
Example 1: A controller uses multiple programs but a common shut down procedure. Each program uses a program-scoped tag named *Shut_Down_Line* that turns on if the program detects a condition that requires a shut down. The logic in each program executes as follows:

If *Shut_Down_Line* = on (conditions require a shut down) then

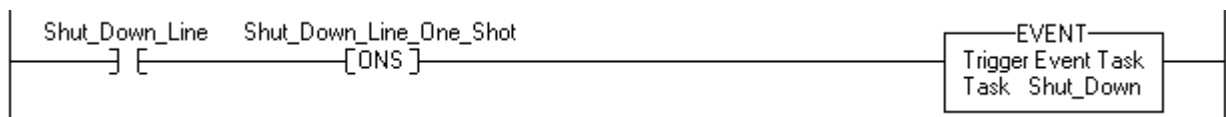
Execute the *Shut_Down* task one time

Relay Ladder

Program A



Program B



Structured Text

Program A

```
IF Shut_Down_Line AND NOT Shut_Down_Line_One_Shot THEN
    EVENT (Shut_Down);
END_IF;
Shut_Down_Line_One_Shot := Shut_Down_Line;
```

Program B

```
IF Shut_Down_Line AND NOT Shut_Down_Line_One_Shot THEN
    EVENT (Shut_Down);
END_IF;
Shut_Down_Line_One_Shot := Shut_Down_Line;
```

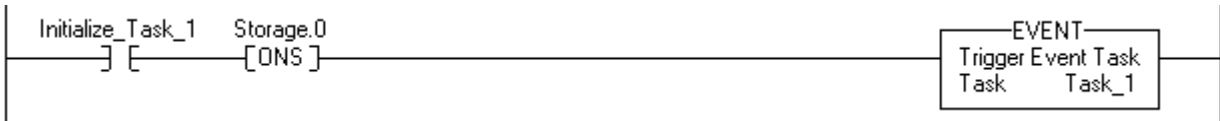
Example 2: The following example uses an EVENT instruction to initialize an event task. (Another type of event normally triggers the event task.)

Continuous task

If *Initialize_Task_1* = 1 then

The ONS instruction limits the execution of the EVENT instruction to one scan.

The EVENT instruction triggers an execution of *Task_1* (event task).



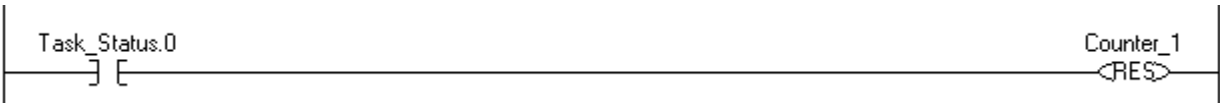
Task_1 (event task)

The GSV instruction sets *Task_Status* (DINT tag) = Status attribute for the event task. In the Instance Name attribute, THIS means the TASK object for the task that the instruction is in (i.e., *Task_1*).



If *Task_Status.0* = 1 then an EVENT instruction triggered the event task (i.e., when the continuous task executes its EVENT instruction to initialize the event task).

The RES instruction resets a counter that the event task uses.

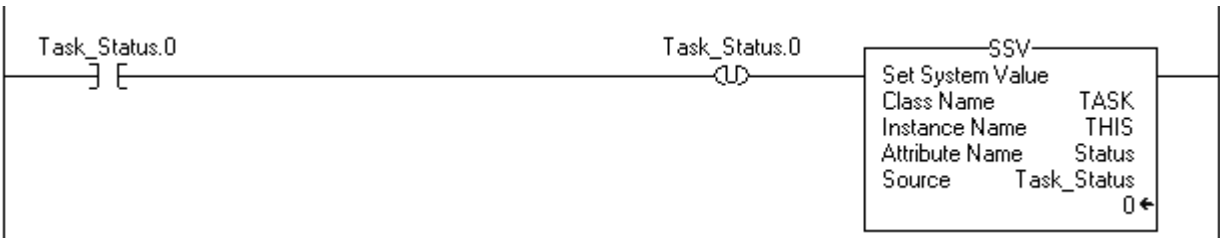


The controller does not clear the bits of the Status attribute once they are set. To use a bit for new status information, you must manually clear the bit.

If *Task_Status.0* = 1 then clear that bit.

The OTU instruction sets *Task_Status.0* = 0.

The SSV instruction sets the Status attribute of THIS task (*Task_1*) = *Task_Status*. This includes the cleared bit.



For/Break Instructions

(FOR, FOR...DO, BRK, EXIT, RET)

Introduction

Use the FOR instruction to repeatedly call a subroutine. Use the BRK instruction to interrupt the execution of a subroutine.

If you want to:	Use this instruction:	Available in these languages:	See page:
Repeatedly execute a routine.	FOR FOR...DO ⁽¹⁾	relay ladder structured text	11-2
Terminate the repeated execution of a routine.	BRK EXIT ⁽¹⁾	relay ladder structured text	11-5
Return to the FOR instruction.	RET	relay ladder	11-6

⁽¹⁾ Structured text only.

For (FOR)

The FOR instruction executes a routine repeatedly.

Operands:



FOR	
For	
Routine name	?
Index	?
	??
Initial value	?
Terminal value	?
Step size	?

Relay Ladder

Operand:	Type:	Format:	Description:
Routine name	ROUTINE	routine name	routine to execute
Index	DINT	tag	counts how many times the routine has been executed
Initial value	SINT INT DINT	immediate tag	value at which to start the index
Terminal value	SINT INT DINT	immediate tag	value at which to stop executing the routine
Step size	SINT INT DINT	immediate tag	amount to add to the index each time the FOR instruction executes the routine



```
FOR count:= initial_value TO
final_value BY increment DO
    <statement>;
END_FOR;
```

Structured Text

Use the FOR...DO construct. See Appendix C for information on structured text constructs.

Description:

IMPORTANT

Do not use a FOR instruction to call (execute) the main routine.

- You can put a FOR instruction in the main routine or any other routine.
- If you use a FOR instruction to call the main routine and then put a RET instruction in the main routine, a major fault occurs (type 4, code 31).

When enabled, the FOR instruction repeatedly executes the Routine until the Index value exceeds the Terminal value. This instruction does not pass parameters to the routine.

Each time the FOR instruction executes the routine, it adds the Step size to the Index.

Be careful not to loop too many times in a single scan. An excessive number of repetitions can cause the controller's watchdog to timeout, which causes a major fault.

Arithmetic Status Flags: not affected

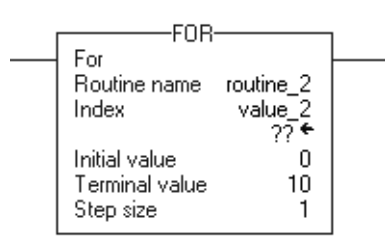
Fault Conditions:

A major fault will occur if:	Fault type:	Fault code:
main routine contains a RET instruction	4	31

Execution:

Condition:	Relay Ladder Action:
prescan	The rung-condition-out is set to false. The controller executes the subroutine once. If recursive FOR instruction0s exist to the same subroutine, the subroutine is prescanned only the first time. If multiple FOR instructions exist (non-recursive) to the same subroutine, the subroutine is prescanned each time.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	<pre> graph TD Start([]) --> Init[index = initial_value] Init --> StepSize{step size < 0} StepSize -- no --> IndexLeq{index <= terminal value} StepSize -- yes --> IndexGeq{index >= terminal value} IndexLeq -- no --> GotoEnd1((goto end)) IndexLeq -- yes --> ExecuteRoutine[execute routine index=(index + step_size)] ExecuteRoutine --> Init IndexGeq -- yes --> ExecuteRoutine IndexGeq -- no --> GotoEnd2((goto end)) GotoEnd1 --> SetRungOut[rung-condition-out is set to true] GotoEnd2 --> SetRungOut GotoEnd3((end)) --> SetRungOut SetRungOut --> End([end]) </pre>
postscan	The rung-condition-out is set to false.

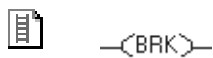
Example: When enabled, the FOR instruction repeatedly executes *routine_2* and increments *value_2* by 1 each time. When *value_2* is > 10 or a BRK instruction is enabled, the FOR instruction no longer executes *routine_2*.



Break (BRK)

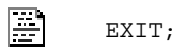
The BRK instruction interrupts the execution of a routine that was called by a FOR instruction.

Operands:



Relay Ladder

none



Structured Text

Use the EXIT statement in a loop construct. See Structured Text Programming for information on structured text constructs.

Description: When enabled, the BRK instruction exits the routine and returns the controller to the instruction that follows the FOR.

If there are nested FOR instructions, a BRK instruction returns control to the innermost FOR instruction.

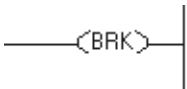
Arithmetic Status Flags: not affected

Fault Conditions: none

Execution:

Condition:	Relay Ladder Action:
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	The rung-condition-out is set to true. Execution returns to the instruction that follows the calling FOR instruction.
postscan	The rung-condition-out is set to false.

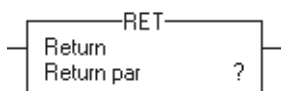
Example: When enabled, the BRK instruction stops executing the current routine and returns to the instruction that follows the calling FOR instruction.



Return (RET)

The RET instruction returns to the calling FOR instruction.

Operands:



Relay Ladder

none

Description:

IMPORTANT

Do not place a RET instruction in the main routine. If you place a RET instruction in the main routine, a major fault occurs (type 4, code 31).

When enabled, the RET instruction returns to the FOR instruction. The FOR instruction increments the Index value by the Step size and executes the subroutine again. If the Index value exceeds the Terminal value, the FOR instruction completes and execution moves on to the instruction that follows the FOR instruction.

The FOR instruction does not use parameters. The FOR instruction ignores any parameters you enter in a RET instruction.

You could also use a TND instruction to end execution of a subroutine.

Arithmetic Status Flags: not affected

Fault Conditions:

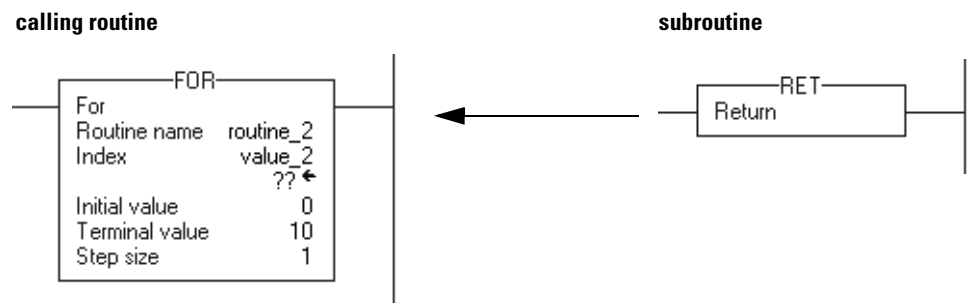
A major fault will occur if:	Fault type:	Fault code:
main routine contains a RET instruction	4	31

Execution:

Condition:	Relay Ladder Action:
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	Returns the specified parameters to the calling routine. The rung-condition-out is set to true.
postscan	The rung-condition-out is set to false.

Example: The FOR instruction repeatedly executes *routine_2* and increments *value_2* by 1 each time. When *value_2* is > 10 or a BRK instruction is enabled, the FOR instruction no longer executes *routine_2*.

The RET instruction returns to the calling FOR instruction. The FOR instruction either executes the subroutine again and increments the Index value by the Step size or, if the Index value exceeds the Terminal value, the FOR instruction is complete and execution moves on to the instruction that follows the FOR instruction.



Notes:

Special Instructions

(FBC, DDT, DTR, PID)

Introduction

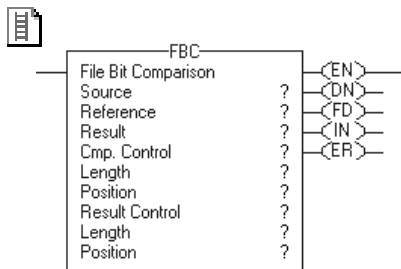
The special instructions perform application-specific operations.

If you want to:	Use this instruction:	Available in these languages:	See page:
Compare data against a known, good reference and record any mismatches.	FBC	relay ladder	12-2
Compare data against a known, good reference, record any mismatches, and update the reference to match the source.	DDT	relay ladder	12-10
Pass the source data through a mask and compare the result to reference data. Then write the source into the reference for the next comparison.	DTR	relay ladder	12-18
Control a PID loop.	PID	relay ladder structured text	12-21

File Bit Comparison (FBC)

The FBC instruction compares bits in a Source array with bits in a Reference array.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Source	DINT	array tag	array to compare to the reference do not use CONTROL.POS in the subscript
Reference	DINT	array tag	array to compare to the source do not use CONTROL.POS in the subscript
Result	DINT	array tag	array to store the result do not use CONTROL.POS in the subscripts
Cmp control	CONTROL	structure	control structure for the compare
Length	DINT	immediate	number of bits to compare
Position	DINT	immediate	current position in the source initial value is typically 0
Result control	CONTROL	structure	control structure for the results
Length	DINT	immediate	number of storage locations in the result
Position	DINT	immediate	current position in the result initial value is typically 0

ATTENTION



Use different tags for the compare control structure and the result control structure. Using the same tag for both could result in unpredictable operation, possibly causing equipment damage and/or injury to personnel.

COMPARE Structure

Mnemonic:	Data Type:	Description:
.EN	BOOL	The enable bit indicates that the FBC instruction is enabled.
.DN	BOOL	The done bit is set when the FBC instruction compares the last bit in the Source and Reference arrays.
.FD	BOOL	The found bit is set each time the FBC instruction records a mismatch (one-at-a-time operation) or after recording all mismatches (all-per-scan operation).
.IN	BOOL	The inhibit bit indicates the FBC search mode. 0 = all mode 1 = one mismatch at a time mode
.ER	BOOL	The error bit is set if the compare .POS < 0, the compare .LEN < 0, the result .POS < 0 or the result .LEN < 0. The instruction stops executing until the program clears the .ER bit.
.LEN	DINT	The length value identifies the number of bits to compare.
.POS	DINT	The position value identifies the current bit.

RESULT Structure

Mnemonic:	Data Type:	Description:
.DN	BOOL	The done bit is set when the Result array is full.
.LEN	DINT	The length value identifies the number of storage locations in the Result array.
.POS	DINT	The position value identifies the current position in the Result array.

Description: When enabled, the FBC instruction compares the bits in the Source array with the bits in the Reference array and records the bit number of each mismatch in the Result array.

The FBC instruction operates on contiguous memory.

The difference between the DDT and FBC instructions is that each time the DDT instruction finds a mismatch, the instruction changes the reference bit to match the source bit. The FBC instruction does not change the reference bit.

Selecting the search mode

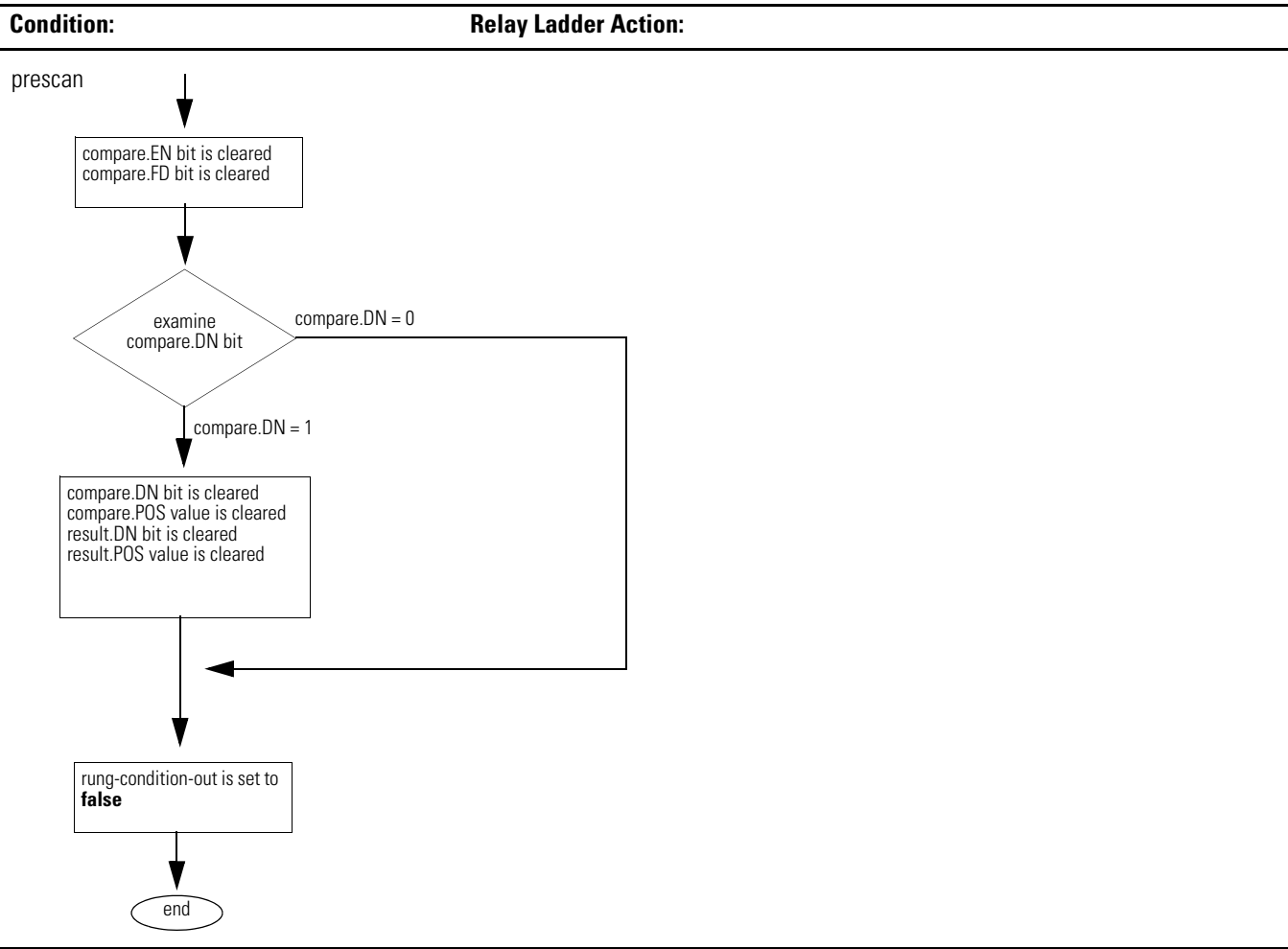
If you want to detect:	Select this mode:
One mismatch at a time	Set the .IN bit in the compare CONTROL structure. Each time the rung-condition-in goes from false to true, the FBC instruction searches for the next mismatch between the Source and Reference arrays. Upon finding a mismatch, the instruction sets the .FD bit, records the position of the mismatch, and stops executing.
All mismatches	Clear the .IN bit in the compare CONTROL structure. Each time the rung-condition-in goes from false to true, the FSC instruction searches for all mismatches between the Source and Reference arrays.

Arithmetic Status Flags: not affected

Fault Conditions:

A major fault will occur if:	Fault type:	Fault code:
Result.POS > size of Result array	4	20

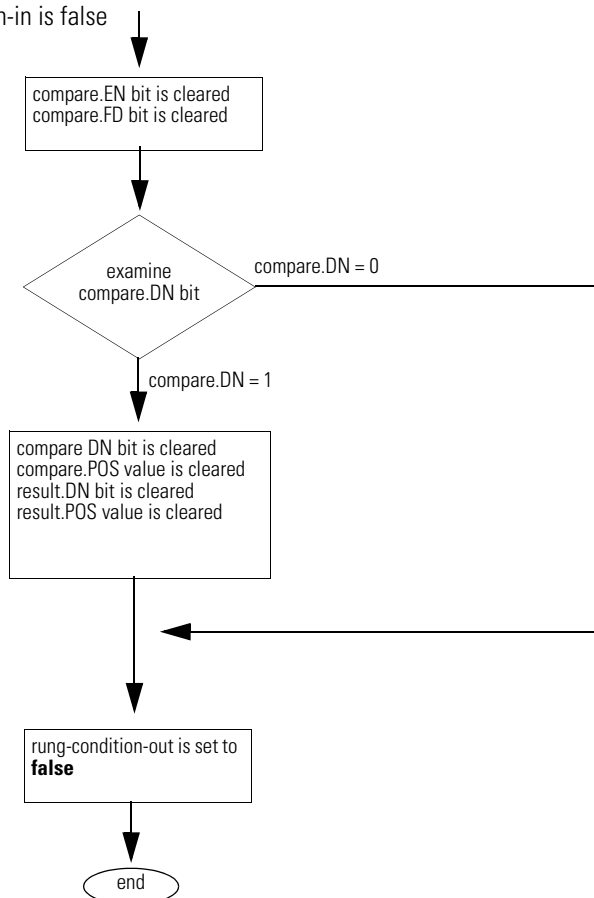
Execution:



Condition:

Relay Ladder Action:

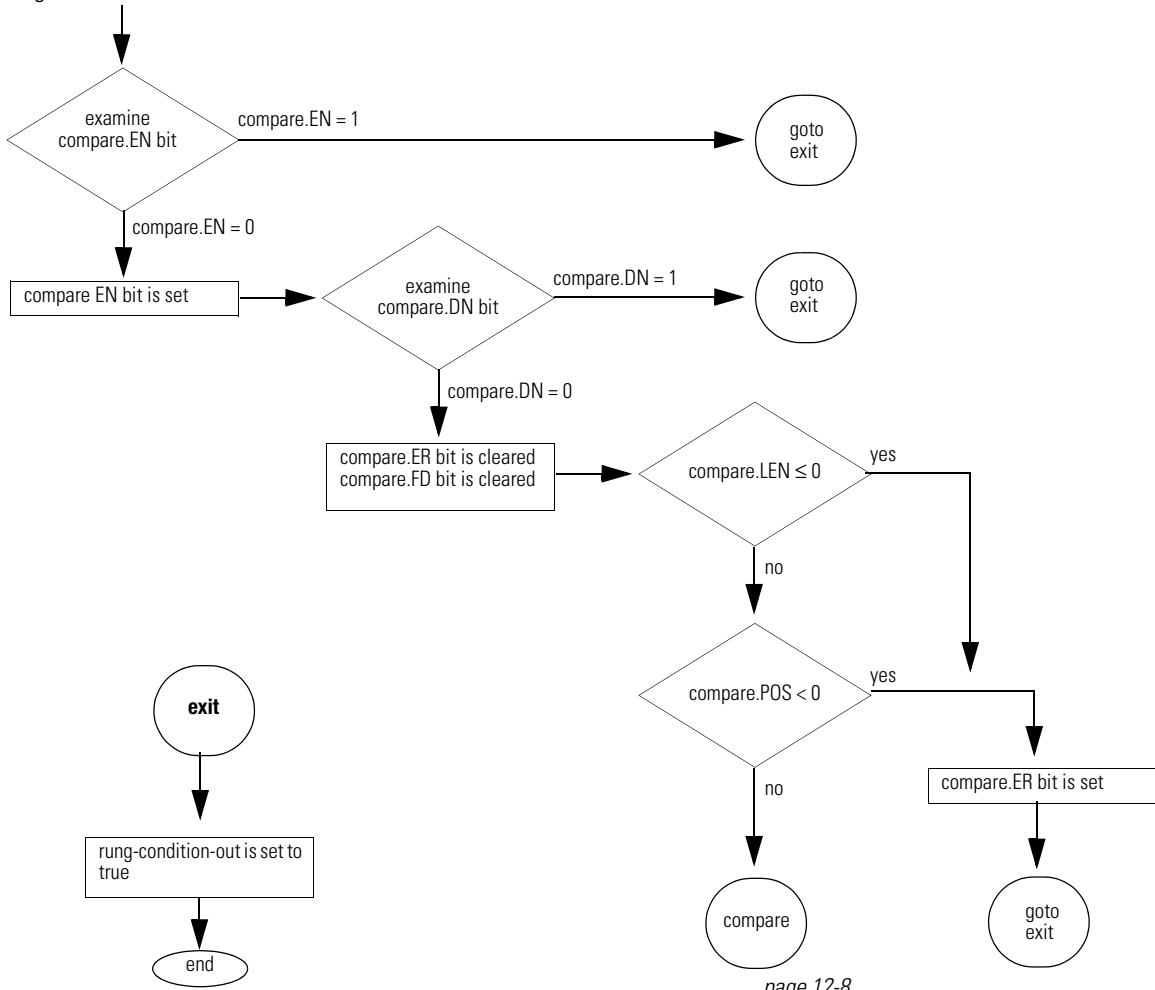
rung-condition-in is false

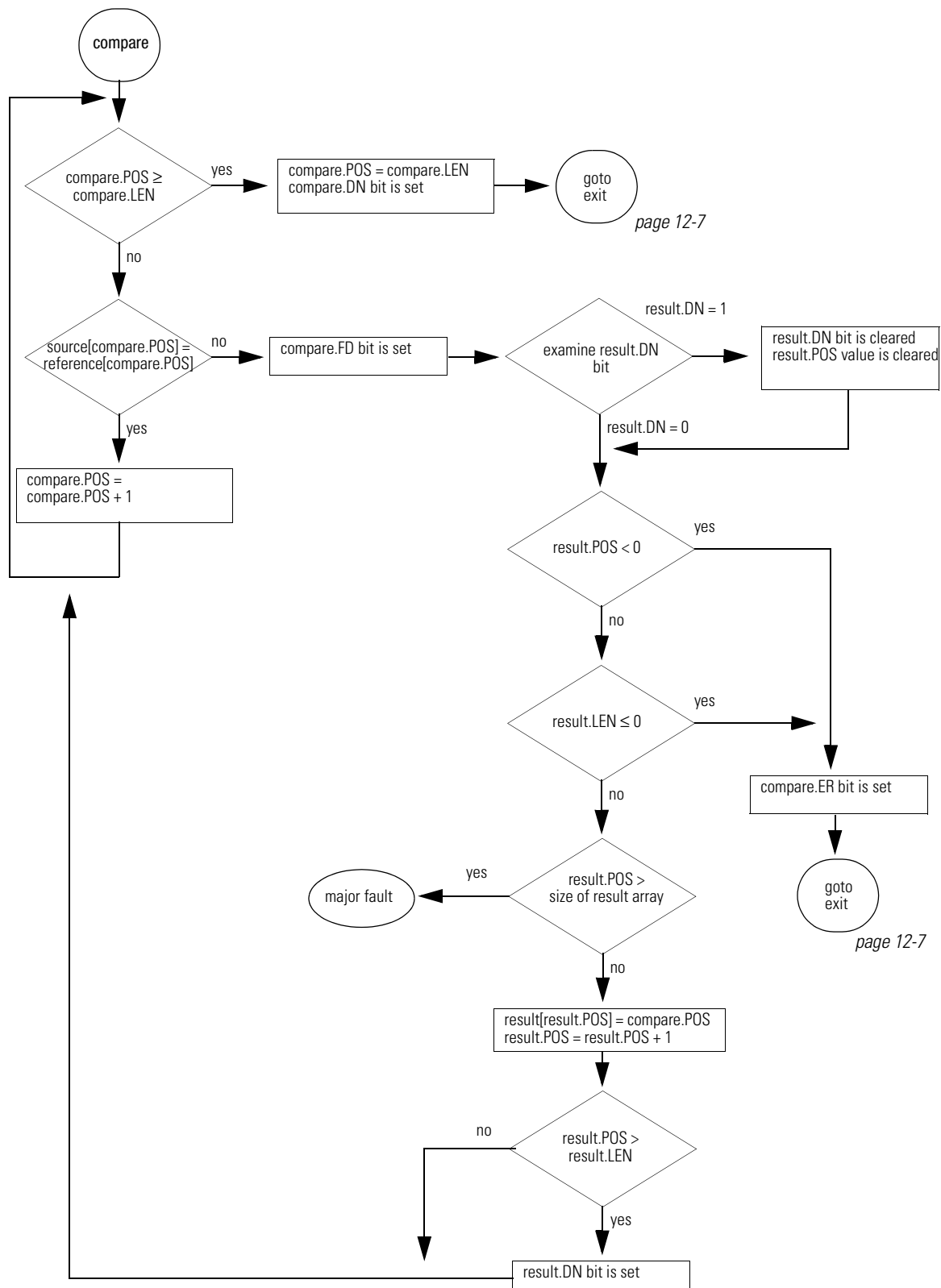


Condition:

Relay Ladder Action:

rung-condition-in is true

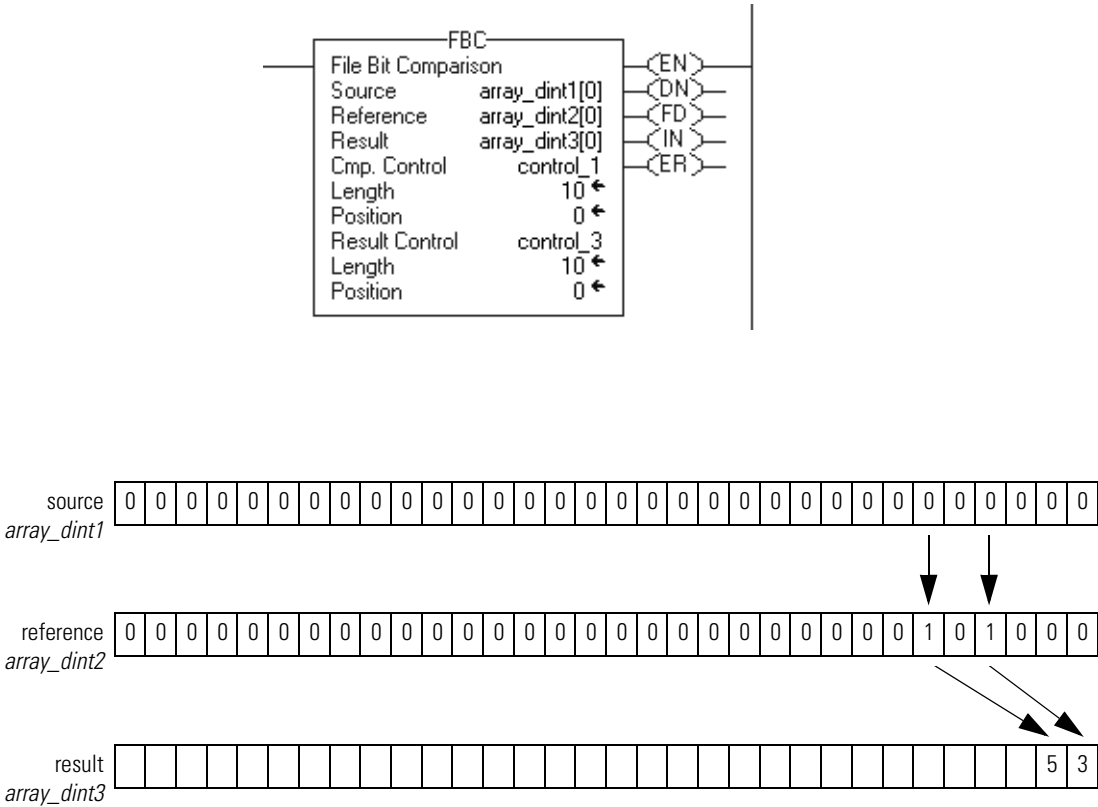


Condition:**Relay Ladder Action:**

postscan

The rung-condition-out is set to false.

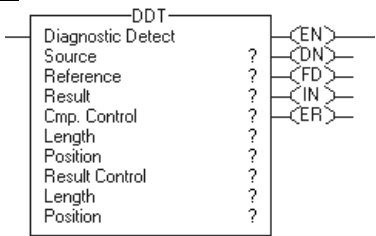
Example: When enabled, the FBC instruction compares the source *array_dint1* to the reference *array_dint2* and stores the locations of any mismatches in the result *array_dint3*.



Diagnostic Detect (DDT)

The DDT instruction compares bits in a Source array with bits in a Reference array to determine changes of state.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Source	DINT	array tag	array to compare to the reference do not use CONTROL.POS in the subscript
Reference	DINT	array tag	array to compare to the source do not use CONTROL.POS in the subscript
Result	DINT	array tag	array to store the results do not use CONTROL.POS in the subscript
Cmp control	CONTROL	structure	control structure for the compare
Length	DINT	immediate	number of bits to compare
Position	DINT	immediate	current position in the source initial value typically 0
Result control	CONTROL	structure	control structure for the results
Length	DINT	immediate	number of storage locations in the result
Position	DINT	immediate	current position in the result initial value typically 0

ATTENTION



Use different tags for the compare control structure and the result control structure. Using the same tag for both could result in unpredictable operation, possibly causing equipment damage and/or injury to personnel.

COMPARE Structure

Mnemonic:	Data Type:	Description:
.EN	BOOL	The enable bit indicates that the DDT instruction is enabled.
.DN	BOOL	The done bit is set when the DDT instruction compares the last bit in the Source and Reference arrays.
.FD	BOOL	The found bit is set each time the DDT instruction records a mismatch (one-at-a-time operation) or after recording all mismatches (all-per-scan operation).
.IN	BOOL	The inhibit bit indicates the DDT search mode. 0 = all mode 1 = one mismatch at a time mode
.ER	BOOL	The error bit is set if the compare .POS < 0, the compare .LEN < 0, the result .POS < 0 or the result .LEN < 0. The instruction stops executing until the program clears the .ER bit.
.LEN	DINT	The length value identifies the number of bits to compare.
.POS	DINT	The position value identifies the current bit.

RESULT Structure

Mnemonic:	Data Type:	Description:
.DN	BOOL	The done bit is set when the Result array is full.
.LEN	DINT	The length value identifies the number of storage locations in the Result array.
.POS	DINT	The position value identifies the current position in the Result array.

Description: When enabled, the DDT instruction compares the bits in the Source array with the bits in the Reference array, records the bit number of each mismatch in the Result array, and changes the value of the Reference bit to match the value of the corresponding Source bit.

The DDT instruction operates on contiguous memory.

The difference between the DDT and FBC instructions is that each time the DDT instruction finds a mismatch, the DDT instruction changes the reference bit to match the source bit. The FBC instruction does not change the reference bit.

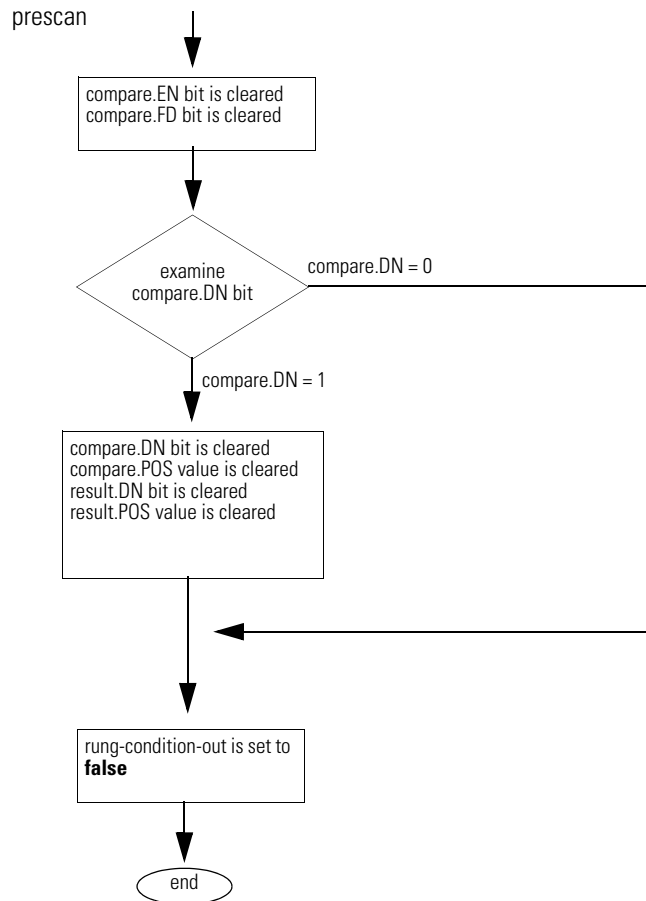
Selecting the search mode

If you want to detect:	Select this mode:
One mismatch at a time	Set the .IN bit in the compare CONTROL structure. Each time the rung-condition-in goes from false to true, the DDT instruction searches for the next mismatch between the Source and Reference arrays. Upon finding a mismatch, the instruction sets the .FD bit, records the position of the mismatch, and stops executing.
All mismatches	Clear the .IN bit in the compare CONTROL structure. Each time the rung-condition-in goes from false to true, the DDT instruction searches for all mismatches between the Source and Reference arrays.

Arithmetic Status Flags: not affected

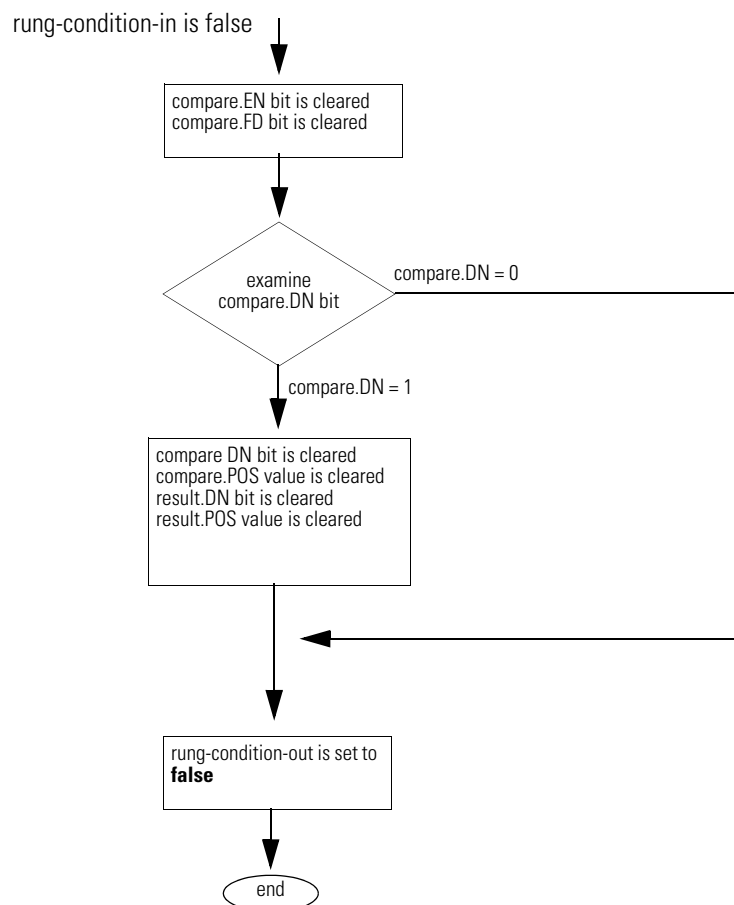
Fault Conditions:

A major fault will occur if:	Fault type:	Fault code:
Result.POS > size of Result array	4	20

Execution:**Condition:****Relay Ladder Action:**

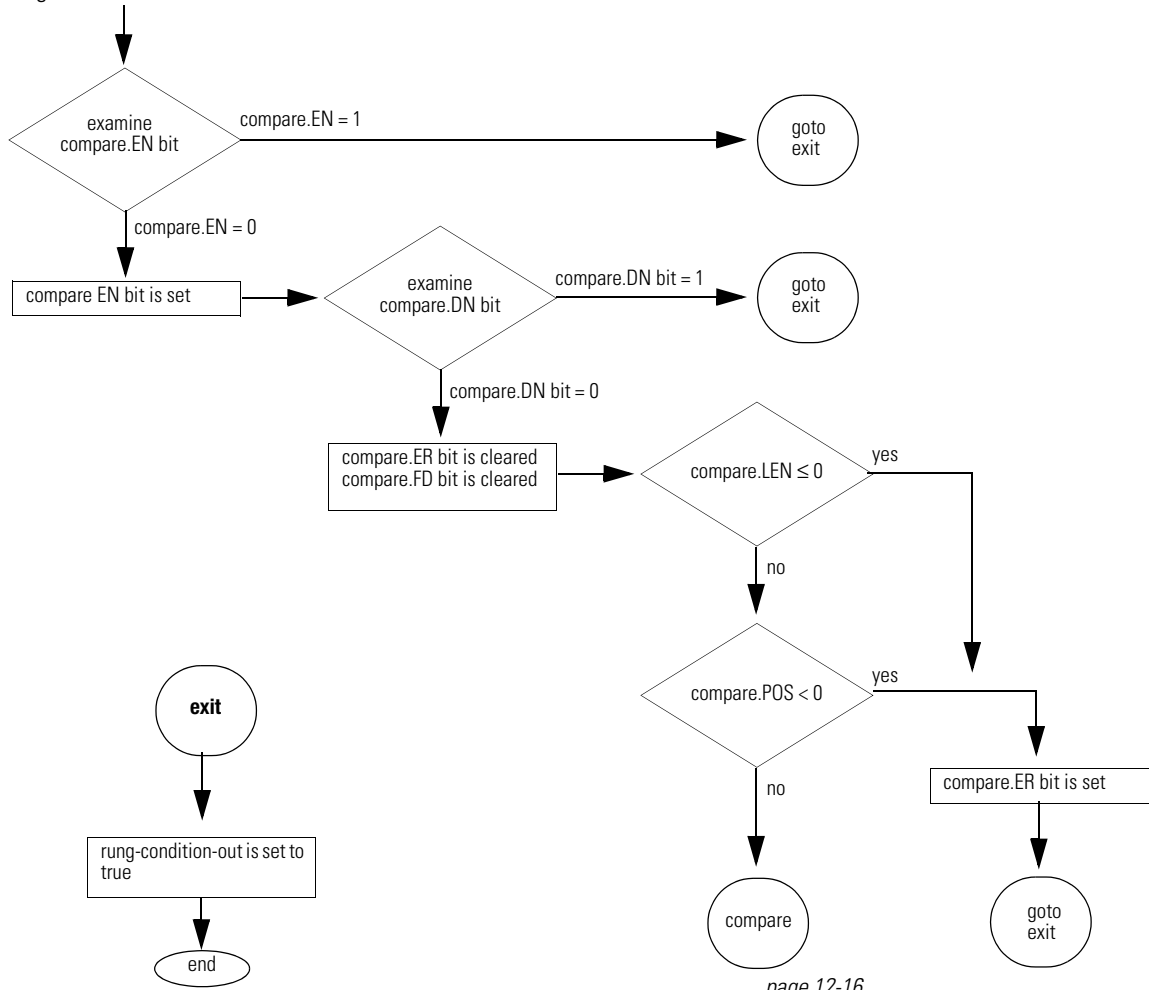
Condition:

Relay Ladder Action:

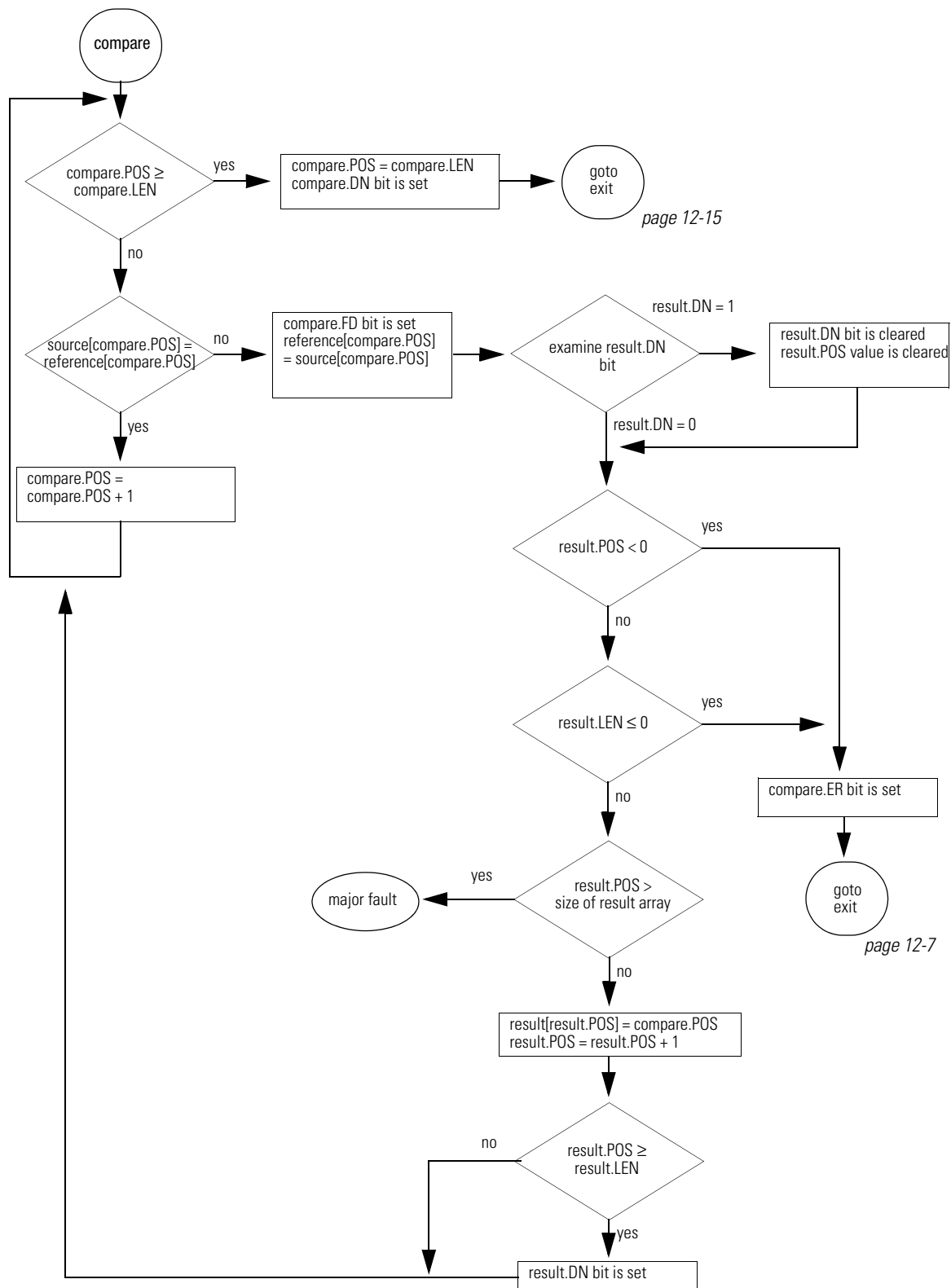


Condition:**Relay Ladder Action:**

rung-condition-in is true



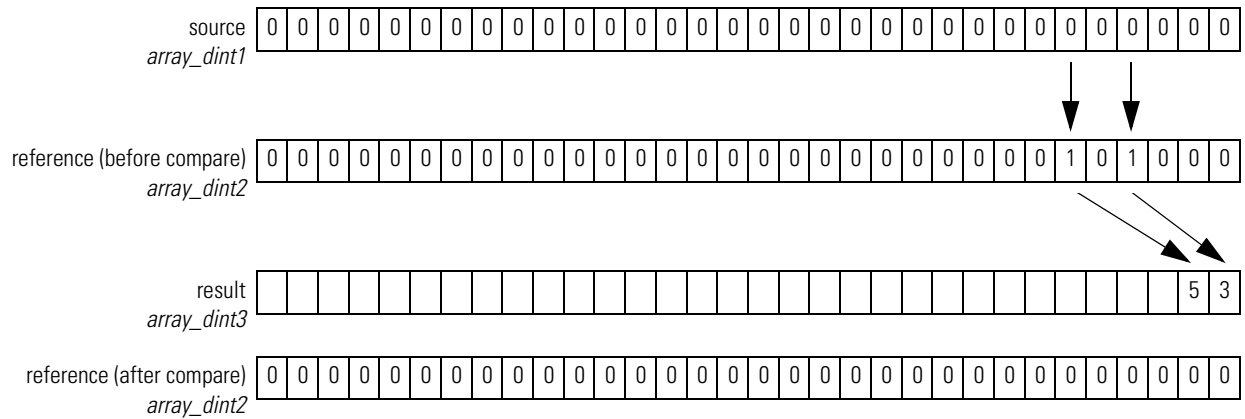
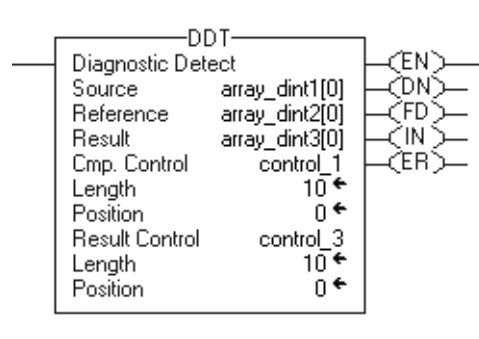
page 12-16

Condition:**Relay Ladder Action:**

postscan

The rung-condition-out is set to false.

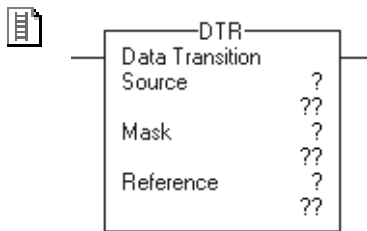
Example: When enabled, the DDT instruction compares the source *array_dint1* to the reference *array_dint2* and stores the locations of any mismatches in the result *array_dint3*. The controller also changes the mismatched bits in the reference *array_dint2* to match the source *array_dint1*.



Data Transitional (DTR)

The DTR instruction passes the Source value through a Mask and compares the result with the Reference value.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Source	DINT	immediate tag	array to compare to the reference
Mask	DINT	immediate tag	which bits to block or pass
Reference	DINT	tag	array to compare to the source

Description: The DTR instruction passes the Source value through a Mask and compares the result with the Reference value. The DTR instruction also writes the masked Source value into the Reference value for the next comparison. The Source remains unchanged.

A “1” in the mask means the data bit is passed. A “0” in the mask means the data bit is blocked.

When the masked Source differs from the Reference, the rung-condition-out goes true for one scan. When the masked Source is the same as the Reference, the rung-condition-out is false.

ATTENTION



Online programming with this instruction can be dangerous. If the Reference value is different than the Source value, the rung-condition-out goes true. Use caution if you insert this instruction when the processor is in Run or Remote Run mode.

Entering an immediate mask value

When you enter a mask, the programming software defaults to decimal values. If you want to enter a mask using another format, precede the value with the correct prefix.

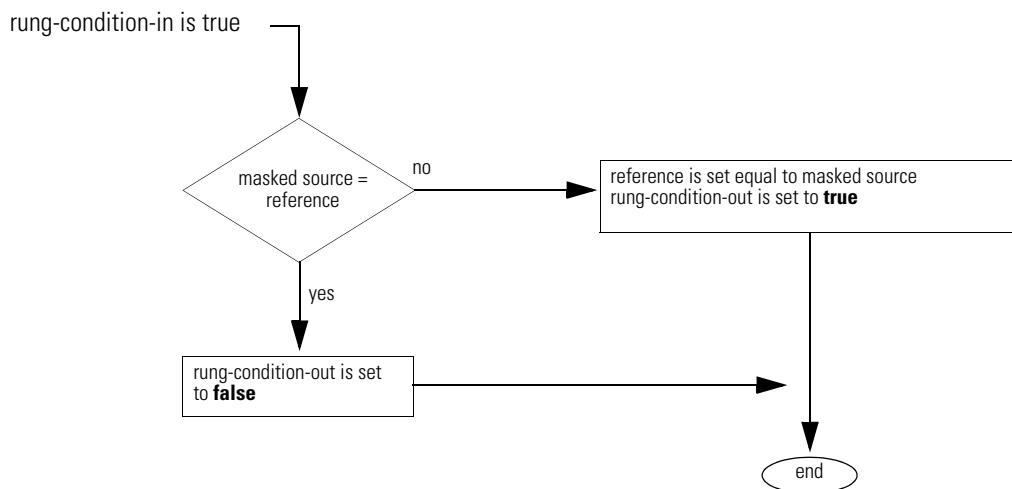
Prefix:	Description:
16#	hexadecimal for example; 16#0F0F
8#	octal for example; 8#16
2#	binary for example; 2#00110011

Arithmetic Status Flags: not affected

Fault Conditions: none

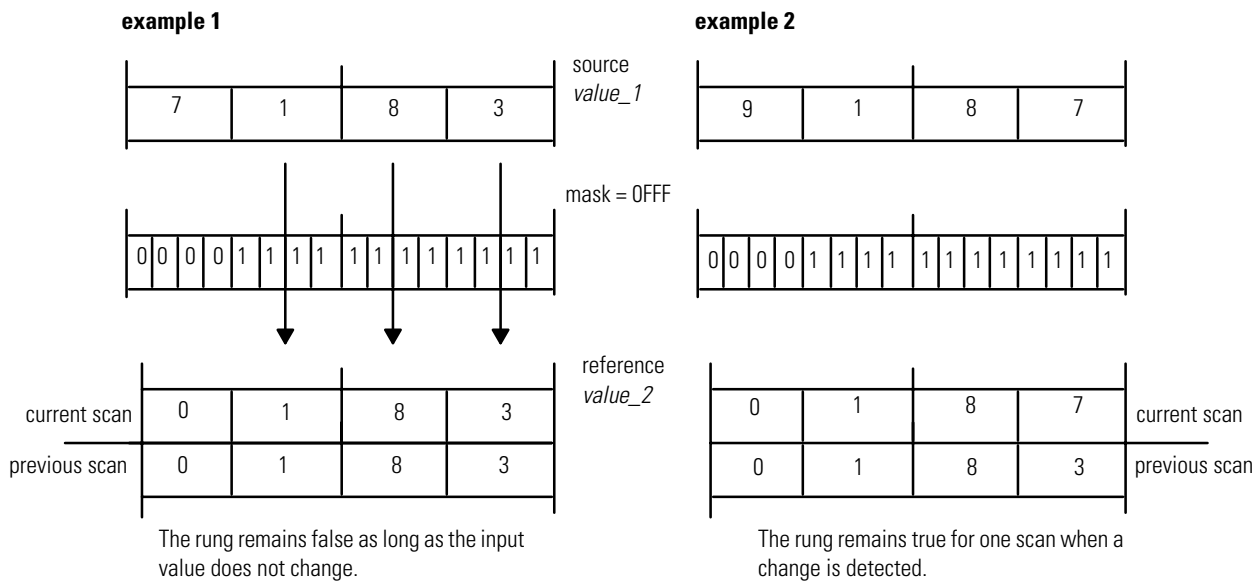
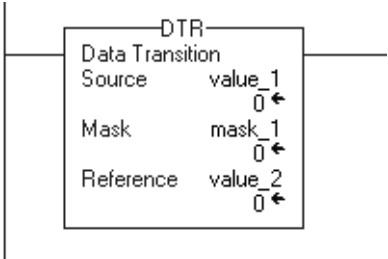
Execution:

Condition:	Relay Ladder Action:
prescan	The Reference = Source AND Mask. The rung-condition-out is set to false.
rung-condition-in is false	The Reference = Source AND Mask. The rung-condition-out is set to false.



postscan	The rung-condition-out is set to false.
----------	---

Example: When enabled, the DTR instruction masks *value_1*. If there is a difference in the two values, the rung-condition-out is set to true.



13385

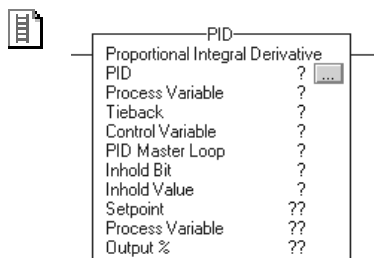
A 0 in the mask leaves the bit unchanged.

Proportional Integral Derivative (PID)

The PID instruction controls a process variable such as flow, pressure, temperature, or level.

Operands:

Relay Ladder



Operand:	Type:	Format:	Description:
PID	PID	structure	PID structure
Process variable	SINT INT DINT REAL	tag	value you want to control
Tieback	SINT INT DINT REAL	immediate tag	<i>(optional)</i> output of a hardware hand/auto station which is bypassing the output of the controller Enter 0 if you don't want to use this parameter.
Control variable	SINT INT DINT REAL	tag	value which goes to the final control device (valve, damper, etc.) If you are using the deadband, the Control variable must be REAL or it will be forced to 0 when the error is within the deadband.
PID master loop	PID	structure	<i>(optional)</i> PID tag for the master PID If you are performing cascade control and this PID is a slave loop, enter the name of the master PID. Enter 0 if you don't want to use this parameter.
Inhold bit	BOOL	tag	<i>(optional)</i> current status of the inhold bit from a 1756 analog output channel to support bumpless restart Enter 0 if you don't want to use this parameter.
Inhold value	SINT INT DINT REAL	tag	<i>(optional)</i> data readback value from a 1756 analog output channel to support bumpless restart Enter 0 if you don't want to use this parameter.
Setpoint			displays current value of the setpoint
Process variable			displays current value of the scaled Process Variable
Output %			displays current output percentage value



```
PID (PID, ProcessVariable,
Tieback, ControlVariable,
PIDMasterLoop, InholdBit,
InHoldValue);
```

Structured Text

The operands are the same as those for the relay ladder PID instruction. However, you specify the Setpoint, Process Variable, and Output % by accessing the .SP, .PV, and .OUT members of the PID structure, rather than by including values in the operand list.

PID Structure

Mnemonic:	Data Type:	Description:	
.CTL	DINT	The .CTL member provides access to the status members (bits) in one, 32-bit word. The PID instruction sets bits 07 -15.	
		This bit:	Is this member:
		31	.EN
		30	.CT
		29	.CL
		28	.PVT
		27	.DOE
		26	.SWM
		25	.CA
		24	.MO
		23	.PE
		22	.NDF
		21	.NOBC
		20	.NOZC
		This bit:	Is this member, which the PID instruction sets:
		15	.INI
		14	.SPOR
		13	.OLL
		12	.OLH
		11	.EWD
		10	.DVNA
		09	.DVPA
		08	.PVLA
		07	.PVHA
		.SP	REAL
.KP	REAL	independent	proportional gain (unitless)
		dependent	controller gain (unitless)
.KI	REAL	independent	integral gain (1/sec)
		dependent	reset time (minutes per repeat)
.KD	REAL	independent	derivative gain (seconds)
		dependent	rate time (minutes)
.BIAS	REAL	feedforward or bias %	
.MAXS	REAL	maximum engineering unit scaling value	
.MINS	REAL	minimum engineering unit scaling value	

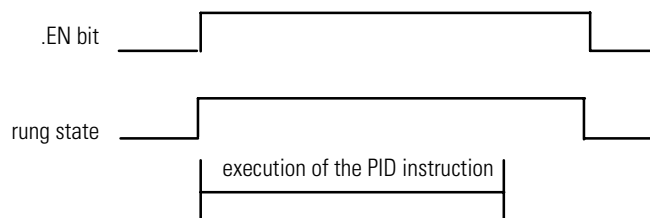
Mnemonic:	Data Type:	Description:
.DB	REAL	deadband engineering units
.SO	REAL	set output %
.MAXO	REAL	maximum output limit (% of output)
.MINO	REAL	minimum output limit (% of output)
.UPD	REAL	loop update time (seconds)
.PV	REAL	scaled PV value
.ERR	REAL	scaled error value
.OUT	REAL	output %
.PVH	REAL	process variable high alarm limit
.PVL	REAL	process variable low alarm limit
.DVP	REAL	positive deviation alarm limit
.DVN	REAL	negative deviation alarm limit
.PVDB	REAL	process variable alarm deadband
.DVDB	REAL	deviation alarm deadband
.MAXI	REAL	maximum PV value (unscaled input)
.MINI	REAL	minimum PV value (unscaled input)
.TIE	REAL	tieback value for manual control
.MAXCV	REAL	maximum CV value (corresponding to 100%)
.MINCV	REAL	minimum CV value (corresponding to 0%)
.MINTIE	REAL	minimum tieback value (corresponding to 100%)
.MAXTIE	REAL	maximum tieback value (corresponding to 0%)

Mnemonic:	Data Type:	Description:	
.DATA	REAL[17]	The .DATA member stores:	
		Element:	Description:
		.DATA[0]	integral accumulation
		.DATA[1]	derivative smoothing temporary value
		.DATA[2]	previous .PV value
		.DATA[3]	previous .ERR value
		.DATA[4]	previous valid .SP value
		.DATA[5]	percent scaling constant
		.DATA[6]	.PV scaling constant
		.DATA[7]	derivative scaling constant
		.DATA[8]	previous .KP value
		.DATA[9]	previous .KI value
		.DATA[10]	previous .KD value
		.DATA[11]	dependent gain .KP
		.DATA[12]	dependent gain .KI
		.DATA[13]	dependent gain .KD
		.DATA[14]	previous .CV value
		.DATA[15]	.CV descaling constant
.DATA[16]	tieback descaling constant		
.EN	BOOL	enabled	
.CT	BOOL	cascade type (0=slave; 1=master)	
.CL	BOOL	cascade loop (0=no; 1=yes)	
.PVT	BOOL	process variable tracking (0=no; 1=yes)	
.DOE	BOOL	derivative of (0=PV; 1=error)	
.SWM	BOOL	software manual mode (0=no-auto; 1=yes- sw manual)	
.CA	BOOL	control action (0 means E=SP-PV; 1 means E=PV-SP)	
.MO	BOOL	station mode (0=automatic; 1=manual)	
.PE	BOOL	PID equation (0=independent; 1=dependent)	
.NDF	BOOL	no derivative smoothing (0=derivative smoothing filter enabled; 1=derivative smoothing filter disabled)	
.NOBC	BOOL	no bias back calculation (0=bias back calculation enabled; 1=bias back calculation disabled)	
.NOZC	BOOL	no zero crossing deadband (0=deadband is zero crossing; 1=deadband is not zero crossing)	
.INI	BOOL	PID initialized (0=no; 1=yes)	
.SPOR	BOOL	setpoint out of range (0=no; 1=yes)	
.OLL	BOOL	CV is below minimum output limit (0=no; 1=yes)	
.OLH	BOOL	CV is above maximum output limit (0=no; 1=yes)	

Mnemonic:	Data Type:	Description:
.EWD	BOOL	error is within deadband (0=no; 1=yes)
.DVNA	BOOL	deviation is alarmed low (0=no; 1=yes)
.DVPA	BOOL	deviation is alarmed high (0=no; 1=yes)
.PVLA	BOOL	PV is alarmed low (0=no; 1=yes)
.PVHA	BOOL	PV is alarmed high (0=no; 1=yes)

Description: The PID instruction typically receives the process variable (PV) from an analog input module and modulates a control variable output (CV) on an analog output module in order to maintain the process variable at the desired setpoint.

The .EN bit indicates execution status. The .EN bit is set when the rung-condition-in transitions from false to true. The .EN bit is cleared when the rung-condition-in becomes false. The PID instruction does not use a .DN bit. The PID instruction executes every scan as long as the rung-condition-in is true.



Arithmetic Status Flags: not affected

Fault Conditions:

IMPORTANT

These faults were major faults in the PLC-5 controller.

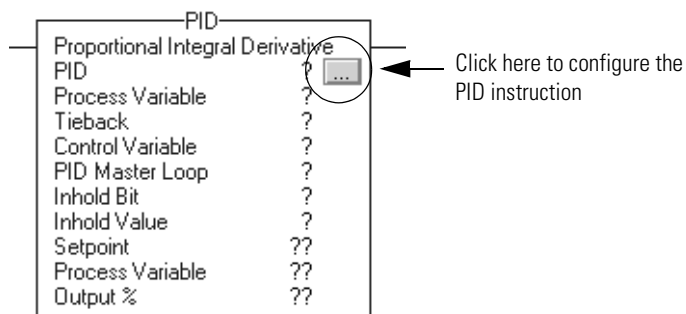
A minor fault will occur if:	Fault type:	Fault code:
.UPD \leq 0	4	35
setpoint out of range	4	36

Execution:

Condition:	Action:	Action:
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	na
EnableIn is set	na	EnableIn is always set. The instruction executes.
instruction execution	The instruction executes the PID loop.	The instruction executes the PID loop.
postscan	The rung-condition-out is set to false.	No action taken.

Configuring a PID Instruction

After you enter the PID instruction and specify the PID structure, you use the configuration tabs to specify how the PID instruction should function.



Specifying tuning

Select the Tuning tab. Changes take effect as soon as you click on another field, click OK, click Apply, or press Enter.

In this field:	Specify:
Setpoint (SP)	Enter a setpoint value (.SP).
Set output %	Enter a set output percentage (.SO). In software manual mode, this value is used for the output. In auto mode, this value displays the output %.
Output bias	Enter an output bias percentage (.BIAS).
Proportional gain (K_p)	Enter the proportional gain (.KP). For independent gains, it's the proportional gain (unitless). For dependent gains, it's the controller gain (unitless).
Integral gain (K_i)	Enter the integral gain (.KI). For independent gains, it's the integral gain (1/sec). For dependent gains, it's the reset time (minutes per repeat).
Derivative time (K_d)	Enter the derivative gain (.KD). For independent gains, it's the derivative gain (seconds). For dependent gains, it's the rate time minutes).
Manual mode	Select either manual (.MO) or software manual (.SWM). Manual mode overrides software manual mode if both are selected.

Specifying configuration

Select the Configuration tab. You must click OK or Apply for any changes to take effect.

In this field:	Specify:
PID equation	Select independent gains or dependent gains (.PE). Use independent when you want the three gains (P, I, and D) to operate independently. Use dependent when you want an overall controller gain that affects all three terms (P, I, and D).
Control action	Select either E=PV-SP or E=SP-PV for the control action (.CA).
Derivative of	Select PV or error (.DOE). Use the derivative of PV to eliminate output spikes resulting from setpoint changes. Use the derivative of error for fast responses to setpoint changes when the algorithm can tolerate overshoots.
Loop update time	Enter the update time (.UPD) for the instruction.
CV high limit	Enter a high limit for the control variable (.MAXO).
CV low limit	Enter a low limit for the control variable (.MINO).
Deadband value	Enter a deadband value (.DB).
No derivative smoothing	Enable or disable this selection (.NDF).

In this field:	Specify:
No bias calculation	Enable or disable this selection (.NOBC).
No zero crossing in deadband	Enable or disable this selection (.NOZC).
PV tracking	Enable or disable this selection (.PVT).
Cascade loop	Enable or disable this selection (.CL).
Cascade type	If cascade loop is enabled, select either slave or master (.CT).

Specifying alarms

Select the Alarms tab. You must click OK or Apply for any changes to take effect.

In this field:	Specify:
PV high	Enter a PV high alarm value (.PVH).
PV low	Enter a PV low alarm value (.PVL).
PV deadband	Enter a PV alarm deadband value (.PVDB).
positive deviation	Enter a positive deviation value (.DVP).
negative deviation	Enter a negative deviation value (.DVN).
deviation deadband	Enter a deviation alarm deadband value (.DVDB).

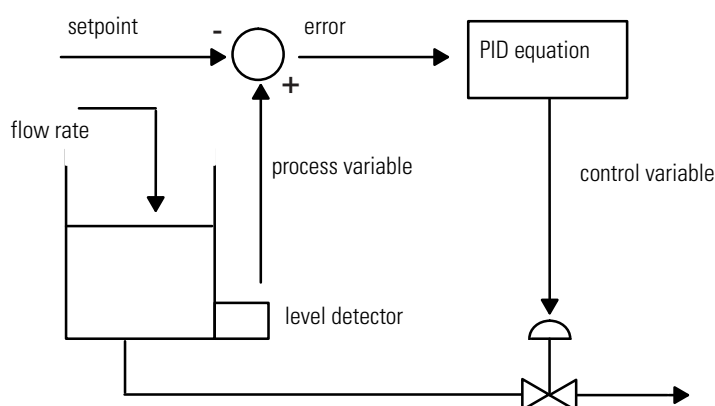
Specifying scaling

Select the Scaling tab. You must click OK or Apply for any changes to take effect.

In this field:	Specify:
PV unscaled maximum	Enter a maximum PV value (.MAXI) that equals the maximum unscaled value received from the analog input channel for the PV value.
PV unscaled minimum	Enter a minimum PV value (.MINI) that equals the minimum unscaled value received from the analog input channel for the PV value.
PV engineering units maximum	Enter the maximum engineering units corresponding to .MAXI (.MAXS)
PV engineering units minimum	Enter the minimum engineering units corresponding to .MINI (.MINS)
CV maximum	Enter a maximum CV value corresponding to 100% (.MAXCV).
CV minimum	Enter a minimum CV value corresponding to 0% (.MINCV).
Tieback maximum	Enter a maximum tieback value (.MAXTIE) that equals the maximum unscaled value received from the analog input channel for the tieback value.
Tieback minimum	Enter a minimum tieback value (.MINTIE) that equals the minimum unscaled value received from the analog input channel for the tieback value.
PID Initialized	If you change scaling constants during Run mode, turn this off to reinitialize internal descaling values (.INI).

Using PID Instructions

PID closed-loop control holds a process variable at a desired set point. The following figure shows a flow-rate/fluid level example:



14271

In the above example, the level in the tank is compared against the setpoint. If the level is higher than the setpoint, the PID equation increases the control variable and causes the outlet valve from the tank to open; thereby decreasing the level in the tank.

The PID equation used in the PID instruction is a positional form equation with the option of using either independent gains or dependent gains. When using independent gains, the proportional, integral, and derivative gains only affect their specific proportional, integral, or derivative terms respectively. When using dependent gains, the proportional gain is replaced with a controller gain which affects all three terms. You can use either form of equation to perform the same type of control. The two equation types are merely provided to let you use the equation type with which you are most familiar.

Gains Option:	Derivative of:	Equation:
Dependent gains (ISA standard)	error (E)	$CV = K_C \left[E + \frac{1}{T_i} \int_0^t E dt + T_d \frac{dE}{dt} \right] + BIAS$
	process variable (PV)	$E = SP - PV$ $CV = K_C \left[E + \frac{1}{T_i} \int_0^t E dt - T_d \frac{dPV}{dt} \right] + BIAS$ $E = PV - SP$ $CV = K_C \left[E + \frac{1}{T_i} \int_0^t E dt + T_d \frac{dPV}{dt} \right] + BIAS$
Independent gains	error (E)	$CV = K_p E + K_i \int_0^t E dt + K_d \frac{dE}{dt} + BIAS$
	process variable (PV)	$E = SP - PV$ $CV = K_p E + K_i \int_0^t E dt - K_d \frac{dPV}{dt} + BIAS$ $E = PV - SP$ $CV = K_p E + K_i \int_0^t E dt + K_d \frac{dPV}{dt} + BIAS$

Where:

Variable:	Description:
K_p	proportional gain (unitless) $K_p = K_c$ unitless
K_i	integral gain (seconds ⁻¹) To convert between K_i (integral gain) and T_i (reset time), use: $K_i = \frac{K_c}{60 T_i}$
K_d	derivative gain (seconds) To convert between K_d (derivative gain) and T_d (rate time), use: $K_d = K_c (T_d) 60$
K_c	controller gain (unitless)
T_i	reset time (minutes/repeat)
T_d	rate time (minutes)
SP	setpoint
PV	process variable
E	error [(SP-PV) or (PV-SP)]
BIAS	feedforward or bias
CV	control variable
dt	loop update time

If you do not want to use a particular term of the PID equation, just set its gain to zero. For example if you want no derivative action, set K_d or T_d equal to zero.

Anti-reset windup and bumpless transfer from manual to auto

The PID instruction automatically avoids reset windup by preventing the integral term from accumulating whenever the CV output reaches its maximum or minimum values, as set by .MAXO and .MINO. The accumulated integral term remains frozen until the CV output drops below its maximum limit or rises above its minimum limit. Then normal integral accumulation automatically resumes.

The PID instruction supports two manual modes of control:

Manual Mode of Control:	Description:
software manual (.SWM)	also known as set output mode lets the user set the output % from the software The set output (.SO) value is used as the output of the loop. The set output value typically comes from an operator input from an operator interface device.
manual (.MO)	takes the tieback value, as an input, and adjusts its internal variables to generate the same value at the output The tieback input to the PID instruction is scaled to 0-100% according to the values of .MINTIE and .MAXTIE and is used as the output of the loop. The tieback input typically comes from the output of a hardware hand/auto station which is bypassing the output from the controller. Note: Manual mode overrides software manual mode if both mode bits are set on.

The PID instruction also automatically provides bumpless transfers from software manual mode to auto mode or from manual to auto mode. The PID instruction back-calculates the value of the integral accumulation term required to make the CV output track either the set output (.SO) value in software manual mode or the tieback input in manual mode. In this manner, when the loop switches to auto mode, the CV output starts off from the set output or tieback value and no “bump” in output value occurs.

The PID instruction can also automatically provide a bumpless transfer from manual to auto even if integral control is not used (i.e. $K_i = 0$). In this case the instruction modifies the .BIAS term to make the CV output track either the set output or tieback values. When automatic control is resumed, the .BIAS term will maintain its last value. You can disable back-calculation of the .BIAS term by setting the .NOBC bit in the PID data structure. Be aware that if you set .NOBC true, the PID instruction no longer provides a bumpless transfer from manual to auto when integral control is not used.

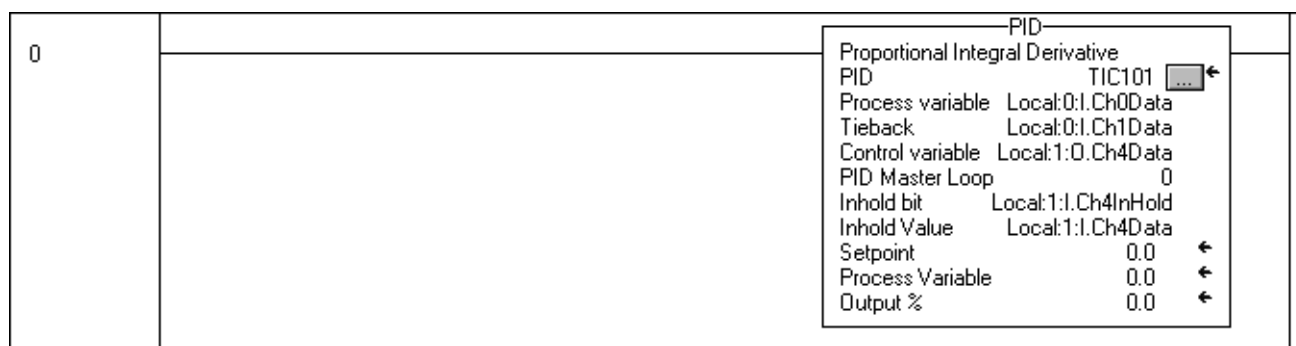
PID instruction timing

The PID instruction and the sampling of the process variable need to be updated at a periodic rate. This update time is related to the physical process you are controlling. For very slow loops, such as temperature loops, an update time of once per second or even longer is usually sufficient to obtain good control. Somewhat faster loops, such as pressure or flow loops, may require an update time such as once every 250 milliseconds. Only rare cases, such as tension control on an unwinder spool, require loop updates as fast as every 10 milliseconds or faster.

Because the PID instruction uses a time base in its calculation, you need to synchronize execution of this instruction with sampling of the process variable (PV).

The easiest way to execute the PID instruction is to put the PID instruction in a periodic task. Set the loop update time (.UPD) equal to the periodic task rate and make sure that the PID instruction is executed every scan of the periodic task

Relay Ladder



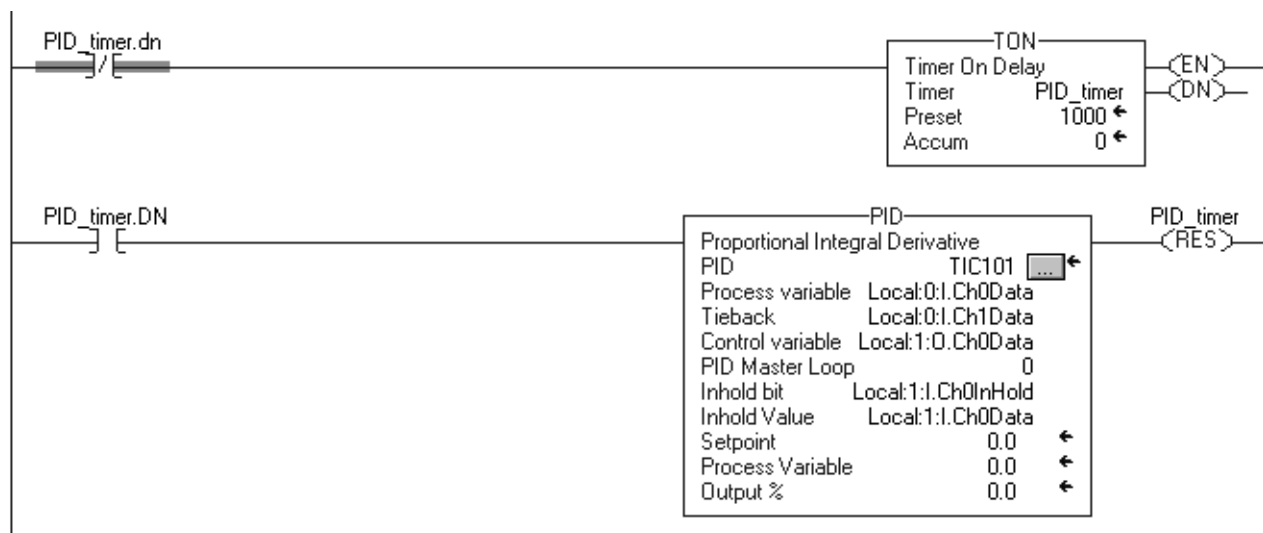
Structured Text

```
PID(TIC101, Local:0:I.Ch0Data, Local:0:I.Ch1Data,
    Local:1:O.Ch4Data, 0, Local:1:I.Ch4InHold,
    Local:1:I.Ch4Data);
```

When using a periodic task, make sure that the analog input used for the process variable is updated to the processor at a rate that is significantly faster than the rate of the periodic task. Ideally, the process variable should be sent to the processor at least five to ten times faster than the periodic task rate. This minimizes the time difference between actual samples of the process variable and execution of the PID loop. For example, if the PID loop is in a 250 millisecond periodic task, use a loop update time of 250 milliseconds (.UPD = .25), and configure the analog input module to produce data at least about every 25 to 50 msecs.

Another, somewhat less accurate, method of executing a PID instruction is to place the instruction in a continuous task and use a timer done bit to trigger execution of the PID instruction.

Relay Ladder



Structured Text

```

PID_timer.pre := 1000

TONR (PID_timer);

IF PID_timer.DN THEN

    PID (TIC101, Local:0:I.Ch0Data, Local:0:I.Ch1Data,
        Local:1:O.Ch0Data, 0, Local:1:I.Ch0InHold,
        Local:1:I.Ch0Data);

END_IF;

```

In this method, the loop update time of the PID instruction should be set equal to the timer preset. As in the case of using a periodic task, you should set the analog input module to produce the process variable at a significantly faster rate than the loop update time. You should only use the timer method of PID execution for loops with loop update times that are at least several times longer than the worst-case execution time for your continuous task.

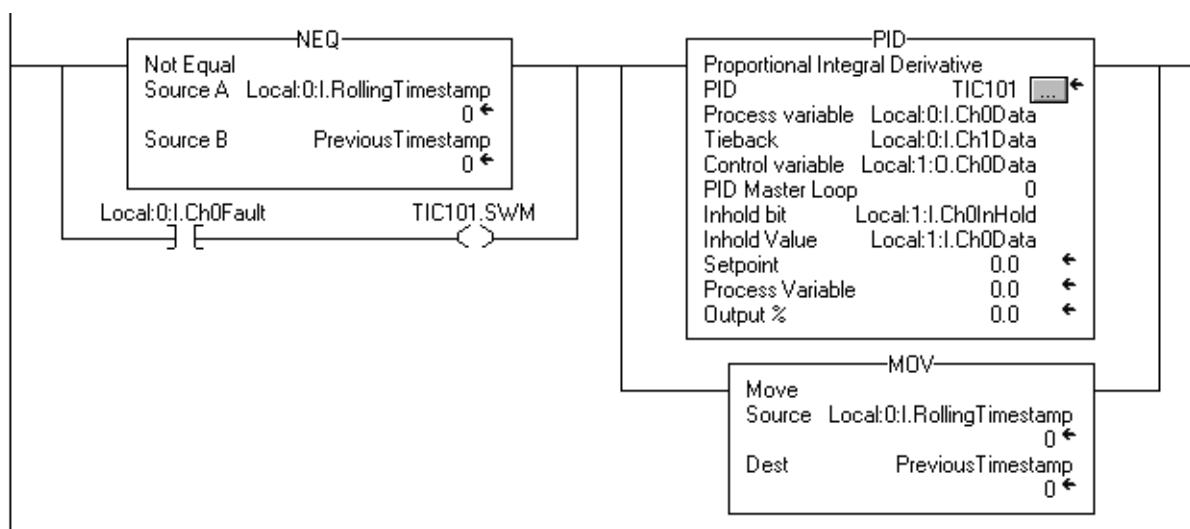
The most accurate way to execute a PID instruction is to use the real time sampling (RTS) feature of the 1756 analog input modules. The analog input module samples its inputs at the real time sampling rate you configure when you set up the module. When the module's real time sample period expires, it updates its inputs and updates a rolling timestamp (represented by the .RollingTimestamp member of the analog input data structure) produced by the module.

The timestamp ranges from 0-32767 milliseconds. Monitor the timestamp. When it changes, a new process variable sample has been received. Every time a timestamp changes, execute the PID instruction once. Because the process variable sample is driven by the analog input module, the input sample time is very accurate, and the loop update time used by the PID instruction should be set equal to the RTS time of the analog input module.

To make sure that you do not miss samples of the process variable, execute your logic at a rate faster than the RTS time. For example, if the RTS time is 250 msecs, you could put the PID logic in a periodic task that runs every 100 msecs to make sure that you never miss a sample. You could even place the PID logic in a continuous task, as long as you make sure that the logic would be updated more frequently than once every 250 milliseconds.

An example of the RTS method of execution is shown below. The execution of the PID instruction depends on receiving new analog input data. If the analog input module fails or is removed, the controller stops receiving rolling timestamps and the PID loop stops executing. You should monitor the status bit of the PV analog input and if it shows bad status, force the loop into software manual mode and execute the loop every scan. This lets operator still manually change the output of the PID loop.

Relay Ladder



Structured Text

```
IF (Local:0:I.Ch0Fault) THEN
    TIC101.SWM [:=] 1;
ELSE
    TIC101.SWM := 0;
END_IF;

IF (Local:0:I.RollingTimestamp<>PreviousTimestamp) OR
    (Local:0:I.Ch0Fault) THEN

    PreviousTimestamp := Local:0:I.RollingTimestamp;

    PID(TIC101,Local:0:I.Ch0Data,Local:0:I.Ch1Data,
        Local:1:O.Ch0Data,0,Local:1:I.Ch0InHold,
        Local:1:I.Ch0Data);

END_IF;
```

Bumpless restart

The PID instruction can interact with the 1756 analog output modules to support a bumpless restart when the controller changes from Program to Run mode or when the controller powers up.

When a 1756 analog output module loses communications with the controller or senses that the controller is in Program mode, the analog output module sets its outputs to the fault condition values you specified when you configured the module. When the controller then returns to Run mode or re-establishes communications with the analog output module, you can have the PID instruction automatically reset its control variable output equal to the analog output by using the Inhold bit and Inhold Value parameters on the PID instruction.

To set a bumpless restart:

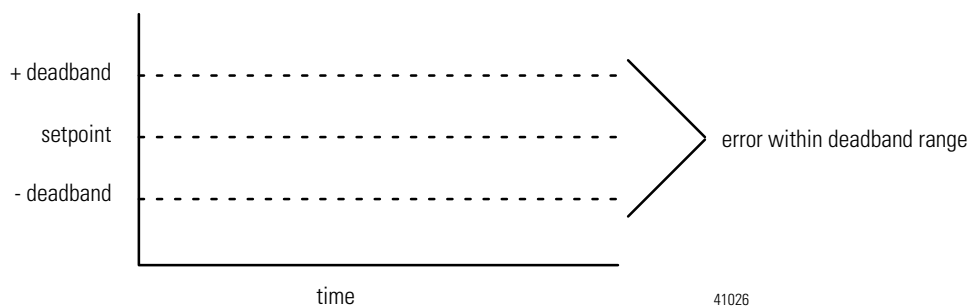
Do this:	Details:
Configure the 1756 analog output module's channel which receives the control variable from the PID instruction	<p>Select the "hold for initialization" check box on the properties page for the specific channel of the module.</p> <p>This tells the analog output module that when the controller returns to Run mode or re-establishes communications with the module, the module should hold the analog output at its current value until the value sent from the controller matches (within 0.1% of span) the current value used by the output channel. The controller's output will ramp to the currently held output value by making use of the .BIAS term. This ramping is similar to auto bumpless transfer.</p>
Enter the Inhold bit tag and Inhold Value tag in the PID instruction	<p>The 1756 analog output module returns two values for each channel in its input data structure. The InHold status bit (.Ch2InHold, for example), when true, indicates that the analog output channel is holding its value. The Data readback value (.Ch2Data, for example) shows the current output value in engineering units.</p> <p>Enter the tag of the InHold status bit as the InHold bit parameter of the PID instruction. Enter the tag of the Data readback value as the Inhold Value parameter.</p> <p>When the Inhold bit goes true, the PID instruction moves the Inhold Value into the Control variable output and re-initializes to support a bumpless restart at that value. When the analog output module receives this value back from the controller, it turns off the InHold status bit, which allows the PID instruction to start controlling normally.</p>

Derivative smoothing

The derivative calculation is enhanced by a derivative smoothing filter. This first order, low pass, digital filter helps to minimize large derivative term spikes caused by noise in the PV. This smoothing becomes more aggressive with larger values of derivative gain. You can disable derivative smoothing if your process requires very large values of derivative gain ($K_d > 10$, for example). To disable derivative smoothing, select the "No derivative smoothing" option on the Configuration tab or set the .NDF bit in the PID structure.

Setting the deadband

The adjustable deadband lets you select an error range above and below the setpoint where output does not change as long as the error remains within this range. This deadband lets you control how closely the process variable matches the setpoint without changing the output. The deadband also helps to minimize wear and tear on your final control device.



Zero-crossing is deadband control that lets the instruction use the error for computational purposes as the process variable crosses into the deadband until the process variable crosses the setpoint. Once the process variable crosses the setpoint (error crosses zero and changes sign) and as long as the process variable remains in the deadband, the output will not change.

The deadband extends above and below the setpoint by the value you specify. Enter zero to inhibit the deadband. The deadband has the same scaled units as the setpoint. You can use the deadband without the zero-crossing feature by selecting the “no zero crossing for deadband” option on the Configuration tab or set the .NOZC bit in the PID structure.

If you are using the deadband, the Control variable must be REAL or it will be forced to 0 when the error is within the deadband

Using output limiting

You can set an output limit (percentage of output) on the control output. When the instruction detects that the output has reached a limit, it sets an alarm bit and prevents the output from exceeding either the lower or upper limit.

Feedforward or output biasing

You can feedforward a disturbance from the system by feeding the .BIAS value into the PID instruction's feedforward/bias value.

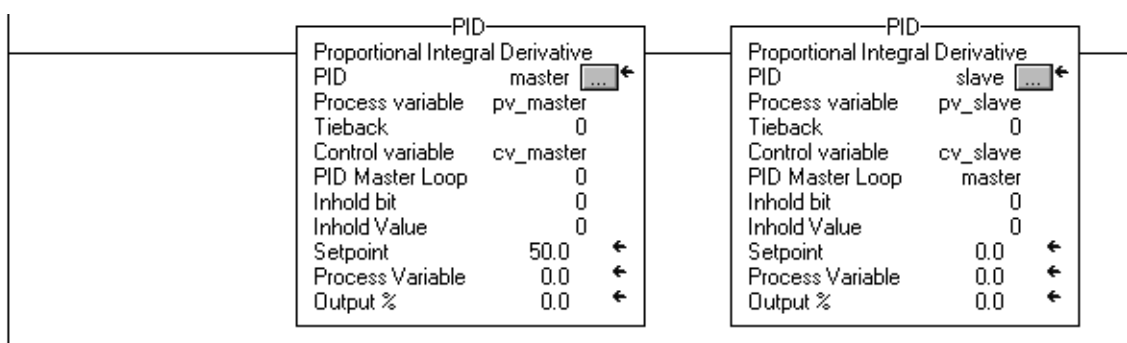
The feedforward value represents a disturbance fed into the PID instruction before the disturbance has a chance to change the process variable. Feedforward is often used to control processes with a transportation lag. For example, a feedforward value representing “cold water poured into a warm mix” could boost the output value faster than waiting for the process variable to change as a result of the mixing.

A bias value is typically used when no integral control is used. In this case, the bias value can be adjusted to maintain the output in the range required to keep the PV near the setpoint.

Cascading loops

The PID cascades two loops by assigning the output in percent of the master loop to the setpoint of the slave loop. The slave loop automatically converts the output of the master loop into the correct engineering units for the setpoint of the slave loop, based on the slave loop's values for .MAXS and .MINS.

Relay Ladder



Structured Text

```
PID(master,pv_master,0,cv_master,0,0,0);
```

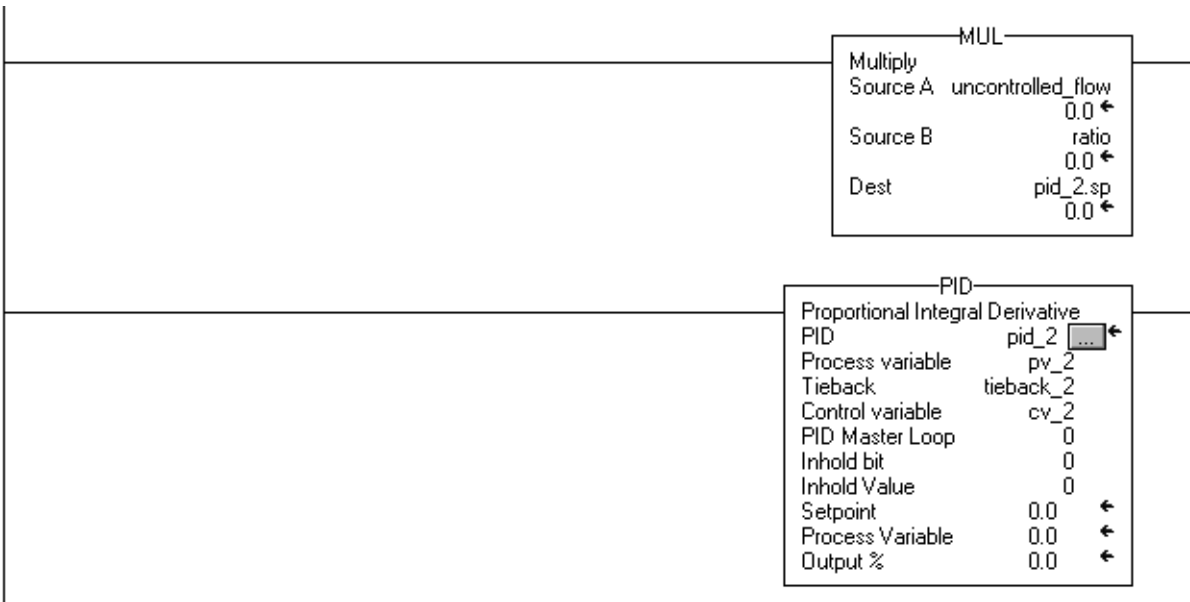
```
PID(slave,pv_slave,0,cv_slave,master,0,0);
```

Controlling a ratio

You can maintain two values in a ratio by using these parameters:

- uncontrolled value
- controlled value (the resultant setpoint to be used by the PID instruction)
- ratio between these two values

Relay Ladder



Structured Text

```
pid_2.sp := uncontrolled_flow * ratio
```

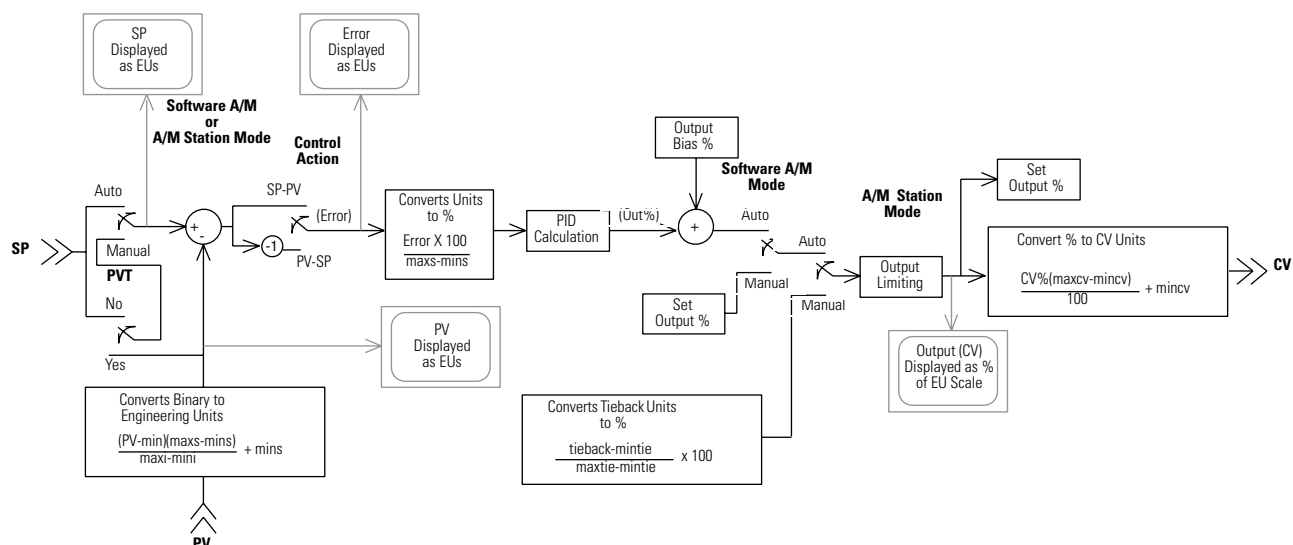
```
PID(pid_2,pv_2,tieback_2,cv_2,0,0,0);
```

For this multiplication parameter:	Enter this value:
destination	controlled value
source A	uncontrolled value
source B	ratio

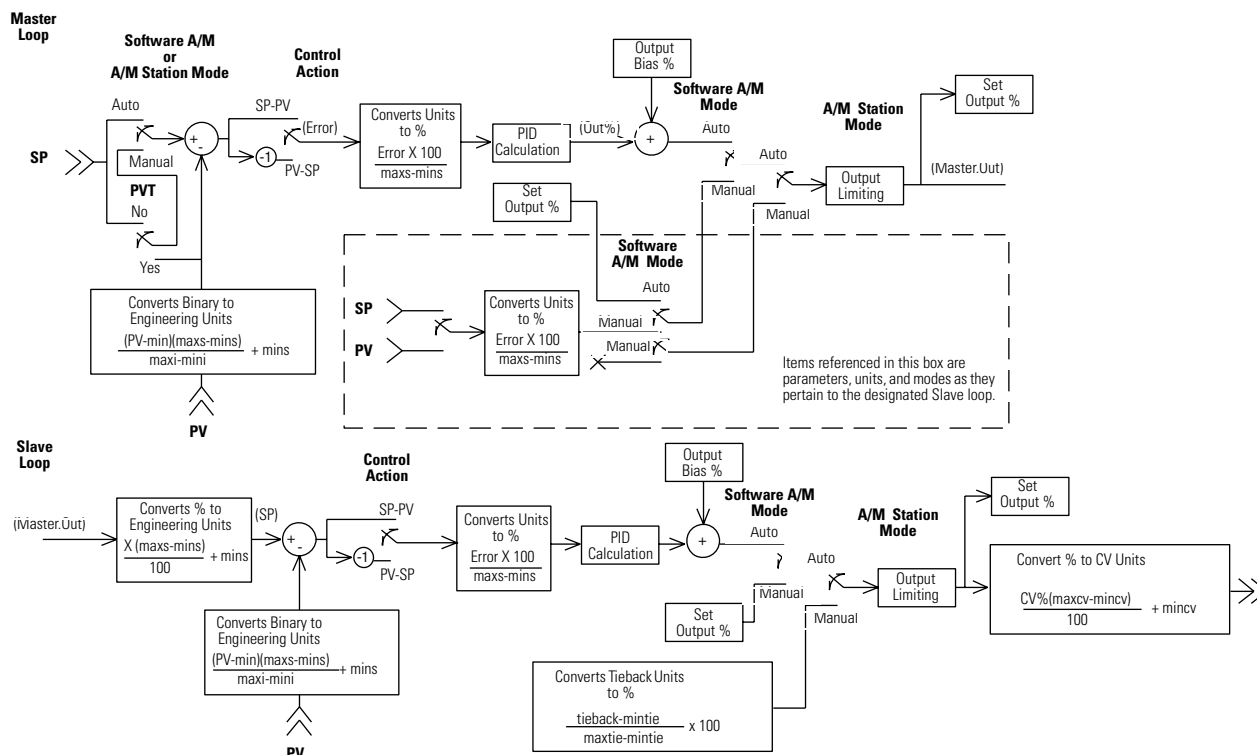
PID Theory

The following figures show the process flow for a PID instructions.

PID process



PID process with master/slave loops



Notes:

Trigonometric Instructions

(SIN, COS, TAN, ASN, ASIN, ACS, ACOS, ATN, ATAN)

Introduction

The trigonometric instructions evaluate arithmetic operations using trigonometric operations.

If you want to:	Use this instruction:	Available in these languages:	See page:
Take the sine of a value.	SIN	relay ladder structured text function block	13-2
Take the cosine of a value.	COS	relay ladder structured text function block	13-5
Take the tangent of a value.	TAN	relay ladder structured text function block	13-8
Take the arc sine of a value.	ASN ASIN ⁽¹⁾	relay ladder structured text function block	13-11
Take the arc cosine of a value.	ACS ACOS ⁽¹⁾	relay ladder structured text function block	13-14
Take the arc tangent of a value.	ATN ATAN ⁽¹⁾	relay ladder structured text function block	13-17

⁽¹⁾ Structured text only.

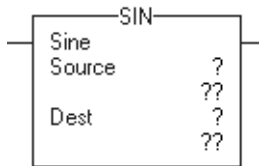
You can mix data types, but loss of accuracy and rounding error might occur and the instruction takes more time to execute. Check the overflow status bit (S:V) to see whether the result was truncated.

For relay ladder instructions, **bold** data types indicate optimal data types. An instruction executes faster and requires less memory if all the operands of the instruction use the same optimal data type, typically DINT or REAL.

Sine (SIN)

The SIN instruction takes the sine of the Source value (in radians) and stores the result in the Destination.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Source	SINT INT DINT REAL	immediate tag	find the sine of this value
Destination	SINT INT DINT REAL	tag	tag to store the result

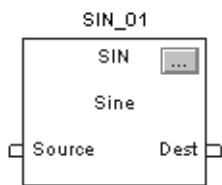


```
dest := SIN(source);
```

Structured Text

Use SIN as a function. This function computes the sine of *source* and stores the result in *dest*.

See C for information on the syntax of expressions within structured text.



Function Block

Operand:	Type:	Format:	Description:
SIN tag	FBD_MATH_ADVANCED	structure	SIN structure

FBD_MATH_ADVANCED Structure

Input Parameter:	Data Type:	Description:
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set.
Source	REAL	Input to the math instruction. Valid = any float
Output Parameter:	Data Type:	Description:
EnableOut	BOOL	The instruction produced a valid result.
Dest	REAL	Result of the math instruction. Arithmetic status flags are set for this output.

Description: The Source must be greater than or equal to $-205887.4 (-2\pi \times 2^{15})$ and less than or equal to $205887.4 (2\pi \times 2^{15})$. The resulting value in the Destination is always greater than or equal to -1 and less than or equal to 1.

Arithmetic Status Flags: Arithmetic status flags are affected.

Fault Conditions: none

Execution:



Relay Ladder

Condition:	Action:
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	The controller calculates the sine of the Source and places the result in the Destination. The rung-condition-out is set to true.
postscan	The rung-condition-out is set to false.

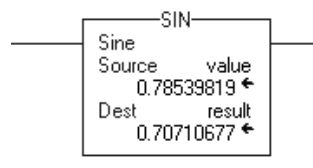


Function Block

Condition:	Action:
prescan	No action taken.
instruction first scan	No action taken.
instruction first run	No action taken.
EnableIn is cleared	EnableOut is cleared.
EnableIn is set	The instruction executes. EnableOut is set.
postscan	No action taken.

Example: Calculate the sine of *value* and place the result in *result*.

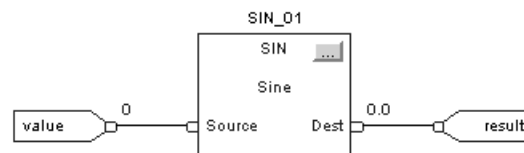
Relay Ladder



Structured Text

```
result := SIN(value);
```

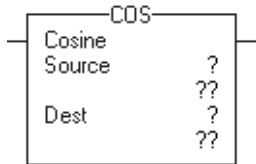
Function Block



Cosine (COS)

The COS instruction takes the cosine of the Source value (in radians) and stores the result in the Destination.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Source	SINT INT DINT REAL	immediate tag	find the cosine of this value
Destination	SINT INT DINT REAL	tag	tag to store the result

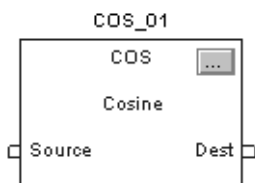


```
dest := COS(source);
```

Structured Text

Use COS as a function. This function computes the cosine of *source* and stores the result in *dest*.

See C for information on the syntax of expressions within structured text.



Function Block

Operand:	Type:	Format:	Description:
COS tag	FBD_MATH_ADVANCED	structure	COS structure

FBD_MATH_ADVANCED Structure

Input Parameter:	Data Type:	Description:
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set.
Source	REAL	Input to the math instruction. Valid = any float
Output Parameter:	Data Type:	Description:
EnableOut	BOOL	The instruction produced a valid result.
Dest	REAL	Result of the math instruction. Arithmetic status flags are set for this output.

Description: The Source must be greater than or equal to $-205887.4 (-2\pi \times 2^{15})$ and less than or equal to $205887.4 (2\pi \times 2^{15})$. The resulting value in the Destination is always greater than or equal to -1 and less than or equal to 1.

Arithmetic Status Flags: Arithmetic status flags are affected.

Fault Conditions: none

Execution:



Relay Ladder

Condition:	Action:
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	The controller calculates the cosine of the Source and places the result in the Destination. The rung-condition-out is set to true.
postscan	The rung-condition-out is set to false.

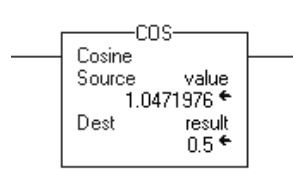


Function Block

Condition:	Action:
prescan	No action taken.
instruction first scan	No action taken.
instruction first run	No action taken.
EnableIn is cleared	EnableOut is cleared.
EnableIn is set	The instruction executes. EnableOut is set.
postscan	No action taken.

Example: Calculate the cosine of *value* and place the result in *result*.

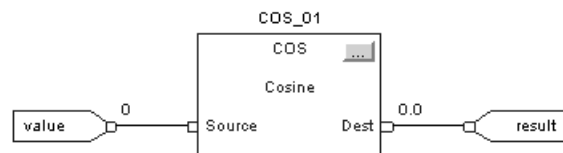
Relay Ladder



Structured Text

```
result := COS(value);
```

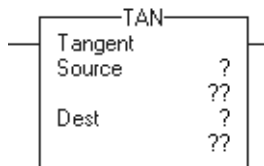
Function Block



Tangent (TAN)

The TAN instruction takes the tangent of the Source value (in radians) and stores the result in the Destination.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Source	SINT INT DINT REAL	immediate tag	find the tangent of this value
Destination	SINT INT DINT REAL	tag	tag to store the result

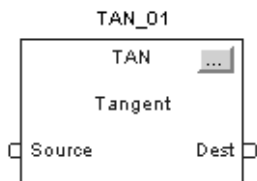


```
dest := TAN(source);
```

Structured Text

Use TAN as a function. This function computes the tangent of *source* and stores the result in *dest*.

See C for information on the syntax of expressions within structured text.



Function Block

Operand:	Type:	Format:	Description:
TAN tag	FBD_MATH_ADVANCED	structure	TAN structure

FBD_MATH_ADVANCED Structure

Input Parameter:	Data Type:	Description:
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set.
Source	REAL	Input to the math instruction. Valid = any float
Output Parameter:	Data Type:	Description:
EnableOut	BOOL	The instruction produced a valid result.
Dest	REAL	Result of the math instruction. Arithmetic status flags are set for this output.

Description: The Source must be greater than or equal to $-102943.7(-2\pi \times 2^{14})$ and less than or equal to $102943.7(2\pi \times 2^{14})$.

Arithmetic Status Flags: Arithmetic status flags are affected.

Fault Conditions: none

Execution:



Relay Ladder

Condition:	Action:
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	The controller calculates the tangent of the Source and places the result in the Destination. The rung-condition-out is set to true.
postscan	The rung-condition-out is set to false.

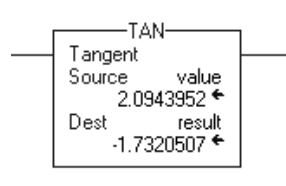


Function Block

Condition:	Action:
prescan	No action taken.
instruction first scan	No action taken.
instruction first run	No action taken.
EnableIn is cleared	EnableOut is cleared.
EnableIn is set	The instruction executes. EnableOut is set.
postscan	No action taken.

Example: Calculate the tangent of *value* and place the result in *result*.

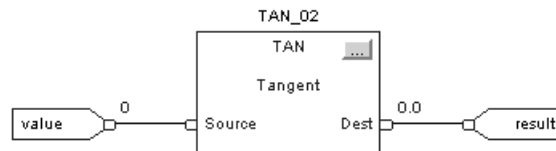
Relay Ladder



Structured Text

```
result := TAN(value);
```

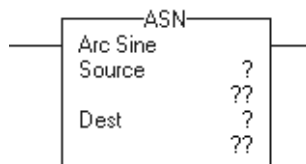
Function Block



Arc Sine (ASN)

The ASN instruction takes the arc sine of the Source value and stores the result in the Destination (in radians).

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Source	SINT INT DINT REAL	immediate tag	find the arc sine of this value
Destination	SINT INT DINT REAL	tag	tag to store the result

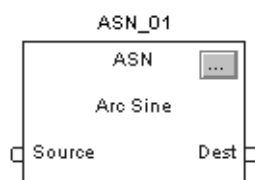


```
dest := ASIN(source);
```

Structured Text

Use ASIN as a function. This function computes the arc sine of *source* and stores the result in *dest*.

See C for information on the syntax of expressions within structured text.



Function Block

Operand:	Type:	Format:	Description:
ASN tag	FBD_MATH_ADVANCED	structure	ASN structure

FBD_MATH_ADVANCED Structure

Input Parameter:	Data Type:	Description:
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set.
Source	REAL	Input to the math instruction. Valid = any float
Output Parameter:	Data Type:	Description:
EnableOut	BOOL	The instruction produced a valid result.
Dest	REAL	Result of the math instruction. Arithmetic status flags are set for this output.

Description: The Source must be greater than or equal to -1 and less than or equal to 1. The resulting value in the Destination is always greater than or equal to $-\pi/2$ and less than or equal to $\pi/2$ (where $\pi = 3.141593$).

Arithmetic Status Flags: Arithmetic status flags are affected.

Fault Conditions: none**Execution:****Relay Ladder**

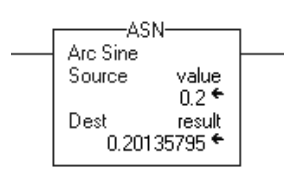
Condition:	Action:
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	The controller calculates the arc sine of the Source and places the result in the Destination. The rung-condition-out is set to true.
postscan	The rung-condition-out is set to false.

**Function Block**

Condition:	Action:
prescan	No action taken.
instruction first scan	No action taken.
instruction first run	No action taken.
EnableIn is cleared	EnableOut is cleared.
EnableIn is set	The instruction executes. EnableOut is set.
postscan	No action taken.

Example: Calculate the arc sine of *value* and place the result in *result*.

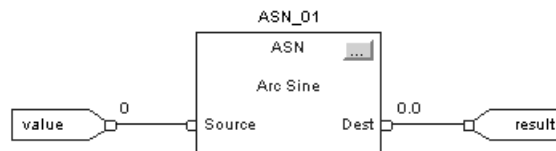
Relay Ladder



Structured Text

```
result := ASIN(value);
```

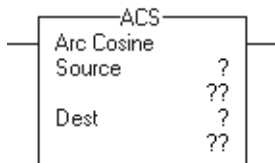
Function Block



Arc Cosine (ACS)

The ACS instruction takes the arc cosine of the Source value and stores the result in the Destination (in radians).

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Source	SINT INT DINT REAL	immediate tag	find the arc cosine of this value
Destination	SINT INT DINT REAL	tag	tag to store the result

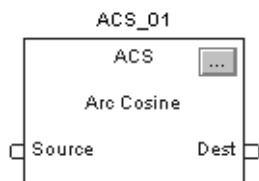


```
dest := ACOS(source);
```

Structured Text

Use ACOS as a function. This function computes the arc cosine of *source* and stores the result in *dest*.

See C for information on the syntax of expressions within structured text.



Function Block

Operand:	Type:	Format:	Description:
ACS tag	FBD_MATH_ADVANCED	structure	ACS structure

FBD_MATH_ADVANCED Structure

Input Parameter:	Data Type:	Description:
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set.
Source	REAL	Input to the math instruction. Valid = any float
Output Parameter:	Data Type:	Description:
EnableOut	BOOL	The instruction produced a valid result.
Dest	REAL	Result of the math instruction. Arithmetic status flags are set for this output.

Description: The Source must be greater than or equal to -1 and less than or equal to 1. The resulting value in the Destination is always greater than or equal to 0 or less than or equal to π (where $\pi = 3.141593$).

Arithmetic Status Flags: Arithmetic status flags are affected.

Fault Conditions: none

Execution:



Relay Ladder

Condition:	Action:
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	The controller calculates the arc cosine of the Source and places the result in the Destination. The rung-condition-out is set to true.
postscan	The rung-condition-out is set to false.

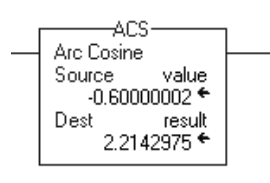


Function Block

Condition:	Action:
prescan	No action taken.
instruction first scan	No action taken.
instruction first run	No action taken.
EnableIn is cleared	EnableOut is cleared.
EnableIn is set	The instruction executes. EnableOut is set.
postscan	No action taken.

Example: Calculate the arc cosine of *value* and place the result in *result*.

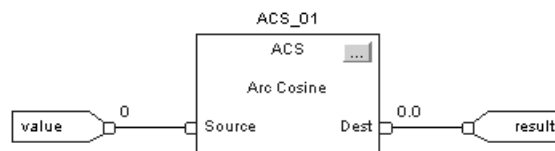
Relay Ladder



Structured Text

```
result := ACOS(value);
```

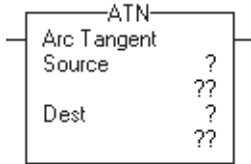
Function Block



Arc Tangent (ATN)

The ATN instruction takes the arc tangent of the Source value and stores the result in the Destination (in radians).

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Source	SINT INT DINT REAL	immediate tag	find the arc tangent of this value
Destination	SINT INT DINT REAL	tag	tag to store the result

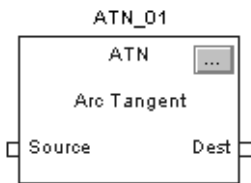


```
dest := ATAN(source);
```

Structured Text

Use ATAN as a function. This function computes the arc tangent of *source* and stores the result in *dest*.

See C for information on the syntax of expressions within structured text.



Function Block

Operand:	Type:	Format:	Description:
ATN tag	FBD_MATH_ADVANCED	structure	ATN structure

FBD_MATH_ADVANCED Structure

Input Parameter:	Data Type:	Description:
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set.
Source	REAL	Input to the math instruction. Valid = any float
Output Parameter:	Data Type:	Description:
EnableOut	BOOL	The instruction produced a valid result.
Dest	REAL	Result of the math instruction. Arithmetic status flags are set for this output.

Description: The resulting value in the Destination is always greater than or equal to $-\pi/2$ and less than or equal to $\pi/2$ (where $\pi = 3.141593$).

Arithmetic Status Flags: Arithmetic status flags are affected.

Fault Conditions: none**Execution:****Relay Ladder**

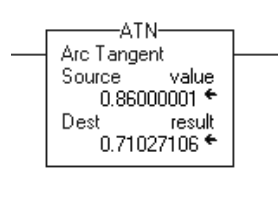
Condition:	Action:
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	The controller calculates the arc tangent of the Source and places the result in the Destination. The rung-condition-out is set to true.
postscan	The rung-condition-out is set to false.

**Function Block**

Condition:	Action:
prescan	No action taken.
instruction first scan	No action taken.
instruction first run	No action taken.
EnableIn is cleared	EnableOut is cleared.
EnableIn is set	The instruction executes. EnableOut is set.
postscan	No action taken.

Example: Calculate the arc tangent of *value* and place the result in *result*.

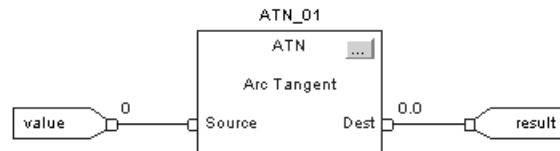
Relay Ladder



Structured Text

```
result := ATAN(value);
```

Function Block



Notes:

Advanced Math Instructions

(LN, LOG, XPY)

Introduction

The advanced math instructions include these instructions:

If you want to:	Use this instruction:	Available in these languages:	See page:
Take the natural log of a value.	LN	relay ladder structured text function block	14-2
Take the log base 10 of a value.	LOG	relay ladder structured text function block	14-4
Raise a value to the power of another value.	XPY	relay ladder structured text ⁽¹⁾ function block	14-6

⁽¹⁾ There is no equivalent structured text instruction. Use the operator in an expression.

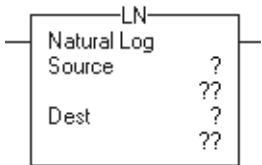
You can mix data types, but loss of accuracy and rounding error might occur and the instruction takes more time to execute. Check the S:V bit to see whether the result was truncated.

For relay ladder instructions, **bold** data types indicate optimal data types. An instruction executes faster and requires less memory if all the operands of the instruction use the same optimal data type, typically DINT or REAL.

Natural Log (LN)

The LN instruction takes the natural log of the Source and stores the result in the Destination.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Source	SINT INT DINT REAL	immediate tag	find the natural log of this value
Destination	SINT INT DINT REAL	tag	tag to store the result

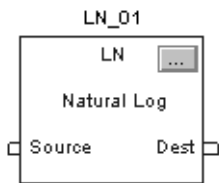


```
dest := LN(source);
```

Structured Text

Use LN as a function. This function computes the natural log of *source* and stores the result in *dest*.

See C for information on the syntax of expressions within structured text.



Function Block

Operand:	Type:	Format:	Description:
LN tag	FBD_MATH_ADVANCED	structure	LN structure

FBD_MATH_ADVANCED Structure

Input Parameter:	Data Type:	Description:
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set.
Source	REAL	Input to math instruction. Valid = any float
Output Parameter:	Data Type:	Description:
EnableOut	BOOL	The instruction produced a valid result.

Description: The Source must be greater than zero, otherwise the overflow status bit (S:V) is set. The resulting Destination is greater than or equal to -87.33655 and less than or equal to 88.72284.

Arithmetic Status Flags: Arithmetic status flags are affected.

Fault Conditions: none

Execution:



Relay Ladder

Condition:	Action:
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	The controller calculates the natural log of the Source and places the result in the Destination. The rung-condition-out is set to true.
postscan	The rung-condition-out is set to false.

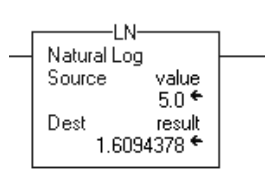


Function Block

Condition:	Action:
prescan	No action taken.
instruction first scan	No action taken.
instruction first run	No action taken.
EnableIn is cleared	EnableOut is cleared.
EnableIn is set	The instruction executes. EnableOut is set.
postscan	No action taken.

Example: Calculate the natural log of *value* and place the result in *result*.

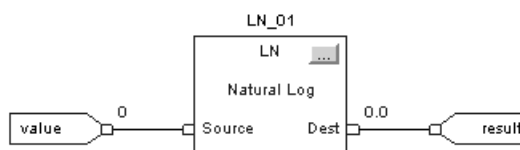
Relay Ladder Example



Structured Text

```
result := LN(value);
```

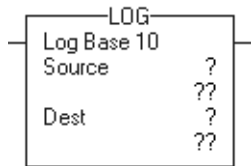
Function Block



Log Base 10 (LOG)

The LOG instruction takes the log base 10 of the Source and stores the result in the Destination.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Source	SINT INT DINT REAL	immediate tag	find the log of this value
Destination	SINT INT DINT REAL	tag	tag to store the result

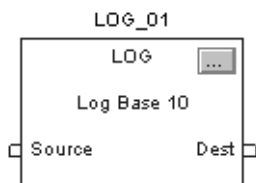


```
dest := LOG(source);
```

Structured Text

Use LOG as a function. This function computes the log of *source* and stores the result in *dest*.

See C for information on the syntax of expressions within structured text.



Function Block

Operand:	Type:	Format:	Description:
LOG tag	FBD_MATH_ADVANCED	structure	LOG structure

FBD_MATH_ADVANCED Structure

Input Parameter:	Data Type:	Description:
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set.
Source	REAL	Input to math instruction. Valid = any float
Output Parameter:	Data Type:	Description:
EnableOut	BOOL	The instruction produced a valid result.
Dest	REAL	Result of the math instruction. Arithmetic status flags are set for this output.

Description: The Source must be greater than zero, otherwise the overflow status bit (S:V) is set. The resulting Destination is greater than or equal to -37.92978 and less than or equal to 38.53184.

Arithmetic Status Flags: Arithmetic status flags are affected.

Fault Conditions: none

Execution:



Relay Ladder

Condition:	Action:
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	The controller calculates the log of the Source and places the result in the Destination. The rung-condition-out is set to true.
postscan	The rung-condition-out is set to false.



Function Block

Condition:	Action:
prescan	No action taken.
instruction first scan	No action taken.
instruction first run	No action taken.
EnableIn is cleared	EnableOut is cleared.
EnableIn is set	The instruction executes. EnableOut is set.
postscan	No action taken.

Example: Calculate the log of *value* and place the result in *result*.

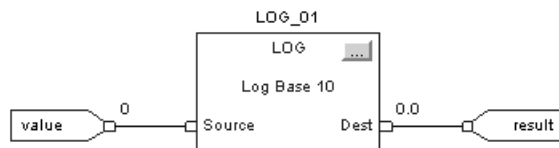
Relay Ladder



Structured Text

```
result := LOG(value);
```

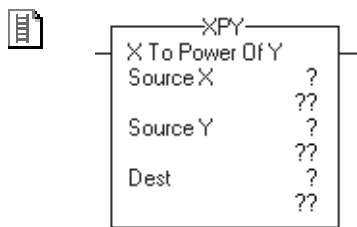
Function Block



X to the Power of Y (XPY)

The XPY instruction takes Source A (X) to the power of Source B (Y) and stores the result in the Destination.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Source X	SINT INT DINT REAL	immediate tag	base value
Source Y	SINT INT DINT REAL	immediate tag	exponent
Destination	SINT INT DINT REAL	tag	tag to store the result

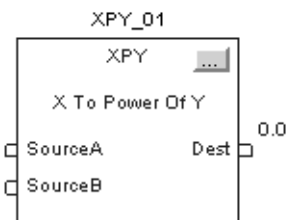


```
dest := sourceX ** sourceY;
```

Structured Text

Use two, adjacent multiply signs “**” as an operator within an expression. This expression takes *sourceX* to the power of *sourceY* and stores the result in *dest*.

See C for information on the syntax of expressions within structured text.



Function Block

Operand:	Type:	Format:	Description:
XPY tag	FBD_MATH	structure	XPY structure

FBD_MATH Structure

Input Parameter:	Data Type:	Description:
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set.
Source X	REAL	Base value. Valid = any float
Source Y	REAL	Exponent. Valid = any float
Output Parameter:	Data Type:	Description:
EnableOut	BOOL	The instruction produced a valid result.
Dest	REAL	Result of the math instruction. Arithmetic status flags are set for this output.

Description: If Source X is negative, Source Y must be an integer value or a minor fault will occur.

The XPY instruction uses this algorithm: *Destination* = $X^{**}Y$

The controller evaluates $x^0=1$ and $0^x=0$.

Arithmetic Status Flags: Arithmetic status flags are affected.

Fault Conditions:

A minor fault will occur if:	Fault type:	Fault code:
Source X is negative and Source Y is not an integer value	4	4

Execution:



Relay Ladder

Condition:	Action:
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	The controller takes Source X to the power of Source Y and places the result in the Destination. The rung-condition-out is set to true.
postscan	The rung-condition-out is set to false.

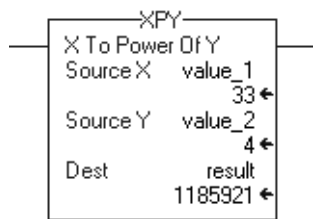


Function Block

Condition:	Action:
prescan	No action taken.
instruction first scan	No action taken.
instruction first run	No action taken.
EnableIn is cleared	EnableOut is cleared.
EnableIn is set	The instruction executes. EnableOut is set.
postscan	No action taken.

Example: The XPY instruction takes *value_1* to the power of *value_2* and places the result in *result*.

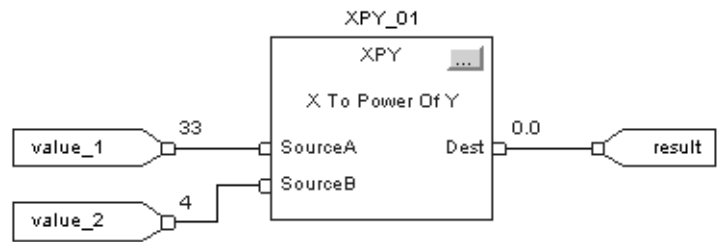
Relay Ladder



Structured Text

```
result := (value_1 ** value_2);
```

Function Block



Math Conversion Instructions

(DEG, RAD, TOD, FRD, TRN, TRUNC)

Introduction

The math conversion instructions convert values.

If you want to:	Use this instruction:	Available in these languages:	See page:
Convert radians to degrees.	DEG	relay ladder structured text function block	15-2
Convert degrees to radians.	RAD	relay ladder structured text function block	15-4
Convert an integer value to a BCD value.	TOD	relay ladder function block	15-6
Convert a BCD value to an integer value.	FRD	relay ladder function block	15-9
Remove the fractional part of a value	TRN TRUNC ⁽¹⁾	relay ladder structured text function block	15-11

⁽¹⁾ Structured text only.

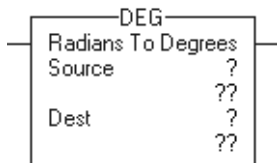
You can mix data types, but loss of accuracy and rounding error might occur and the instruction takes more time to execute. Check the S:V bit to see whether the result was truncated.

For relay ladder instructions, **bold** data types indicate optimal data types. An instruction executes faster and requires less memory if all the operands of the instruction use the same optimal data type, typically DINT or REAL.

Degrees (DEG)

The DEG instruction converts the Source (in radians) to degrees and stores the result in the Destination.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Source	SINT INT DINT REAL	immediate tag	value to convert to degrees
Destination	SINT INT DINT REAL	tag	tag to store the result

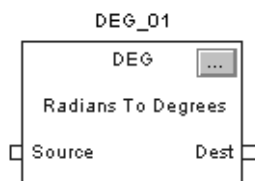


```
dest := DEG(source);
```

Structured Text

Use DEG as a function. This function converts *source* to degrees and stores the result in *dest*.

See C for information on the syntax of expressions within structured text.



Function Block

Operand:	Type:	Format:	Description:
DEG tag	FBD_MATH_ADVANCED	structure	DEG structure

FBD_MATH_ADVANCED Structure

Input Parameter:	Data Type:	Description:
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set.
Source	REAL	Input to the conversion instruction. Valid = any float
Output Parameter:	Data Type:	Description:
EnableOut	BOOL	The instruction produced a valid result.
Dest	REAL	Result of the conversion instruction. Arithmetic status flags are set for this output.

Description: The DEG instruction uses this algorithm:
 $\text{Source} * 180 / \pi$ (where $\pi = 3.141593$)

Arithmetic Status Flags: Arithmetic status flags are affected.

Fault Conditions: none

Execution:



Relay Ladder

Condition:	Action:
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	The controller converts the Source to degrees and places the result in the Destination. The rung-condition-out is set to true.
postscan	The rung-condition-out is set to false.

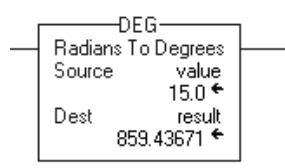


Function Block

Condition:	Action:
prescan	No action taken.
instruction first scan	No action taken.
instruction first run	No action taken.
EnableIn is cleared	EnableOut is cleared.
EnableIn is set	The instruction executes. EnableOut is set.
postscan	No action taken.

Example: Convert *value* to degrees and place the result in *result*.

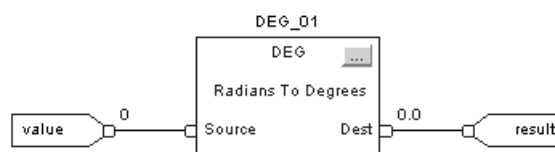
Relay Ladder



Structured Text

```
result := DEG(value);
```

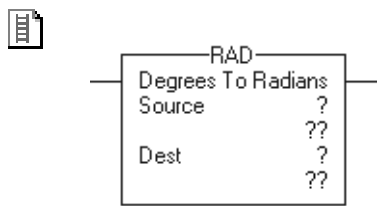
Function Block



Radians (RAD)

The RAD instruction converts the Source (in degrees) to radians and stores the result in the Destination.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Source	SINT INT DINT REAL	immediate tag	value to convert to radians
Destination	SINT INT DINT REAL	tag	tag to store the result

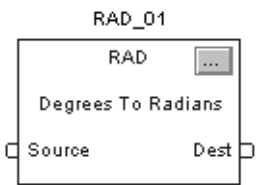


```
dest := RAD(source);
```

Structured Text

Use RAD as a function. This function converts *source* to radians and stores the result in *dest*.

See C for information on the syntax of expressions within structured text.



Function Block

Operand:	Type:	Format:	Description:
RAD tag	FBD_MATH_ADVANCED	structure	RAD structure

FBD_MATH_ADVANCED Structure

Input Parameter:	Data Type:	Description:
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set.
Source	REAL	Input to the conversion instruction. Valid = any float
Output Parameter:	Data Type:	Description:
EnableOut	BOOL	The instruction produced a valid result.
Dest	REAL	Result of the conversion instruction. Arithmetic status flags are set for this output.

Description: The RAD instruction uses this algorithm:
 $\text{Source} * \pi / 180$ (where $\pi = 3.141593$)

Arithmetic Status Flags: Arithmetic status flags are affected.

Fault Conditions: none

Execution:



Relay Ladder

Condition:	Action:
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	The controller converts the Source to radians and places the result in the Destination. The rung-condition-out is set to true.
postscan	The rung-condition-out is set to false.

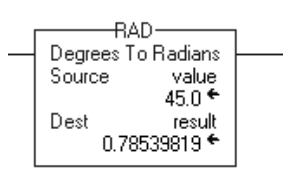


Function Block

Condition:	Action:
prescan	No action taken.
instruction first scan	No action taken.
instruction first run	No action taken.
EnableIn is cleared	EnableOut is cleared.
EnableIn is set	The instruction executes. EnableOut is set.
postscan	No action taken.

Example: Convert *value* to radians and place the result in *result*.

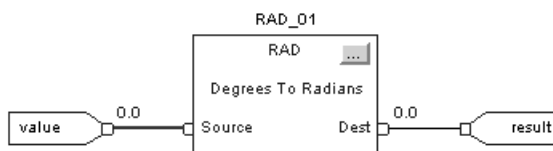
Relay Ladder



Structured Text

```
result := RAD(value);
```

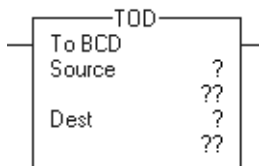
Function Block



Convert to BCD (TOD)

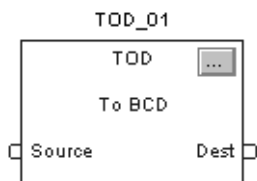
The TOD instruction converts a decimal value ($0 \leq \text{Source} \leq 99,999,999$) to a BCD value and stores the result in the Destination.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Source	SINT INT DINT	immediate tag	value to convert to decimal
A SINT or INT tag converts to a DINT value by zero-fill.			
Destination	SINT INT DINT	tag	stores the result



Function Block

Operand:	Type:	Format:	Description:
TOD tag	FBD_CONVERT	structure	TOD structure

FBD_CONVERT Structure

Input Parameter:	Data Type:	Description:
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set.
Source	DINT	Input to the conversion instruction. Valid = any integer
Output Parameter:	Data Type:	Description:
EnableOut	BOOL	The instruction produced a valid result.
Dest	DINT	Result of the conversion instruction. Arithmetic status flags are set for this output.

Description: BCD is the Binary Coded Decimal number system that expresses individual decimal digits (0-9) in a 4-bit binary notation.

If you enter a negative Source, the instruction generates a minor fault and clears the Destination.

Arithmetic Status Flags: Arithmetic status flags are affected.

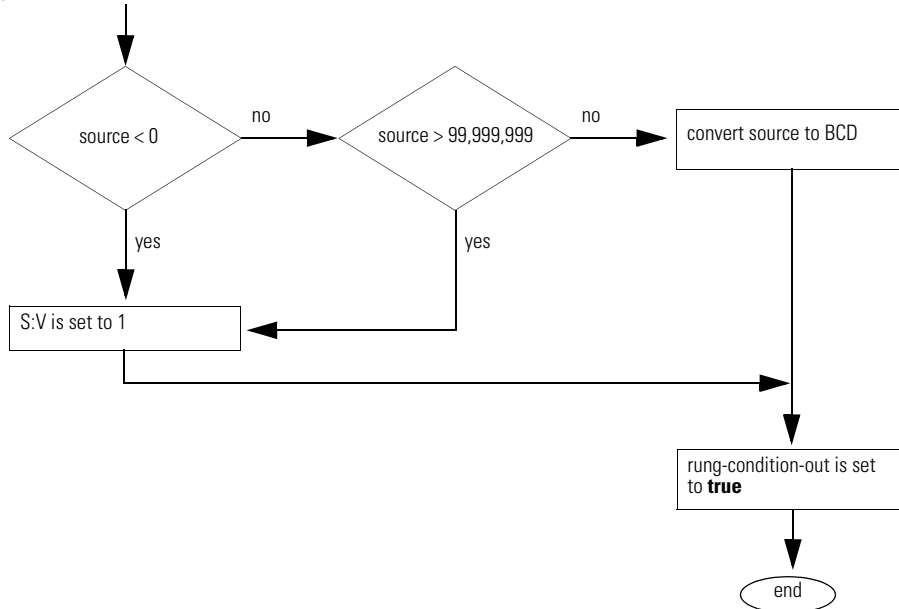
Fault Conditions:

A minor fault will occur if:	Fault type:	Fault code:
Source < 0	4	4

Execution:**Relay Ladder**

Condition:	Action:
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.

rung-condition-in is true



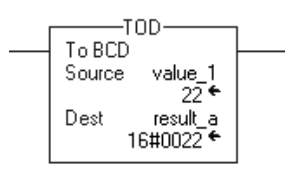
rung-condition-in is true	The controller converts the Source to BCD and places the result in the Destination. The rung-condition-out is set to true.
postscan	The rung-condition-out is set to false.

**Function Block**

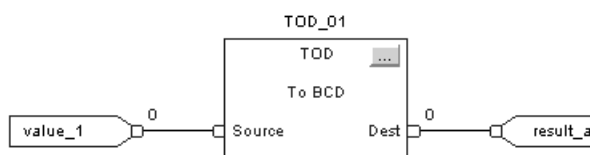
Condition:	Action:
prescan	No action taken.
instruction first scan	No action taken.
instruction first run	No action taken.
EnableIn is cleared	EnableOut is cleared.
EnableIn is set	The instruction executes. EnableOut is set.
postscan	No action taken.

Example: The TOD instruction converts *value_1* to a BCD value and places the result in *result_a*.

Relay Ladder



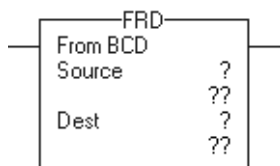
Function Block



Convert to Integer (FRD)

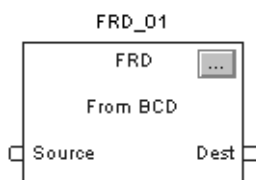
The FRD instruction converts a BCD value (Source) to a decimal value and stores the result in the Destination.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Source	SINT INT DINT	immediate tag	value to convert to decimal
A SINT or INT tag converts to a DINT value by zero-fill.			
Destination	SINT INT DINT	tag	stores the result



Function Block

Operand:	Type:	Format:	Description:
FRD tag	FBD_CONVERT	structure	FRD structure

FBD_CONVERT Structure

Input Parameter:	Data Type:	Description:
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set.
Source	DINT	Input to the conversion instruction. Valid = any integer
Output Parameter:	Data Type:	Description:
EnableOut	BOOL	The instruction produced a valid result.
Dest	DINT	Result of the conversion instruction. Arithmetic status flags are set for this output.

Description: The FRD instruction converts a BCD value (Source) to a decimal value and stores the result in the Destination.

Arithmetic Status Flags: Arithmetic status flags are affected.

Fault Conditions: none

Execution:



Relay Ladder

Condition:	Action:
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	The controller converts the Source to a decimal value and places the result in the Destination. The rung-condition-out is set to true.
postscan	The rung-condition-out is set to false.

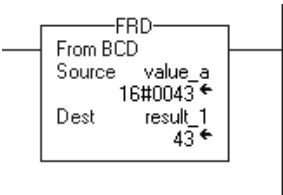


Function Block

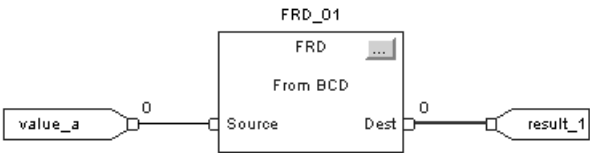
Condition:	Action:
prescan	No action taken.
instruction first scan	No action taken.
instruction first run	No action taken.
EnableIn is cleared	EnableOut is cleared.
EnableIn is set	The instruction executes. EnableOut is set.
postscan	No action taken.

Example: The FRD instruction converts *value_a* to a decimal value and places the result in *result_1*.

Relay Ladder



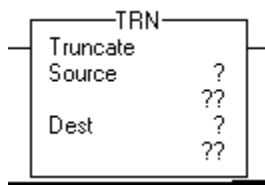
Function Block



Truncate (TRN)

The TRN instruction removes (truncates) the fractional part of the Source and stores the result in the Destination.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Source	REAL	immediate tag	value to truncate
Destination	SINT INT DINT REAL	tag	tag to store the result

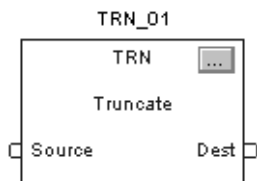


```
dest := TRUNC(source);
```

Structured Text

Use TRUNC as a function. This function truncates *source* and stores the result in *dest*.

See C for information on the syntax of expressions within structured text.



Function Block

Operand:	Type:	Format:	Description:
TRN tag	FBD_TRUNCATE	structure	TRN structure

FBD_TRUNCATE Structure

Input Parameter:	Data Type:	Description:
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set.
Source	REAL	Input to the conversion instruction. Valid = any float
Output Parameter:	Data Type:	Description:
EnableOut	BOOL	The instruction produced a valid result.
Dest	DINT	Result of the conversion instruction. Arithmetic status flags are set for this output.

Description: Truncating does not round the value; rather, the non-fractional part remains the same regardless of the value of the fractional part.

Arithmetic Status Flags: Arithmetic status flags are affected.

Fault Conditions: none

Execution:



Relay Ladder

Condition:	Action:
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	The controller removes the fractional part of the Source and places the result in the Destination. The rung-condition-out is set to true.
postscan	The rung-condition-out is set to false.

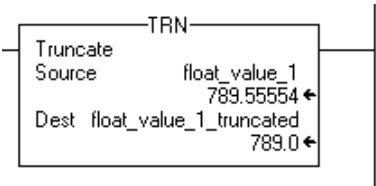


Function Block

Condition:	Action:
prescan	No action taken.
instruction first scan	No action taken.
instruction first run	No action taken.
EnableIn is cleared	EnableOut is cleared.
EnableIn is set	The instruction executes. EnableOut is set.
postscan	No action taken.

Example: Remove the fractional part of *float_value_1*, leaving the non-fractional part the same, and place the result in *float_value_1_truncated*.

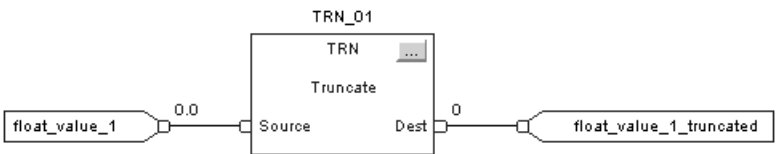
Relay Ladder



Structured Text

```
float_value_1_truncated := TRUNC(float_value_1);
```

Function Block



ASCII Serial Port Instructions

(ABL, ACB, ACL, AHL, ARD, ARL, AWA, AWT)

Introduction

Use the ASCII serial port instructions to read and write ASCII characters.

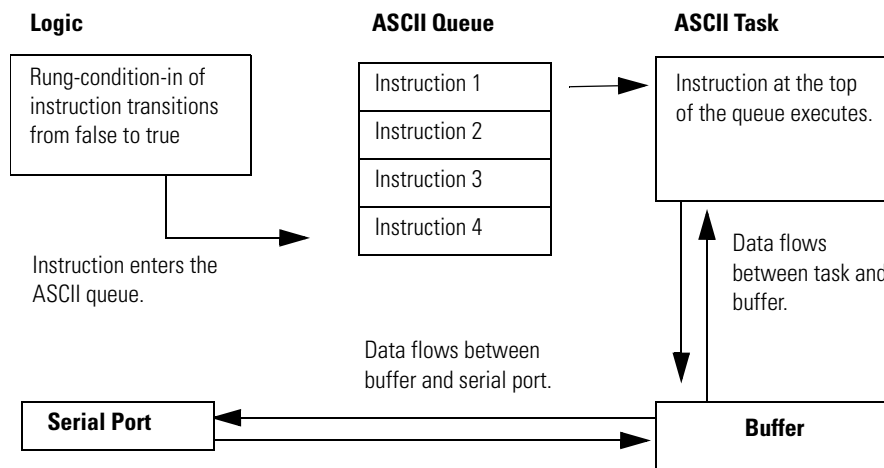
IMPORTANT

To use the ASCII serial port instructions, you must configure the serial port of the controller. See the *Logix5000 Controllers Common Procedures*, publication 1756-PM001.

If you want to:	For example:	Use this instruction:	Available in these languages:	See page:
determine when the buffer contains termination characters	check for data that contains termination characters	ABL	relay ladder structured text	16-5
count the characters in the buffer	check for the required number of characters before reading the buffer	ACB	relay ladder structured text	16-8
clear the buffer	<ul style="list-style-type: none"> remove old data from the buffer at start-up synchronize the buffer with a device 	ACL	relay ladder structured text	16-10
clear out ASCII Serial Port instructions that are currently executing or are in the queue				
obtain the status of the serial port control lines	cause a modem to hang up	AHL	relay ladder structured text	16-12
turn on or off the DTR signal				
turn on or off the RTS signal				
read a fixed number of characters	read data from a device that sends the same number of characters each transmission	ARD	relay ladder structured text	16-16
read a varying number of characters, up to and including the first set of termination characters	read data from a device that sends a varying number of characters each transmission	ARL	relay ladder structured text	16-19
send characters and automatically append one or two additional characters to mark the end of the data	send messages that always use the same termination character(s)	AWA	relay ladder structured text	16-23
send characters	send messages that use a variety of termination characters	AWT	relay ladder structured text	16-28

Instruction Execution

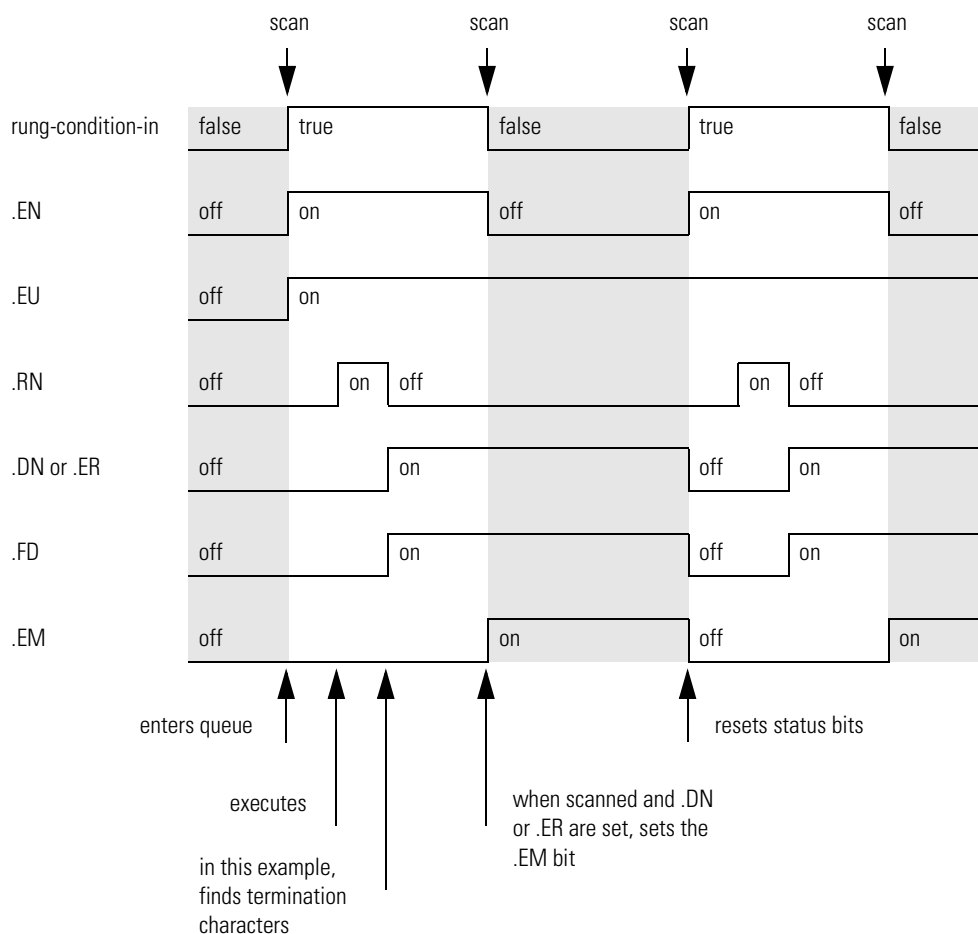
ASCII serial port instructions execute asynchronous to the scan of the logic:



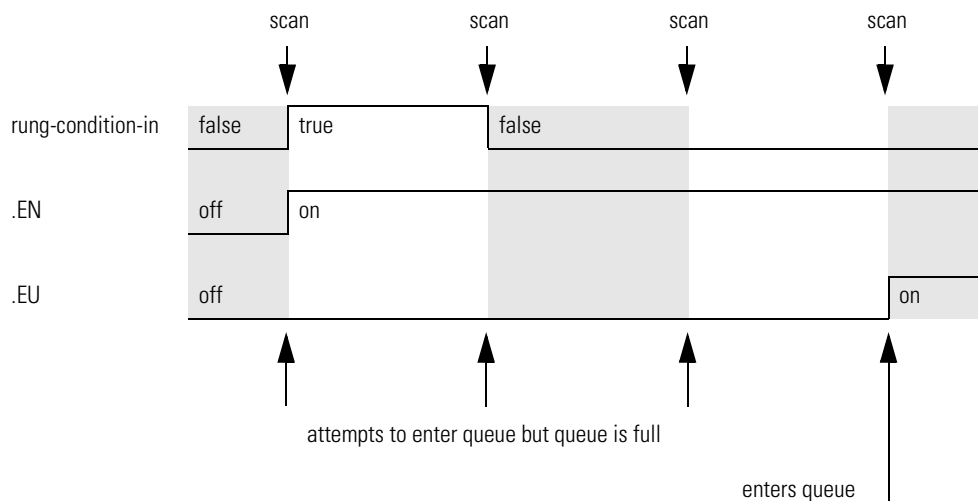
Each ASCII serial port instruction (except ACL) uses a SERIAL_PORT_CONTROL structure to perform the following functions:

- control the execution of the instruction
- provide status information about the instruction

The following timing diagram depicts the changes in the status bits as an ABL instruction tests the buffer for termination characters.



The ASCII queue holds up to 16 instructions. When the queue is full, an instruction tries to enter the queue on each subsequent scan of the instruction, as depicted below:



ASCII Error Codes

If an ASCII serial port instruction fails to execute, the ERROR member of its SERIAL_PORT_CONTROL structure will contain one of the following hexadecimal error codes:

This hex code:	Indicates that the:
16#2	Modem went offline.
16#3	CTS signal was lost during communication.
16#4	Serial port was in system mode.
16#A	Before the instruction executed, the .UL bit was set. This prevents the execution of the instruction.
16#C	The controller changed from Run mode to Program mode. This stops the execution of an ASCII serial port instruction and clears the queue.
16#D	In the Controller Properties dialog box, User Protocol tab, the buffer size or echo mode parameters were changed and applied. This stops the execution of an ASCII serial port instruction and clears the queue.
16#E	ACL instruction executed.
16#F	Serial port configuration changed from User mode to System mode. This stops the execution of an ASCII serial port instruction and clears the ASCII serial port instruction queue.
16#51	The LEN value of the string tag is either negative or greater than the DATA size of the string tag.
16#54	The Serial Port Control Length is greater than the size of the buffer.
16#55	The Serial Port Control Length is either negative or greater than the size of the Source or Destination.

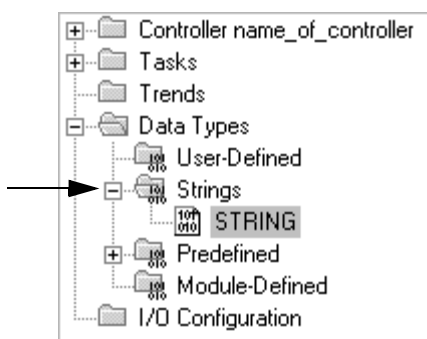
String Data Types

You store ASCII characters in tags that use a string data type.

- You can use the default STRING data type. It stores up to 82 characters.
- You can create a new string data type that stores less or more characters.

To create a new string data type, see *Logix5000 Controllers Common Procedures*, publication 1756-PM001.

Each string data type contains the following members:

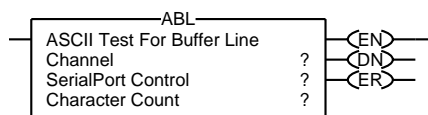


Name:	Data Type:	Description:	Notes:
LEN	DINT	number of characters in the string	<p>The LEN automatically updates to the new count of characters whenever you:</p> <ul style="list-style-type: none"> • use the String Browser dialog box to enter characters • use instructions that read, convert, or manipulate a string <p>The LEN shows the length of the current string. The DATA member may contain additional, old characters, which are not included in the LEN count.</p>
DATA	SINT array	ASCII characters of the string	<ul style="list-style-type: none"> • To access the characters of the string, address the name of the tag. For example, to access the characters of the <i>string_1</i> tag, enter <i>string_1</i>. • Each element of the DATA array contains one character. • You can create new string data types that store less or more characters.

ASCII Test For Buffer Line (ABL)

The ABL instruction counts the characters in the buffer up to and including the first termination character.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Channel	DINT	immediate tag	0
Serial Port Control	SERIAL_PORT_CONTROL	tag	tag that controls the operation
Character Count	DINT	immediate	0
			During execution, displays the number of characters in the buffer, including the first set of termination characters.



```
ABL(Channel
    SerialPortControl);
```

Structured Text

The operands are the same as those for the relay ladder ABL instruction. You access the Character Count value via the .POS member of the SERIAL_PORT_CONTROL structure.

SERIAL_PORT_CONTROL Structure

Mnemonic:	Data Type:	Description:
.EN	BOOL	The enable bit indicates that the instruction is enabled.
.EU	BOOL	The queue bit indicates that the instruction entered the ASCII queue.
.DN	BOOL	The done bit indicates when the instruction is done, but it is asynchronous to the logic scan.
.RN	BOOL	The run bit indicates that the instruction is executing.
.EM	BOOL	The empty bit indicates that the instruction is done, but it is synchronous to the logic scan.
.ER	BOOL	The error bit indicates when the instruction fails (errors).
.FD	BOOL	The found bit indicates that the instruction found the termination character or characters.
.POS	DINT	The position determines the number of characters in the buffer, up to and including the first set of termination characters. The instruction only returns this number after it finds the termination character or characters.
.ERROR	DINT	The error contains a hexadecimal value that identifies the cause of an error.

Description: The ABL instruction searches the buffer for the first set of termination characters. If the instruction finds the termination characters, it:

- sets the .FD bit
- counts the characters in the buffer up to and including the first set of termination characters

The Controller Properties dialog box, User Protocol tab, defines the ASCII characters that the instruction considers as the termination characters.

To program the ABL instruction, follow these guidelines:

1. Configure the serial port of the controller for user mode and define the characters that serve as the termination characters.
2. This is a transitional instruction:
 - In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.
 - In structured text, condition the instruction so that it only executes on a transition. See C.

Arithmetic Status Flags: not affected

Fault Conditions: none

Execution:

Condition:	Relay Ladder Action:	Structured Text Action:
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes when rung-condition-in toggles from cleared to set. The rung-condition-out is set to true.	na
EnableIn is set	na	EnableIn is always set. The instruction executes.
instruction execution	The instruction counts the characters in the buffer. The .EN bit is set. The remaining status bits, except .UL, are cleared. The instruction attempts to enter the ASCII queue.	
postscan	The rung-condition-out is set to false.	No action taken.

Example: Continuously test the buffer for the termination characters.

Relay Ladder



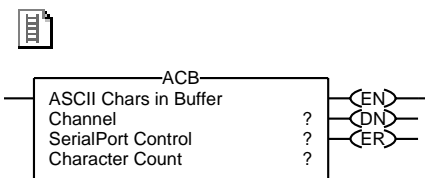
Structured Text

```
ABL(0,MV_line);
```

ASCII Chars in Buffer (ACB)

The ACB instruction counts the characters in the buffer.

Operands:



Relay Ladder

Operand:	Type:	Format:	Enter:
Channel	DINT	immediate tag	0
Serial Port Control	SERIAL_PORT_CONTROL	tag	tag that controls the operation
Character Count	DINT	immediate	0
During execution, displays the number of characters in the buffer.			



```
ACB (Channel
    SerialPortControl);
```

Structured Text

The operands are the same as those for the relay ladder ACB instruction. However, you specify the Character Count value by accessing the .POS member of the SERIAL_PORT_CONTROL structure, rather than by including the value in the operand list.

SERIAL_PORT_CONTROL Structure

Mnemonic:	Data Type:	Description:
.EN	BOOL	The enable bit indicates that the instruction is enabled.
.EU	BOOL	The queue bit indicates that the instruction entered the ASCII queue.
.DN	BOOL	The done bit indicates when the instruction is done, but it is asynchronous to the logic scan.
.RN	BOOL	The run bit indicates that the instruction is executing.
.EM	BOOL	The empty bit indicates that the instruction is done, but it is synchronous to the logic scan.
.ER	BOOL	The error bit indicates when the instruction fails (errors).
.FD	BOOL	The found bit indicates that the instruction found a character.
.POS	DINT	The position determines the number of characters in the buffer, up to and including the first set of termination characters.
.ERROR	DINT	The error contains a hexadecimal value that identifies the cause of an error.

Description: The ACB instruction counts the characters in the buffer.

To program the ACB instruction, follow these guidelines:

1. Configure the serial port of the controller for user mode.
2. This is a transitional instruction:
 - In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.
 - In structured text, condition the instruction so that it only executes on a transition. See C.

Arithmetic Status Flags: not affected

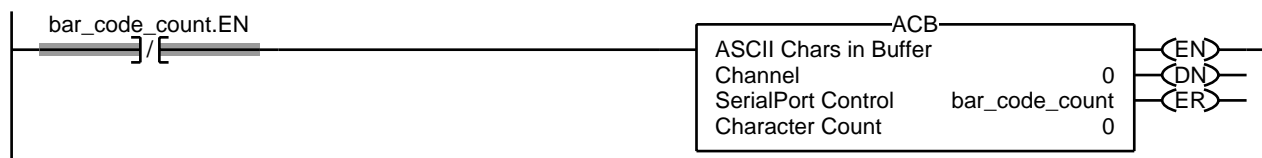
Fault Conditions: none

Execution:

Condition:	Relay Ladder Action:	Structured Text Action:
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes when rung-condition-in toggles from cleared to set. The rung-condition-out is set to true.	na
EnableIn is set	na	EnableIn is always set. The instruction executes.
instruction execution	The instruction counts the characters in the buffer. The .EN bit is set. The remaining status bits, except .UL, are cleared. The instruction attempts to enter the ASCII queue.	
postscan	The rung-condition-out is set to false.	No action taken.

Example: Continuously count the characters in the buffer.

Relay Ladder



Structured Text

```
ACB(0, bar_code_count);
```

ASCII Clear Buffer (ACL)

The ACL instruction immediately clears the buffer and ASCII queue.

Operands:



ACL	
ASCII Clear Buffer	?
Channel	?
Clear Serial Port Read	?
Clear Serial Port Write	?

Relay Ladder

Operand:	Type:	Format:	Enter:
Channel	DINT	immediate tag	0
Clear Serial Port Read	BOOL	immediate tag	To empty the buffer and remove ARD and ARL instructions from the queue, enter Yes.
Clear Serial Port Write	BOOL	immediate tag	To remove AWA and AWT instructions from the queue, enter Yes.



```
ACL (Channel,
    ClearSerialPortRead,
    ClearSerialPortWrite);
```

Structured Text

The operands are the same as those for the relay ladder ACL instruction.

Description: The ACL instruction immediately performs one or both of the following actions:

- clears the buffer of characters and clears the ASCII queue of read instructions
- clears the ASCII queue of write instructions

To program the ACL instruction, follow these guidelines:

1. Configure the serial port of the controller:

If your application:	Then:
uses ARD or ARL instructions	Select User mode
<i>does not</i> use ARD or ARL instructions	Select either System or User mode

2. To determine if an instruction was removed from the queue or aborted, examine the following of the appropriate instruction:
 - .ER bit is set
 - .ERROR member is 16#E

Arithmetic Status Flags: not affected

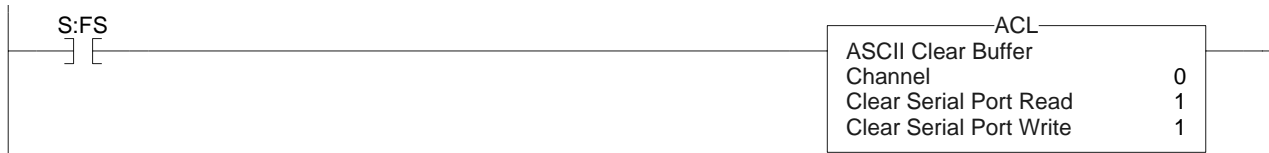
Fault Conditions: none

Execution:

Condition:	Relay Ladder Action:	Structured Text Action:
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	na
EnableIn is set	na	EnableIn is always set. The instruction executes.
instruction execution	The instruction clears the specified instructions and buffer(s).	
postscan	The rung-condition-out is set to false.	No action taken.

Example: When the controller enters Run mode, clear the buffer and the ASCII queue.

Relay Ladder



Structured Text

```

osri_1.InputBit := S:FS;
OSRI(osri_1);

IF (osri_1.OutputBit) THEN
    ACL(0,0,1);
END_IF;

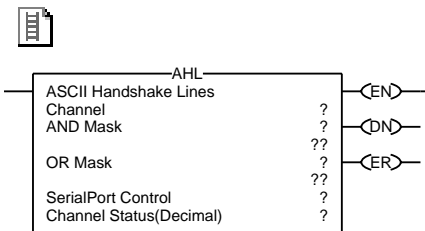
```

ASCII Handshake Lines (AHL)

The AHL instruction obtains the status of control lines and turns on or off the DTR and RTS signals.

Operands:

Relay Ladder



Operand:	Type:	Format:	Enter:
Channel	DINT	immediate tag	0
ANDMask	DINT	immediate tag	Refer to the description.
ORMask	DINT	immediate tag	
Serial Port Control	SERIAL_PORT_CONTROL	tag	tag that controls the operation
Channel Status (Decimal)	DINT	immediate	0

During execution, displays the status of the control lines.

For the status of this control line:	Examine this bit:
CTS	0
RTS	1
DSR	2
DCD	3
DTR	4
Received the XOFF character	5

Structured Text

```
AHL (Channel,ANDMask,ORMask,  
      SerialPortControl);
```

The operands are the same as those for the relay ladder AHL instruction. However, you specify the Channel Status value by accessing the .POS member of the SERIAL_PORT_CONTROL structure, rather than by including the value in the operand list.

SERIAL_PORT_CONTROL Structure

Mnemonic:	Data Type:	Description:
.EN	BOOL	The enable bit indicates that the instruction is enabled.
.EU	BOOL	The queue bit indicates that the instruction entered the ASCII queue.
.DN	BOOL	The done bit indicates when the instruction is done, but it is asynchronous to the logic scan.
.RN	BOOL	The run bit indicates that the instruction is executing.
.EM	BOOL	The empty bit indicates that the instruction is done, but it is synchronous to the logic scan.
.ER	BOOL	The error bit indicates when the instruction fails (errors).
.FD	BOOL	The found bit does not apply to this instruction.
.POS	DINT	The position stores the status of the control lines.
.ERROR	DINT	The error contains a hexadecimal value that identifies the cause of an error.

Description: The AHL instruction can:

- obtain the status of the control lines of the serial port
- turn on or off the data terminal ready (DTR) signal
- turn on or off the request to send signal (RTS)

To program the AHL instruction, follow these guidelines:

1. Configure the serial port of the controller:

If your application:	Then:
uses ARD or ARL instructions	Select User mode
<i>does not</i> use ARD or ARL instructions	Select either System or User mode

2. Use the following table to select the correct values for the ANDMask and ORMask operands:

To turn DTR:	And turn RTS:	Enter this ANDMask value:	And enter this ORMask value:
off	off	3	0
	on	1	2
	unchanged	1	0
on	off	2	1
	on	0	3
	unchanged	0	1
unchanged	off	2	0
	on	0	2
	unchanged	0	0

3. This is a transitional instruction:

- In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.
- In structured text, condition the instruction so that it only executes on a transition. See C.

Arithmetic Status Flags: not affected

Fault Conditions:

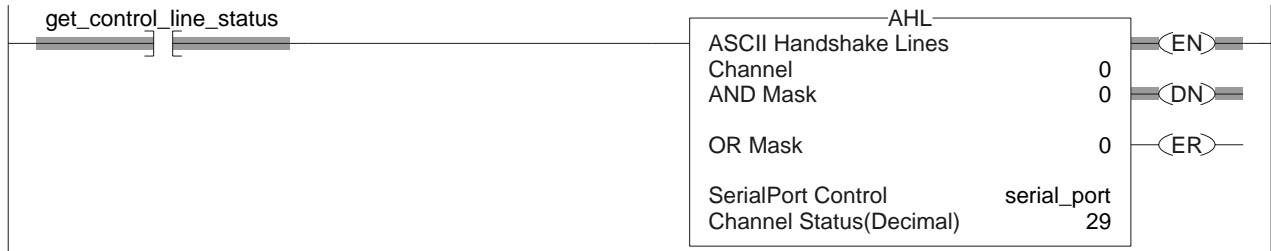
Type:	Code:	Cause:	Recovery Method:
4	57	The AHL instruction failed to execute because the serial port is set to no handshaking.	Either: <ul style="list-style-type: none"> • Change the Control Line setting of the serial port. • Delete the AHL instruction.

Execution:

Condition:	Relay Ladder Action:	Structured Text Action:
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes when rung-condition-in toggles from cleared to set. The rung-condition-out is set to true.	na
EnableIn is set	na	EnableIn is always set. The instruction executes.
instruction execution	The instruction obtains the control line status and turns on or off DTR and RTS signals. The .EN bit is set. The remaining status bits, except .UL, are cleared. The instruction attempts to enter the ASCII queue.	
postscan	The rung-condition-out is set to false.	No action taken.

Example: When *get_control_line_status* becomes set, obtain the status of the control lines of the serial port and store the status in the Channel Status operand. To view the status of a specific control line, monitor the SerialPortControl tag and expand the POS member.

Relay Ladder



Structured Text

```
osri_1.InputBit := get_control_line_status;
OSRI(osri_1);

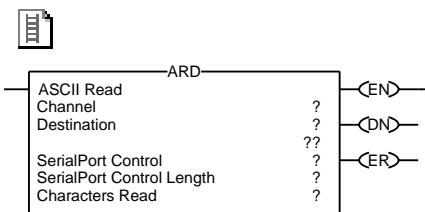
IF (osri_1.OutputBit) THEN
    AHL(0,0,0,serial_port);
END_IF;
```

ASCII Read (ARD)

The ARD instruction removes characters from the buffer and stores them in the Destination.

Operands:

Relay Ladder



Operand:	Type:	Format:	Enter:	Notes:
Channel	DINT	immediate tag	0	
Destination	string SINT INT DINT	tag	tag into which the characters are moved (read): <ul style="list-style-type: none">For a string data type, enter the name of the tag.For a SINT, INT, or DINT array, enter the first element of the array.	<ul style="list-style-type: none">If you want to compare, convert, or manipulate the characters, use a string data type.String data types are:<ul style="list-style-type: none">default STRING data typeany new string data type that you create
Serial Port Control	SERIAL_PORT_ CONTROL	tag	tag that controls the operation	
Serial Port Control Length	DINT	immediate	number of characters to move to the destination (read)	<ul style="list-style-type: none">The Serial Port Control Length must be less than or equal to the size of the Destination.If you want to set the Serial Port Control Length equal to the size of the Destination, enter 0.
Characters Read	DINT	immediate	0	During execution, displays the number of characters that were read.

Structured Text

```
ARD(Channel, Destination,  
SerialPortControl);
```

The operands are the same as those for the relay ladder ARD instruction. However, you specify the Serial Port Control Length and the Characters Read values by accessing the .LEN and .POS members of the SERIAL_PORT_CONTROL structure, rather than by including the values in the operand list.

SERIAL_PORT_CONTROL Structure

Mnemonic:	Data Type:	Description:
.EN	BOOL	The enable bit indicates that the instruction is enabled.
.EU	BOOL	The queue bit indicates that the instruction entered the ASCII queue.
.DN	BOOL	The done bit indicates when the instruction is done, but it is asynchronous to the logic scan.
.RN	BOOL	The run bit indicates that the instruction is executing.
.EM	BOOL	The empty bit indicates that the instruction is done, but it is synchronous to the logic scan.
.ER	BOOL	The error bit indicates when the instruction fails (errors).
.FD	BOOL	The found bit does not apply to this instruction.
.LEN	DINT	The length indicates the number of characters to move to the destination (read).
.POS	DINT	The position displays the number of characters that were read.
.ERROR	DINT	The error contains a hexadecimal value that identifies the cause of an error.

Description: The ARD instruction removes the specified number of characters from the buffer and stores them in the Destination.

- The ARD instruction continues to execute until it removes the specified number of characters (Serial Port Control Length).
- While the ARD instruction is executing, no other ASCII Serial Port instruction executes.

To program the ARD instruction, follow these guidelines:

1. Configure the serial port of the controller for user mode.
2. Use the results of an ACB instruction to trigger the ARD instruction. This prevents the ARD instruction from holding up the ASCII queue while it waits for the required number of characters.
3. This is a transitional instruction:
 - In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.
 - In structured text, condition the instruction so that it only executes on a transition. See C.
4. To trigger a subsequent action when the instruction is done, examine the EM bit.

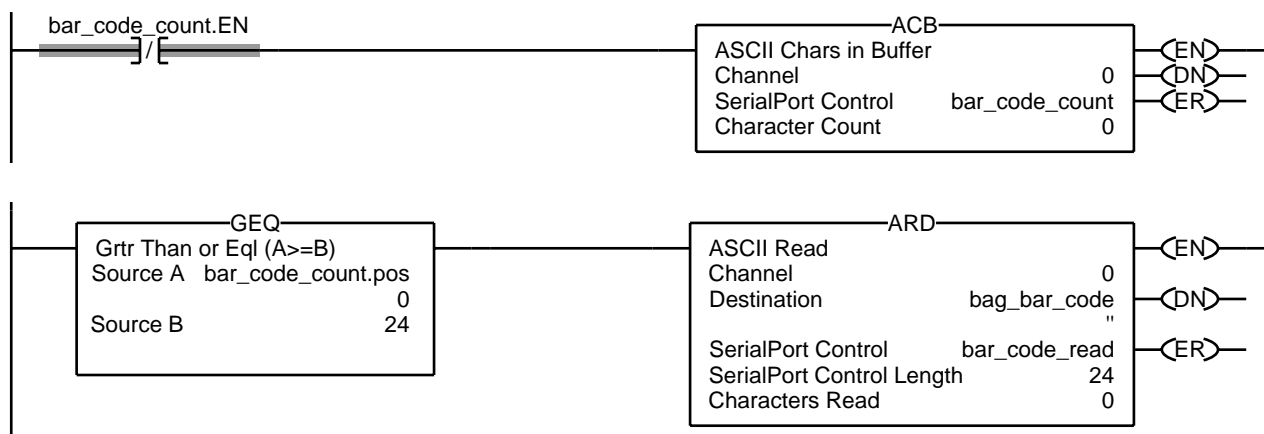
Arithmetic Status Flags: not affected

Fault Conditions: none

Execution:

Condition:	Relay Ladder Action:	Structured Text Action:
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes when rung-condition-in toggles from cleared to set. The rung-condition-out is set to true.	na
EnableIn is set	na	EnableIn is always set. The instruction executes.
instruction execution	The instruction removes characters from the buffer and stores them in the destination. The .EN bit is set. The remaining status bits, except .UL, are cleared. The instruction attempts to enter the ASCII queue.	
postscan	The rung-condition-out is set to false.	No action taken.

Example: A bar code reader sends bar codes to the serial port (channel 0) of the controller. Each bar code contains 24 characters. To determine when the controller receives a bar code, the ACB instruction continuously counts the characters in the buffer. When the buffer contains at least 24 characters, the controller has received a bar code. The ARD instruction moves the bar code to the DATA member of the *bag_bar_code* tag, which is a string.

Relay Ladder**Structured Text**

```

ACB(0,bar_code_count);

IF bar_code_count.POS >= 24 THEN
    bar_code_read.LEN := 24;
    ARD(0,bag_bar_code,bar_code_read);
END_IF;

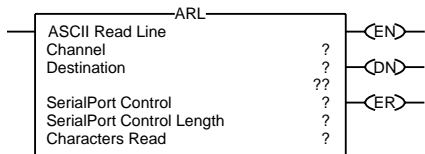
```

ASCII Read Line (ARL)

The ARL instruction removes specified characters from the buffer and stores them in the Destination.

Operands:

Relay Ladder



Operand:	Type:	Format:	Enter:	Notes:
Channel	DINT	immediate tag	0	
Destination	string SINT INT DINT	tag	tag into which the characters are moved (read): <ul style="list-style-type: none"> For a string data type, enter the name of the tag. For a SINT, INT, or DINT array, enter the first element of the array. 	<ul style="list-style-type: none"> If you want to compare, convert, or manipulate the characters, use a string data type. String data types are: <ul style="list-style-type: none"> default STRING data type any new string data type that you create
Serial Port Control	SERIAL_PORT_CONTROL	tag	tag that controls the operation	
Serial Port Control Length	DINT	immediate	maximum number of characters to read if no termination characters are found	<ul style="list-style-type: none"> Enter the maximum number of characters that any message will contain (i.e., when to stop reading if no termination characters are found). <p>For example, if messages range from 3 to 6 characters in length, enter 6.</p> <ul style="list-style-type: none"> The Serial Port Control Length must be less than or equal to the size of the Destination. If you want to set the Serial Port Control Length equal to the size of the Destination, enter 0.
Characters Read	DINT	immediate	0	During execution, displays the number of characters that were read.



Structured Text

```
ARL(Channel, Destination,
     SerialPortControl);
```

The operands are the same as those for the relay ladder ARL instruction. However, you specify the Serial Port Control Length and the Characters Read values by accessing the .LEN and .POS members of the SERIAL_PORT_CONTROL structure, rather than by including the values in the operand list.

SERIAL_PORT_CONTROL Structure

Mnemonic:	Data Type:	Description:
.EN	BOOL	The enable bit indicates that the instruction is enabled.
.EU	BOOL	The queue bit indicates that the instruction entered the ASCII queue.
.DN	BOOL	The done bit indicates when the instruction is done, but it is asynchronous to the logic scan.
.RN	BOOL	The run bit indicates that the instruction is executing.
.EM	BOOL	The empty bit indicates that the instruction is done, but it is synchronous to the logic scan.
.ER	BOOL	The error bit indicates when the instruction fails (errors).
.FD	BOOL	The found bit does not apply to this instruction.
.LEN	DINT	The length indicates the maximum number of characters to move to the destination (i.e., when to stop reading if no termination characters are found).
.POS	DINT	The position displays the number of characters that were read.
.ERROR	DINT	The error contains a hexadecimal value that identifies the cause of an error.

Description: The ARL instruction removes characters from the buffer and stores them in the Destination as follows:

- The ARL instruction continues to execute until it removes either the:
 - first set of termination characters
 - specified number of characters (Serial Port Control Length)
- While the ARL instruction is executing, no other ASCII serial port instruction executes.

To program the ARL instruction, follow these guidelines:

1. Configure the serial port of the controller:
 - a. Select User mode.
 - b. Define the characters that serve as the termination characters.
2. Use the results of an ABL instruction to trigger the ARL instruction. This prevents the ARL instruction from holding up the ASCII queue while it waits for the termination characters.
3. This is a transitional instruction:
 - In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.
 - In structured text, condition the instruction so that it only executes on a transition. See C.
4. To trigger a subsequent action when the instruction is done, examine the EM bit.

Arithmetic Status Flags: not affected

Fault Conditions: none

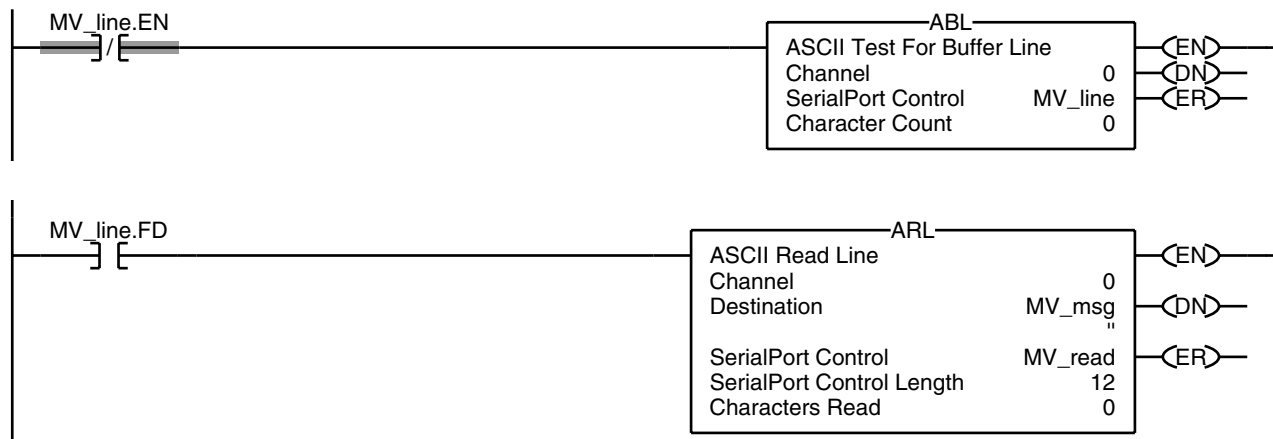
Execution:

Condition:	Relay Ladder Action:	Structured Text Action:
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes when rung-condition-in toggles from cleared to set. The rung-condition-out is set to true.	na
EnableIn is set	na	EnableIn is always set. The instruction executes.
instruction execution	The instruction removes the specified characters from the buffer and stores them in the destination. The .EN bit is set. The remaining status bits, except .UL, are cleared. The instruction attempts to enter the ASCII queue.	
postscan	The rung-condition-out is set to false.	No action taken.

Example: Continuously test the buffer for a message from a MessageView terminal. Since each message ends in a carriage return (\$r), the carriage return is configured as the termination character in the Controller Properties dialog box, User Protocol tab. When the ABL finds a carriage return, it sets the FD bit.

When the ABL instruction finds the carriage return (*MV_line.FD* is set), the controller has received a complete message. The ARL instruction removes the characters from the buffer, up to and including the carriage return, and places them in the DATA member of the *MV_msg* tag, which is a string.

Relay Ladder



Structured Text

```

ABL(0,MV_line);

osri_1.InputBit := MVLine.FD;
OSRI(osri_1);

IF (osri_1.OutputBit) THEN
    mv_read.LEN := 12;
    ARL(0,MV_msg,MV_read);
END_IF;

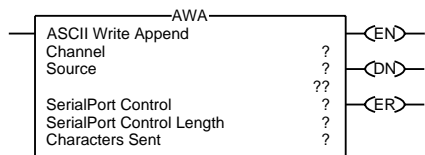
```

ASCII Write Append (AWA)

The AWA instruction sends a specified number of characters of the Source tag to a serial device and appends either one or two predefined characters.

Operands:

Relay Ladder



Operand:	Type:	Format:	Enter:	Notes:
Channel	DINT	immediate tag	0	
Source	string SINT INT DINT	tag	tag that contains the characters to send: <ul style="list-style-type: none"> For a string data type, enter the name of the tag. For a SINT, INT, or DINT array, enter the first element of the array. 	<ul style="list-style-type: none"> If you want to compare, convert, or manipulate the characters, use a string data type. String data types are: <ul style="list-style-type: none"> default STRING data type any new string data type that you create
Serial Port Control	SERIAL_PORT_CONTROL	tag	tag that controls the operation	
Serial Port Control Length	DINT	immediate	number of characters to send	<ul style="list-style-type: none"> The Serial Port Control Length must be less than or equal to the size of the Source. If you want to set the Serial Port Control Length equal to the number of characters in the Source, enter 0.
Characters Sent	DINT	immediate	0	During execution, displays the number of characters that were sent.



Structured Text

```
AWA (Channel, Source,
     SerialPortControl);
```

The operands are the same as those for the relay ladder AWA instruction. However, you specify the Serial Port Control Length and the Characters Sent values by accessing the .LEN and .POS members of the SERIAL_PORT_CONTROL structure, rather than by including the values in the operand list.

SERIAL_PORT_CONTROL Structure

Mnemonic:	Data Type:	Description:
.EN	BOOL	The enable bit indicates that the instruction is enabled.
.EU	BOOL	The queue bit indicates that the instruction entered the ASCII queue.
.DN	BOOL	The done bit indicates when the instruction is done, but it is asynchronous to the logic scan.
.RN	BOOL	The run bit indicates that the instruction is executing.
.EM	BOOL	The empty bit indicates that the instruction is done, but it is synchronous to the logic scan.
.ER	BOOL	The error bit indicates when the instruction fails (errors).
.FD	BOOL	The found bit does not apply to this instruction.
.LEN	DINT	The length indicates the number of characters to send.
.POS	DINT	The position displays the number of characters that were sent.
.ERROR	DINT	The error contains a hexadecimal value that identifies the cause of an error.

Description: The AWA instruction:

- sends the specified number of characters (Serial Port Control Length) of the Source tag to the device that is connected to the serial port of the controller
- adds to the end of the characters (appends) either one or two characters that are defined in the Controller Properties dialog box, User Protocol tab

To program the AWA instruction, follow these guidelines:

1. Configure the serial port of the controller:

a. Does your application also include ARD or ARL instructions?

If:	Then:
Yes	Select User mode
No	Select either System or User mode

b. Define the characters to append to the data.

2. This is a transitional instruction:

- In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.
- In structured text, condition the instruction so that it only executes on a transition. See C.

3. Each time the instruction executes, do you always send the same number of characters?

If:	Then:
Yes	In the Serial Port Control Length, enter the number of characters to send.
No	Before the instruction executes, set the LEN member of the Source tag to the LEN member of the Serial Port Control tag.

Arithmetic Status Flags: not affected

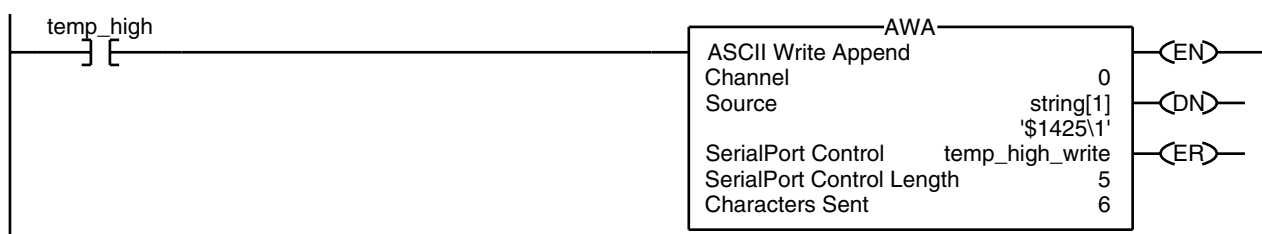
Fault Conditions: none

Execution:

Condition:	Relay Ladder Action:	Structured Text Action:
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes when rung-condition-in toggles from cleared to set. The rung-condition-out is set to true.	na
EnableIn is set	na	EnableIn is always set. The instruction executes.
instruction execution	The instruction sends a specified number of characters and appends either one or two predefined characters. The .EN bit is set. The remaining status bits, except .UL, are cleared. The instruction attempts to enter the ASCII queue.	
postscan	The rung-condition-out is set to false.	No action taken.

Example 1: When the temperature exceeds the high limit (*temp_high* is set), the AWA instruction sends a message to a MessageView terminal that is connected to the serial port of the controller. The message contains five characters from the DATA member of the *string[1]* tag, which is a string. (The *\$14* counts as one character. It is the hex code for the Ctrl-T character.) The instruction also sends (appends) the characters defined in the controller properties. In this example, the AWA instruction sends a carriage return (\$0D), which marks the end of the message.

Relay Ladder



Structured Text

```
IF temp_high THEN

    temp_high_write.LEN := 5;

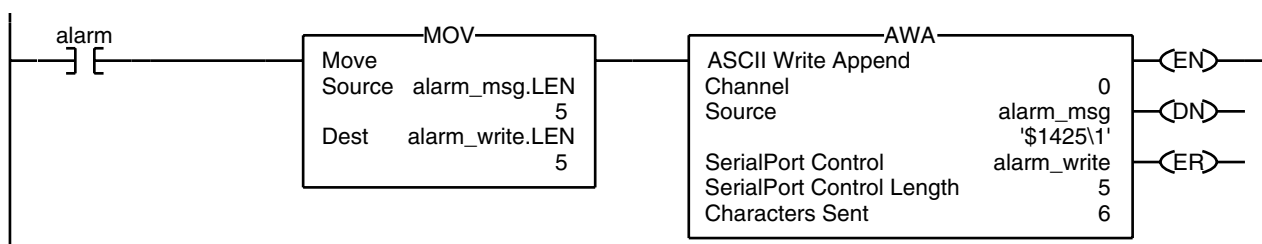
    AWA(0,string[1],temp_high_write);

    temp_high := 0;

END_IF;
```

Example 2: When *alarm* is set, the AWA instruction sends the specified number of characters in *alarm_msg* and appends a termination character (s). Because the number of characters in *alarm_msg* varies, the rung first moves the length of the string (*alarm_msg.LEN*) to the Serial Port Control Length of the AWA instruction (*alarm_write.LEN*). In *alarm_msg*, the \$14 counts as one character. It is the hex code for the Ctrl-T character.

Relay Ladder



Structured Text

```
osri_1.InputBit := alarm;
OSRI(osri_1);

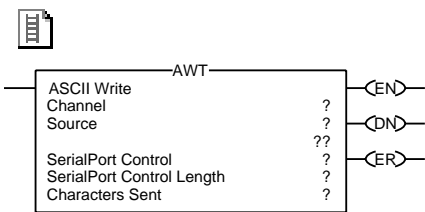
IF (osri_1.OutputBit) THEN
    alarm_write.LEN := alarm_msg.LEN;
    AWA(0,alarm_msg,alarm_write);
END_IF;
```

ASCII Write (AWT)

The AWT instruction sends a specified number of characters of the Source tag to a serial device.

Operands:

Relay Ladder



Operand:	Type:	Format:	Enter:	Notes:
Channel	DINT	immediate tag	0	
Source	string SINT INT DINT	tag	tag that contains the characters to send: <ul style="list-style-type: none"> For a string data type, enter the name of the tag. For a SINT, INT, or DINT array, enter the first element of the array. 	<ul style="list-style-type: none"> If you want to compare, convert, or manipulate the characters, use a string data type. String data types are: <ul style="list-style-type: none"> default STRING data type any new string data type that you create
Serial Port Control	SERIAL_PORT_CONTROL	tag	tag that controls the operation	
Serial Port Control Length	DINT	immediate	number of characters to send	<ul style="list-style-type: none"> The Serial Port Control Length must be less than or equal to the size of the Source. If you want to set the Serial Port Control Length equal to the number of characters in the Source, enter 0.
Characters Sent	DINT	immediate	0	During execution, displays the number of characters that were sent.

Structured Text

```

AWT (Channel, Source,
      SerialPortControl);

```

The operands are the same as those for the relay ladder AWT instruction. However, you specify the Serial Port Control Length and the Characters Sent values by accessing the .LEN and .POS members of the SERIAL_PORT_CONTROL structure, rather than by including the values in the operand list

SERIAL_PORT_CONTROL Structure

Mnemonic:	Data Type:	Description:
.EN	BOOL	The enable bit indicates that the instruction is enabled.
.EU	BOOL	The queue bit indicates that the instruction entered the ASCII queue.
.DN	BOOL	The done bit indicates when the instruction is done, but it is asynchronous to the logic scan.
.RN	BOOL	The run bit indicates that the instruction is executing.
.EM	BOOL	The empty bit indicates that the instruction is done, but it is synchronous to the logic scan.
.ER	BOOL	The error bit indicates when the instruction fails (errors).
.FD	BOOL	The found bit does not apply to this instruction.
.LEN	DINT	The length indicates the number of characters to send.
.POS	DINT	The position displays the number of characters that were sent.
.ERROR	DINT	The error contains a hexadecimal value that identifies the cause of an error.

Description: The AWT instruction sends the specified number of characters (Serial Port Control Length) of the Source tag to the device that is connected to the serial port of the controller.

To program the AWT instruction, follow these guidelines:

1. Configure the serial port of the controller:

If your application:	Then:
uses ARD or ARL instructions	Select User mode
<i>does not</i> use ARD or ARL instructions	Select either System or User mode

2. This is a transitional instruction:
 - In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.
 - In structured text, condition the instruction so that it only executes on a transition. See C.
3. Each time the instruction executes, do you always send the same number of characters?

If:	Then:
Yes	In the Serial Port Control Length, enter the number of characters to send.
No	Before the instruction executes, move the LEN member of the Source tag to the LEN member of the Serial Port Control tag.

Arithmetic Status Flags: not affected

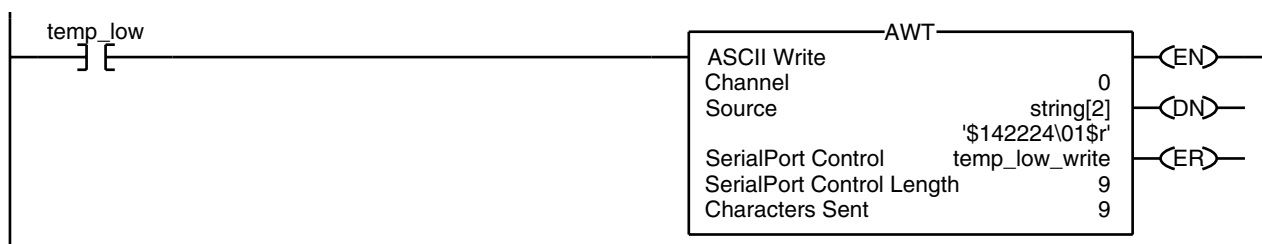
Fault Conditions: none

Execution:

Condition:	Relay Ladder Action:	Structured Text Action:
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes when rung-condition-in toggles from cleared to set. The rung-condition-out is set to true.	na
EnableIn is set	na	EnableIn is always set. The instruction executes.
instruction execution	The instruction sends a specified number of characters. The .EN bit is set. The remaining status bits, except .UL, are cleared. The instruction attempts to enter the ASCII queue.	
postscan	The rung-condition-out is set to false.	No action taken.

Example 1: When the temperature reaches the low limit (*temp_low* is set), the AWT instruction sends a message to the MessageView terminal that is connected to the serial port of the controller. The message contains nine characters from the DATA member of the *string[2]* tag, which is a string. (The *\$14* counts as one character. It is the hex code for the Ctrl-T character.) The last character is a carriage return (\$r), which marks the end of the message.

Relay Ladder



Structured Text

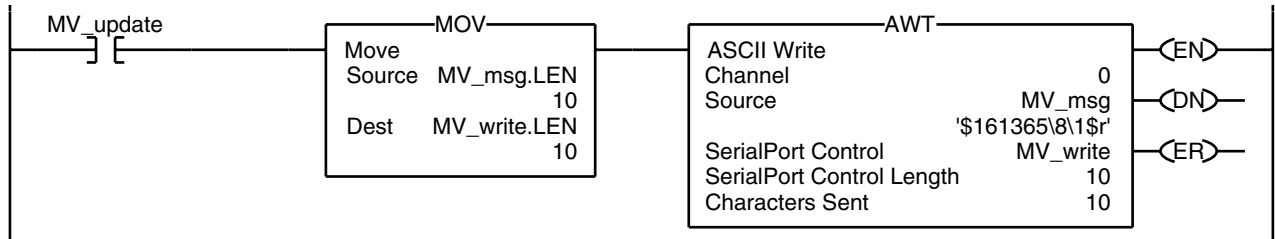
```

osri_1.InputBit := temp_low;
OSRI(osri_1);
IF (osri_1.OutputBit) THEN
    temp_low_write.LEN := 9;
    AWT(0,string[2],temp_low_write);
END_IF;

```

Example 2: When *MV_update* is set, the AWT instruction sends the characters in *MV_msg*. Because the number of characters in *MV_msg* varies, the rung first moves the length of the string (*MV_msg.LEN*) to the Serial Port Control Length of the AWT instruction (*MV_write.LEN*). In *MV_msg*, the *\$16* counts as one character. It is the hex code for the Ctrl-V character.

Relay Ladder



Structured Text

```

osri_1.InputBit := MV_update;
OSRI(osri_1);

IF (osri_1.OutputBit) THEN

    MV_write.LEN := Mv_msg.LEN;

    AWT(0,MV_msg,MV_write);

END_IF;

```

Notes:

ASCII String Instructions (CONCAT, DELETE, FIND, INSERT, MID)

Introduction

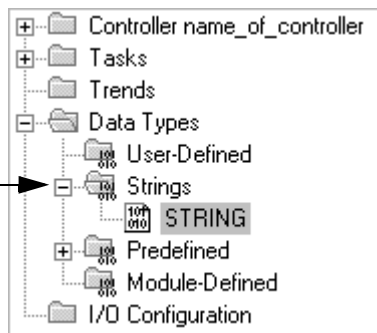
Use the ASCII string instructions to modify and create strings of ASCII characters.

If you want to:	For example:	Use this instruction:	Available in these languages:	See page:
add characters to the end of a string	add termination characters or delimiters to a string	CONCAT	relay ladder structured text	17-3
delete characters from a string	remove header or control characters from a string	DELETE	relay ladder structured text	17-5
determine the starting character of a sub-string	locate a group of characters within a string	FIND	relay ladder structured text	17-7
insert characters into a string	create a string that uses variables	INSERT	relay ladder structured text	17-9
extract characters from a string	extract information from a bar code	MID	relay ladder structured text	17-11

You can also use the following instructions to compare or convert ASCII characters:

If you want to:	Use this instruction:	See page:
compare a string to another string	CMP	4-2
see if the characters are equal to specific characters	EQU	4-7
see if the characters are not equal to specific characters	NEQ	4-38
see if the characters are equal to or greater than specific characters	GEQ	4-11
see if the characters are greater than specific characters	GRT	4-15
see if the characters are equal to or less than specific characters	LEQ	4-19
see if the characters are less than specific characters	LES	4-23
rearrange the bytes of a INT, DINT, or REAL tag	SWPB	6-19
find a string in an array of strings	FSC	7-19
convert characters to a SINT, INT, DINT, or REAL value	STOD	18-4
convert characters to a REAL value	STOR	18-6
convert a SINT, INT, DINT, or REAL value to a string of ASCII characters	DTOS	18-8
convert REAL value to a string of ASCII characters	RTOS	18-10

String Data Types



You store ASCII characters in tags that use a string data type.

- You can use the default STRING data type. It stores up to 82 characters.
- You can create a new string data type that stores less or more characters.

To create a new string data type, see *Logix5000 Controllers Common Procedures*, publication 1756-PM001.

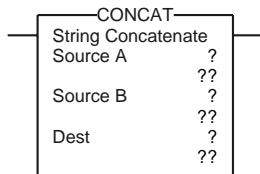
Each string data type contains the following members:

Name:	Data Type:	Description:	Notes:
LEN	DINT	number of characters in the string	<p>The LEN automatically updates to the new count of characters whenever you:</p> <ul style="list-style-type: none"> • use the String Browser dialog box to enter characters • use instructions that read, convert, or manipulate a string <p>The LEN shows the length of the current string. The DATA member may contain additional, old characters, which are not included in the LEN count.</p>
DATA	SINT array	ASCII characters of the string	<ul style="list-style-type: none"> • To access the characters of the string, address the name of the tag. For example, to access the characters of the <i>string_1</i> tag, enter <i>string_1</i>. • Each element of the DATA array contains one character. • You can create new string data types that store less or more characters.

String Concatenate (CONCAT)

The CONCAT instruction adds ASCII characters to the end of a string.

Operands:



Relay Ladder

Operand:	Type:	Format:	Enter:	Notes:
Source A	string	tag	tag that contains the initial characters	String data types are: <ul style="list-style-type: none"> • default STRING data type • any new string data type that you create
Source B	string	tag	tag that contains the end characters	
Destination	string	tag	tag to store the result	



```
CONCAT (SourceA, SourceB,
        Dest);
```

Structured Text

The operands are the same as those for the relay ladder CONCAT instruction.

Description: The CONCAT instruction combines the characters in Source A with the characters in Source B and places the result in the Destination.

- The characters from Source A are first, followed by the characters from Source B.
- Unless Source A and the Destination are the same tag, Source A remains unchanged.

Arithmetic Status Flags: not affected

Fault Conditions:

Type:	Code:	Cause:	Recovery Method:
4	51	The LEN value of the string tag is greater than the DATA size of the string tag.	<ol style="list-style-type: none"> 1. Check that no instruction is writing to the LEN member of the string tag. 2. In the LEN value, enter the number of characters that the string contains.

Execution:

Condition:	Relay Ladder Action:	Structured Text Action:
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	na
EnableIn is set	na	EnableIn is always set. The instruction executes.
instruction execution	The instruction concatenates the strings.	
postscan	The rung-condition-out is set to false.	No action taken.

Example: To trigger a message in a MessageView terminal, the controller must send an ASCII string that contains a message number and node number. *String_1* contains the message number. When *add_node* is set, the CONCAT instruction adds the characters in *node_num_ascii* (node number) to the end of the characters in *string_1* and then stores the result in *msg*.

Relay Ladder



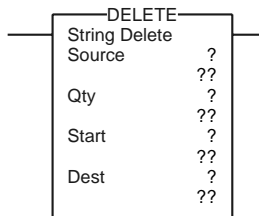
Structured Text

```
IF add_node THEN  
    CONCAT(string_1,node_num_ascii,msg);  
    add_node := 0;  
END_IF;
```

String Delete (DELETE)

The DELETE instruction removes ASCII characters from a string.

Operands:



Relay Ladder

Operand:	Type:	Format:	Enter:	Notes:
Source	string	tag	tag that contains the string from which you want to delete characters	String data types are: <ul style="list-style-type: none"> • default STRING data type • any new string data type that you create
Quantity	SINT INT DINT	immediate tag	number of characters to delete	The Start plus the Quantity must be less than or equal to the DATA size of the Source.
Start	SINT INT DINT	immediate tag	position of the first character to delete	Enter a number between 1 and the DATA size of the Source.
Destination	string	tag	tag to store the result	



Structured Text

```
DELETE (Source, Qty, Start,
        Dest);
```

The operands are the same as those for the relay ladder DELETE instruction.

Description: The DELETE instruction deletes (removes) a group of characters from the Source and places the remaining characters in the Destination.

- The Start position and Quantity define the characters to remove.
- Unless the Source and Destination are the same tag, the Source remains unchanged.

Arithmetic Status Flags: not affected

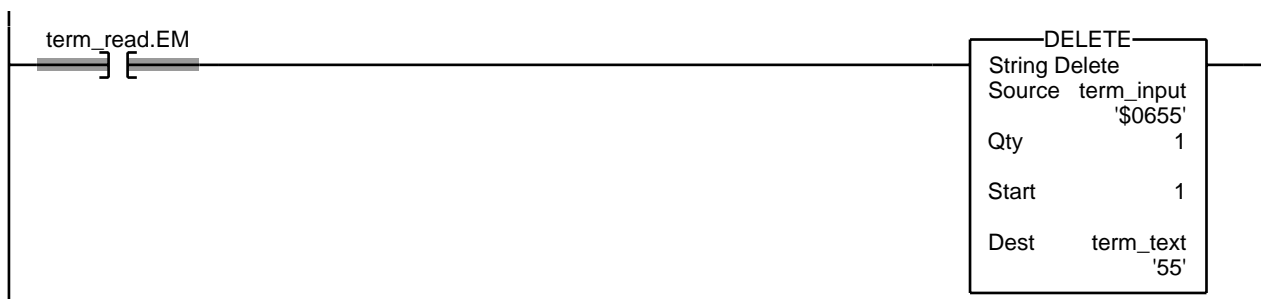
Fault Conditions:

Type:	Code:	Cause:	Recovery Method:
4	51	The LEN value of the string tag is greater than the DATA size of the string tag.	<ol style="list-style-type: none"> 1. Check that no instruction is writing to the LEN member of the string tag. 2. In the LEN value, enter the number of characters that the string contains.
4	56	The Start or Quantity value is invalid.	<ol style="list-style-type: none"> 1. Check that the Start value is between 1 and the DATA size of the Source. 2. Check that the Start value plus the Quantity value is less than or equal to the DATA size of the Source.

Execution:

Condition:	Relay Ladder Action:	Structured Text Action:
prescan	The rung-condition-out is set to false.	No action taken
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	na
EnableIn is set	na	EnableIn is always set. The instruction executes.
instruction execution	The instruction deletes the specified characters.	
postscan	The rung-condition-out is set to false.	No action taken.

Example: ASCII information from a terminal contains a header character. After the controller reads the data (*term_read.EM* is set) the DELETE instruction removes the header character.

Relay Ladder**Structured Text**

```

IF term_read.EM THEN

    DELETE (term_input, 1, 1, term_text);

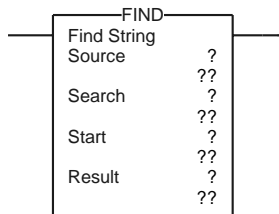
    term_read.EM := 0;

END_IF;
  
```

Find String (FIND)

The FIND instruction locates the starting position of a specified string within another string

Operands:



Relay Ladder

Operand:	Type:	Format:	Enter:	Notes:
Source	string	tag	string to search in	String data types are: <ul style="list-style-type: none"> • default STRING data type • any new string data type that you create
Search	string	tag	string to find	
Start	SINT INT DINT	immediate tag	position in Source to start the search	Enter a number between 1 and the DATA size of the Source.
Result	SINT INT DINT	tag	tag that stores the starting position of the string to find	



Structured Text

```
FIND(Source,Search,Start,
      Result);
```

The operands are the same as those for the relay ladder FIND instruction described above.

Description: The FIND instruction searches the Source string for the Search string. If the instruction finds the Search string, the Result shows the starting position of the Search string within the Source string.

Arithmetic Status Flags: not affected

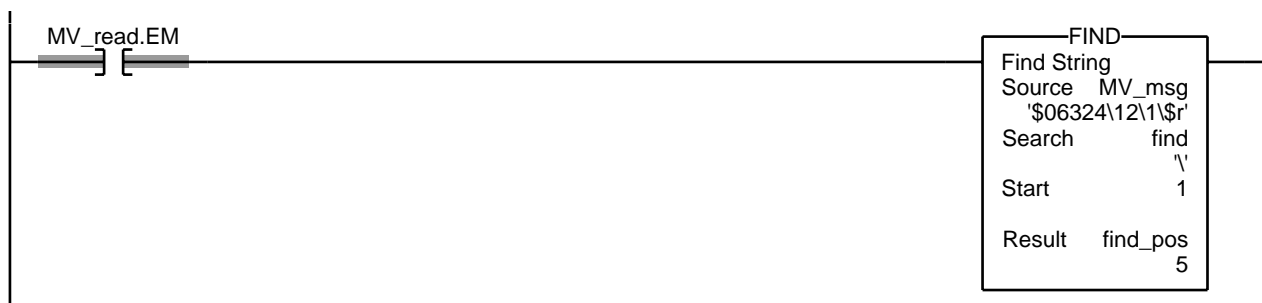
Fault Conditions:

Type:	Code:	Cause:	Recovery Method:
4	51	The LEN value of the string tag is greater than the DATA size of the string tag.	1. Check that no instruction is writing to the LEN member of the string tag. 2. In the LEN value, enter the number of characters that the string contains.
4	56	The Start value is invalid.	Check that the Start value is between 1 and the DATA size of the Source.

Execution:

Condition:	Relay Ladder Action:	Structured Text Action:
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	na
EnableIn is set	na	EnableIn is always set. The instruction executes.
instruction execution	The instruction searches for the specified characters.	
postscan	The rung-condition-out is set to false.	No action taken.

Example: A message from a MessageView terminal contains several pieces of information. The backslash character [\] separates each piece of information. To locate a piece of information, the FIND instruction searches for the backslash character and records its position in *find_pos*.

Relay Ladder**Structured Text**

```

IF MV_read.EM THEN

    FIND(MV_msg, find, 1, find_pos);

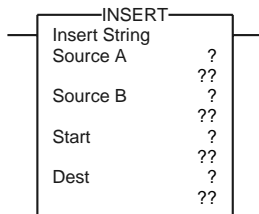
    MV_read.EM := 0;

END_IF;
  
```


Insert String (INSERT)

The INSERT instruction adds ASCII characters to a specified location within a string.

Operands:



Relay Ladder

Operand:	Type:	Format:	Enter:	Notes:
Source A	string	tag	string to add the characters to	String data types are: <ul style="list-style-type: none"> • default STRING data type • any new string data type that you create
Source B	string	tag	string containing the characters to add	
Start	SINT INT DINT	immediate tag	position in Source A to add the characters	Enter a number between 1 and the DATA size of the Source.
Result	string	tag	string to store the result	



```
INSERT (SourceA, SourceB,
      Start, Dest);
```

Structured Text

The operands are the same as those for the relay ladder INSERT instruction.

Description: The INSERT instruction adds the characters in Source B to a designated position within Source A and places the result in the Destination:

- Start defines where in Source A that Source B is added.
- Unless SourceA and the Destination are the same tag, Source A remains unchanged.

Arithmetic Status Flags: not affected

Fault Conditions:

Type:	Code:	Cause:	Recovery Method:
4	51	The LEN value of the string tag is greater than the DATA size of the string tag.	1. Check that no instruction is writing to the LEN member of the string tag. 2. In the LEN value, enter the number of characters that the string contains.
4	56	The Start value is invalid.	Check that the Start value is between 1 and the DATA size of the Source.

Execution:

Condition:	Relay Ladder Action:	Structured Text Action:
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	na
EnableIn is set	na	EnableIn is always set. The instruction executes.
instruction execution	The instruction inserts the specified characters.	
postscan	The rung-condition-out is set to false.	No action taken.

Example: When *temp_high* is set, the INSERT instruction adds the characters in *string_2* to position 2 within *string_1* and places the result in *string_3*:

Relay Ladder



Structured Text

```
IF temp_high THEN

    INSERT(string_1,string_2,2,string_3);

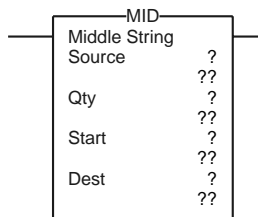
    temp_high := 0;

END_IF;
```

Middle String (MID)

The MID instruction copies a specified number of ASCII characters from a string and stores them in another string.

Operands:



Relay Ladder

Operand:	Type:	Format:	Enter:	Notes:
Source	string	tag	string to copy characters from	String data types are: <ul style="list-style-type: none"> • default STRING data type • any new string data type that you create
Quantity	SINT INT DINT	immediate tag	number of characters to copy	The Start plus the Quantity must be less than or equal to the DATA size of the Source.
Start	SINT INT DINT	immediate tag	position of the first character to copy	Enter a number between 1 and the DATA size of the Source.
Destination	string	tag	string to copy the characters to	



Structured Text

```
MID(Source,Qty,Start,
     Dest);
```

The operands are the same as those for the relay ladder MID instruction.

Description: The MID instruction copies a group of characters from the Source and places the result in the Destination.

- The Start position and Quantity define the characters to copy.
- Unless the Source and Destination are the same tag, the Source remains unchanged.

Arithmetic Status Flags: not affected

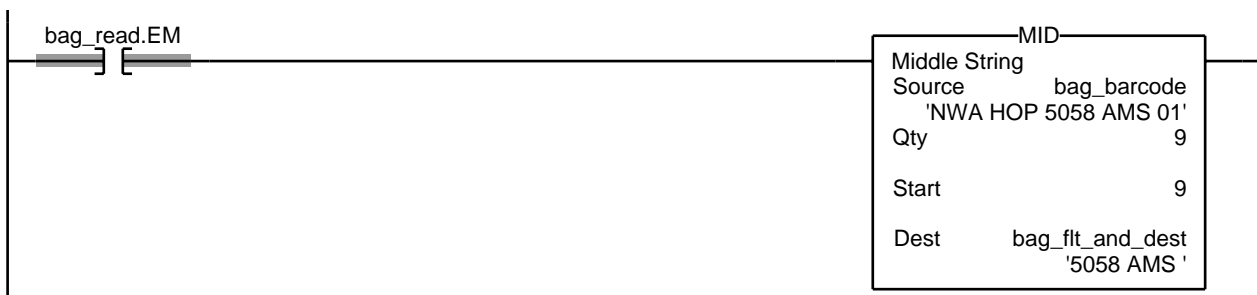
Fault Conditions:

Type:	Code:	Cause:	Recovery Method:
4	51	The LEN value of the string tag is greater than the DATA size of the string tag.	1. Check that no instruction is writing to the LEN member of the string tag. 2. In the LEN value, enter the number of characters that the string contains.
4	56	The Start or Quantity value is invalid.	1. Check that the Start value is between 1 and the DATA size of the Source. 2. Check that the Start value plus the Quantity value is less than or equal to the DATA size of the Source.

Execution:

Condition:	Relay Ladder Action:	Structured Text Action:
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	na
EnableIn is set	na	EnableIn is always set. The instruction executes.
instruction execution	The instruction copies the specified characters from a string and stores them in another string.	
postscan	The rung-condition-out is set to false.	No action taken.

Example: In a baggage handling conveyor of an airport, each bag gets a bar code. Characters 9 - 17 of the bar code are the flight number and destination airport of the bag. After the bar code is read (*bag_read.EM* is set) the MID instruction copies the flight number and destination airport to the *bag_flt_and_dest* string.

Relay Ladder**Structured Text**

```

IF bag_read.EM THEN
    MID(bar_barcode, 9, 9, bag_flt_and_dest);
    bag_read.EM := 0;
END_IF;
  
```

ASCII Conversion Instructions

(STOD, STOR, DTOS, RTOS, UPPER, LOWER)

Introduction

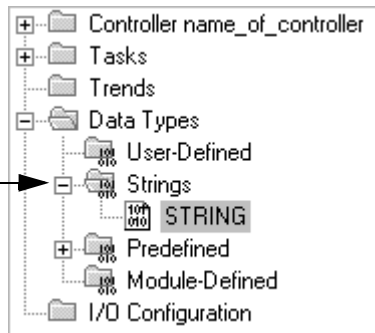
Use the ASCII conversion instructions to alter the format of data.

If you want to:	For example:	Use this instruction:	Available in these languages:	See page:
convert the ASCII representation of an integer value to a SINT, INT, DINT, or REAL value	convert a value from a weight scale or other ASCII device to an integer so you can use it in your logic	STOD	relay ladder structured text	18-4
convert the ASCII representation of a floating-point value to a REAL value	convert a value from a weight scale or other ASCII device to a REAL value so you can use it in your logic	STOR	relay ladder structured text	18-6
convert a SINT, INT, DINT, or REAL value to a string of ASCII characters	convert a variable to an ASCII string so you can send it to a MessageView terminal	DTOS	relay ladder structured text	18-8
convert a REAL value to a string of ASCII characters	convert a variable to an ASCII string so you can send it to a MessageView terminal	RTOS	relay ladder structured text	18-10
convert the letters in a string of ASCII characters to upper case	convert an entry made by an operator to all upper case so you can search for it in an array	UPPER	relay ladder structured text	18-12
convert the letters in a string of ASCII characters to lower case	convert an entry made by an operator to all lower case so you can search for it in an array	LOWER	relay ladder structured text	18-14

You can also use the following instructions to compare or manipulate ASCII characters:

If you want to:	Use this instruction:	See page:
add characters to the end of a string	CONCAT	17-3
delete characters from a string	DELETE	17-5
determine the starting character of a sub-string	FIND	17-7
insert characters into a string	INSERT	17-9
extract characters from a string	MID	17-11
rearrange the bytes of a INT, DINT, or REAL tag	SWPB	6-19
compare a string to another string	CMP	4-2
see if the characters are equal to specific characters	EQU	4-7
see if the characters are not equal to specific characters	NEQ	4-38
see if the characters are equal to or greater than specific characters	GEQ	4-11
see if the characters are greater than specific characters	GRT	4-15
see if the characters are equal to or less than specific characters	LEQ	4-19
see if the characters are less than specific characters	LES	4-23
find a string in an array of strings	FSC	7-19

String Data Types



You store ASCII characters in tags that use a string data type.

- You can use the default STRING data type. It stores up to 82 characters.
- You can create a new string data type that stores less or more characters.

To create a new string data type, see *Logix5000 Controllers Common Procedures*, publication 1756-PM001.

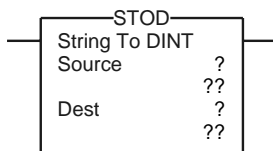
Each string data type contains the following members:

Name:	Data Type:	Description:	Notes:
LEN	DINT	number of characters in the string	<p>The LEN automatically updates to the new count of characters whenever you:</p> <ul style="list-style-type: none"> • use the String Browser dialog box to enter characters • use instructions that read, convert, or manipulate a string <p>The LEN shows the length of the current string. The DATA member may contain additional, old characters, which are not included in the LEN count.</p>
DATA	SINT array	ASCII characters of the string	<ul style="list-style-type: none"> • To access the characters of the string, address the name of the tag. For example, to access the characters of the <i>string_1</i> tag, enter <i>string_1</i>. • Each element of the DATA array contains one character. • You can create new string data types that store less or more characters.

String To DINT (STOD)

The STOD instruction converts the ASCII representation of an integer to an integer or REAL value.

Operands:



Relay Ladder

Operand:	Type:	Format:	Enter:	Notes:
Source	string	tag	tag that contains the value in ASCII	String data types are: <ul style="list-style-type: none"> • default STRING data type • any new string data type that you create
Destination	SINT INT DINT REAL	tag	tag to store the integer value	If the Source value is a floating-point number, the instruction converts only the non-fractional part of the number (regardless of the destination data type).



STOD (Source, Dest) ;

Structured Text

The operands are the same as those for the relay ladder STOD instruction.

Description: The STOD converts the Source to an integer and places the result in the Destination.

- The instruction converts positive and negative numbers.
- If the Source string contains non-numeric characters, the STOD converts the first set of contiguous numbers:
 - The instruction skips any initial control or non-numeric characters (except the minus sign in front of a number).
 - If the string contains multiple groups of numbers that are separated by delimiters (e.g., /), the instruction converts only the first group of numbers.

Arithmetic Status Flags: Arithmetic status flags are affected.

Fault Conditions:

Type:	Code:	Cause:	Recovery Method:
4	51	The LEN value of the string tag is greater than the DATA size of the string tag.	1. Check that no instruction is writing to the LEN member of the string tag. 2. In the LEN value, enter the number of characters that the string contains.
4	53	The output number is beyond the limits of the destination data type.	Either: <ul style="list-style-type: none"> • Reduce the size of the ASCII value. • Use a larger data type for the destination.

Execution:

Condition:	Relay Ladder Action:	Structured Text Action:
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	na
EnableIn is set	na	EnableIn is always set. The instruction executes.
instruction execution	S:C is set. Destination is cleared. The instruction converts the Source. If the result is zero, then S:Z is set	
postscan	The rung-condition-out is set to false.	No action taken.

Example: When *MV_read.EM* is set, the STOD instruction converts the first set of numeric characters in *MV_msg* to an integer value. The instruction skips the initial control character (\$06) and stops at the delimiter (\).

Relay Ladder**Structured Text**

```

IF MV_read.EM THEN

    STOD (MV_msg, MV_msg_nmbr) ;

    MV_read.EM := 0;

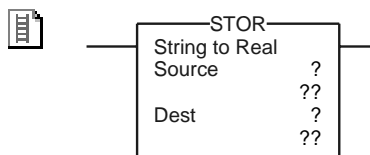
END_IF;

```

String To REAL (STOR)

The STOR instruction converts the ASCII representation of a floating-point value to a REAL value.

Operands:



Relay Ladder Operands

Operand:	Type:	Format:	Enter:	Notes:
Source	string	tag	tag that contains the value in ASCII	String data types are: <ul style="list-style-type: none"> • default STRING data type • any new string data type that you create
Destination	REAL	tag	tag to store the REAL value	

 STOR (Source, Dest) ;

Structured Text

The operands are the same as those for the relay ladder STOR instruction.

Description: The STOR converts the Source to a REAL value and places the result in the Destination.

- The instruction converts positive and negative numbers.
- If the Source string contains non-numeric characters, the STOR converts the first set of contiguous numbers, including the decimal point [.]:
 - The instruction skips any initial control or non-numeric characters (except the minus sign in front of a number).
 - If the string contains multiple groups of numbers that are separated by delimiters (e.g., /), the instruction converts only the first group of numbers.

Arithmetic Status Flags: Arithmetic status flags are affected.

Fault Conditions:

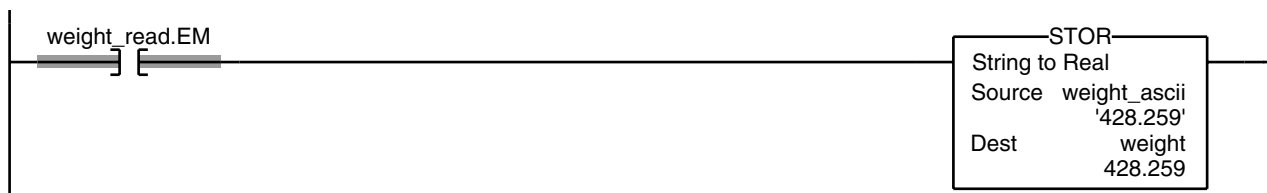
Type:	Code:	Cause:	Recovery Method:
4	51	The LEN value of the string tag is greater than the DATA size of the string tag.	1. Check that no instruction is writing to the LEN member of the string tag. 2. In the LEN value, enter the number of characters that the string contains.
4	53	The output number is beyond the limits of the destination data type.	Either: <ul style="list-style-type: none"> • Reduce the size of the ASCII value. • Use a larger data type for the destination.

Execution:

Condition:	Relay Ladder Action:	Structured Text Action:
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	na
EnableIn is ste	na	EnableIn is always set. The instruction executes.
instruction execution	S:C is set. Destination is cleared. The instruction converts the Source. If the result is zero, then S:Z is set	
postscan	The rung-condition-out is set to false.	No action taken.

Example: After reading the weight from a scale (*weight_read.EM* is set) the STOR instruction converts the numeric characters in *weight_ascii* to a REAL value.

You *may* see a slight difference between the fractional parts of the Source and Destination.

Relay Ladder**Structured Text**

```

IF weight_read.EM THEN

    STOR(weight_ascii,weight);

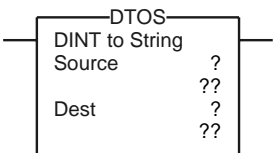
    weight_read.EM := 0;

END_IF;
  
```

DINT to String (DTOS)

The DTOS instruction produces the ASCII representation of a value.

Operands:



Relay Ladder

Operand:	Type:	Format:	Enter:	Notes:
Source	SINT INT DINT REAL	tag	tag that contains the value	If the Source is a REAL, the instruction converts it to a DINT value. Refer to REAL to an integer on page A-6.
Destination	string	tag	tag to store the ASCII value	String data types are: <ul style="list-style-type: none"> • default STRING data type • any new string data type that you create



DTOS (Source, Dest) ;

Structured Text

The operands are the same as those for the relay ladder DTOS instruction.

Description: The DTOS converts the Source to a string of ASCII characters and places the result in the Destination.

Arithmetic Status Flags: not affected

Fault Conditions:

Type:	Code:	Cause:	Recovery Method:
4	51	The LEN value of the string tag is greater than the DATA size of the string tag.	1. Check that no instruction is writing to the LEN member of the string tag. 2. In the LEN value, enter the number of characters that the string contains.
4	52	The output string is larger than the destination.	Create a new string data type that is large enough for the output string. Use the new string data type as the data type for the destination.

Execution:

Condition:	Relay Ladder Action:	Structured Text Action:
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	na
EnableIn is set	na	EnableIn is always set. The instruction executes.
instruction execution	The instruction converts the Source.	
postscan	The rung-condition-out is set to false.	No action taken.

Example: When *temp_high* is set, the DTOS instruction converts the value in *msg_num* to a string of ASCII characters and places the result in *msg_num_ascii*. Subsequent rungs insert or concatenate *msg_num_ascii* with other strings to produce a complete message for a display terminal.

Relay Ladder



Structured Text

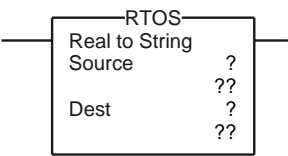
```

IF temp_high THEN
    DTOS(msg_num,msg_num_ascii);
    temp_high := 0;
END_IF;
  
```

REAL to String (RTOS)

The RTOS instruction produces the ASCII representation of a REAL value.

Operands:



Relay Ladder

Operand:	Type:	Format:	Enter:	Notes:
Source	REAL	tag	tag that contains the REAL value	
Destination	string	tag	tag to store the ASCII value	String data types are: <ul style="list-style-type: none">• default STRING data type• any new string data type that you create



RTOS (Source, Dest) ;

Structured Text

The operands are the same as those for the relay ladder RTOS instruction.

Description: The RTOS converts the Source to a string of ASCII characters and places the result in the Destination.

Arithmetic Status Flags: not affected

Fault Conditions:

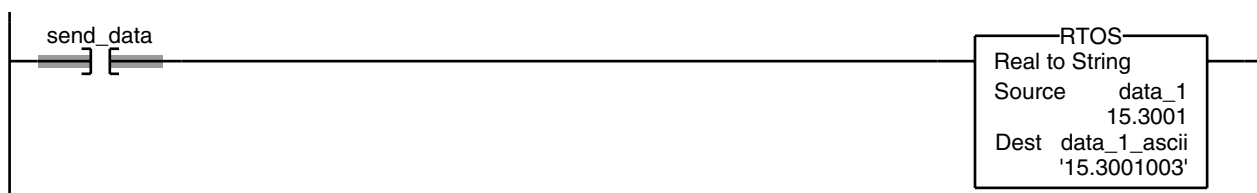
Type:	Code:	Cause:	Recovery Method:
4	51	The LEN value of the string tag is greater than the DATA size of the string tag.	1. Check that no instruction is writing to the LEN member of the string tag. 2. In the LEN value, enter the number of characters that the string contains.
4	52	The output string is larger than the destination.	Create a new string data type that is large enough for the output string. Use the new string data type as the data type for the destination.

Execution:

Condition:	Relay Ladder Action:	Structured Text Action:
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	na
EnableIn is set	na	EnableIn is always set. The instruction executes.
instruction execution	The instruction converts the Source.	
postscan	The rung-condition-out is set to false.	No action taken.

Example: When *send_data* is set, the RTOS instruction converts the value in *data_1* to a string of ASCII characters and places the result in *data_1_ascii*. Subsequent rungs insert or concatenate *data_1_ascii* with other strings to produce a complete message for a display terminal.

You *may* see a slight difference between the fractional parts of the Source and Destination.

Relay Ladder**Structured Text**

```

IF send_data THEN

    RTOS(data_1,data_1_ascii);

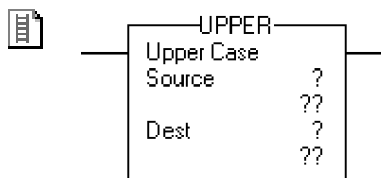
    send_data := 0;

END_IF;
  
```

Upper Case (UPPER)

The UPPER instruction converts the alphabetical characters in a string to upper case characters.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Source	string	tag	tag that contains the characters that you want to convert to upper case
Destination	string	tag	tag to store the characters in upper case



```
UPPER (Source, Dest) ;
```

Structured Text

The operands are the same as those for the relay ladder UPPER instruction.

Description: The UPPER instruction converts to upper case all the letters in the Source and places the result in the Destination.

- ASCII characters are case sensitive. Upper case “A” (\$41) is *not* equal to lower case “a” (\$61).
- If operators directly enter ASCII characters, convert the characters to all upper case or all lower case before you compare them.

Any characters in the Source string that are not letters remain unchanged.

Arithmetic Status Flags: not affected

Fault Conditions: none

Execution:

Condition:	Relay Ladder Action:	Structured Text Action:
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	na
EnableIn is set	na	EnableIn is always set. The instruction executes.
instruction execution	The instruction converts the Source to upper case.	
postscan	The rung-condition-out is set to false.	No action taken.

Example: To find information about a specific item, an operator enters the catalog number of the item into an ASCII terminal. After the controller reads the input from a terminal (*terminal_read.EM* is set), the UPPER instruction converts the characters in *catalog_number* to all upper case characters and stores the result in *catalog_number_upper_case*. A subsequent rung then searches an array for characters that match those in *catalog_number_upper_case*.

Relay Ladder



Structured Text

```

IF terminal_read.EM THEN

    UPPER(catalog_number,catalog_number_upper_case);

    terminal_read.EM := 0;

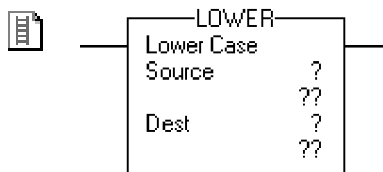
END_IF;

```

Lower Case (LOWER)

The LOWER instruction converts the alphabetical characters in a string to lower case characters.

Operands:



Relay Ladder

Operand:	Type:	Format:	Description:
Source	string	tag	tag that contains the characters that you want to convert to lower case
Destination	string	tag	tag to store the characters in lower case



```
LOWER (Source, Dest) ;
```

Structured Text

The operands are the same as those for the relay ladder LOWER instruction.

Description: The LOWER instruction converts to lower case all the letters in the Source and places the result in the Destination.

- ASCII characters are case sensitive. Upper case “A” (\$41) is *not* equal to lower case “a” (\$61).
- If operators directly enter ASCII characters, convert the characters to all upper case or all lower case before you compare them.

Any characters in the Source string that are not letters remain unchanged.

Arithmetic Status Flags: not affected

Fault Conditions: none

Execution:

Condition:	Relay Ladder Action:	Structured Text Action:
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	na
EnableIn is set	na	EnableIn is always set. The instruction executes.
instruction execution	The instruction converts the Source to lower case.	
postscan	The rung-condition-out is set to false.	No action taken.

Example: To find information about a specific item, an operator enters the item number into an ASCII terminal. After the controller reads the input from a terminal (*terminal_read.EM* is set), the LOWER instruction converts the characters in *item_number* to all lower case characters and stores the result in *item_number_lower_case*. A subsequent rung then searches an array for characters that match those in *item_number_lower_case*.

Relay Ladder



Structured Text

```

IF terminal_read.EM THEN

    LOWER(item_number,item_number_lower_case);

    terminal_read.EM := 0;

END_IF;
  
```

Notes:

Common Attributes

Introduction

This appendix describes attributes that are common to the Logix instructions.

For information about:	See page:
Immediate Values	A-1
Data Conversions	A-1

Immediate Values

Whenever you enter an immediate value (constant) in decimal format (e.g., -2, 3) the controller stores the value using 32 bits. If you enter a value in a radix other than decimal, such as binary or hexadecimal, and do not specify all 32 bits, the controller places a zero in the bits that you do not specify (zero-fill).

EXAMPLE

Zero-filling of immediate values

If you enter:	The controller stores:
-1	16#ffff ffff (-1)
16#ffff (-1)	16#0000 ffff (65535)
8#1234 (668)	16#0000 029c (668)
2#1010 (10)	16#0000 000a (10)

Data Conversions

Data conversions occur when you mix data types in your programming:

When programming in:	Conversions can occur when:
relay ladder logic	mix data types for the parameters within one instruction
function block	you wire two parameters that have different data types

Instructions execute faster and require less memory if all the operands of the instruction use:

- the same data type
- an optimal data type:
 - In the “Operands” section of each instruction in this manual, a **bold** data type indicates an optimal data type.
 - The DINT and REAL data types are typically the optimal data types.
 - Most function block instruction only support one data type (the optimal data type) for its operands.

If you mix data types and use tags that are not the optimal data type, the controller converts the data according to these rules

- Are *any* of the operands a REAL value?

If:	Then input operands (e.g., source, tag in an expression, limit) convert to:
Yes	REALs
No	DINTs

- After instruction execution, the result (a DINT or REAL value) converts to the destination data type, if necessary.

You cannot specify a BOOL tag in an instruction that operates on integer or REAL data types.

Because the conversion of data takes additional time and memory, you can increase the efficiency of your programs by:

- using the same data type throughout the instruction
- minimizing the use of the SINT or INT data types

In other words, use all DINT tags or all REAL tags, along with immediate values, in your instructions.

The following sections explain how the data is converted when you use SINT or INT tags or when you mix data types.

SINT or INT to DINT

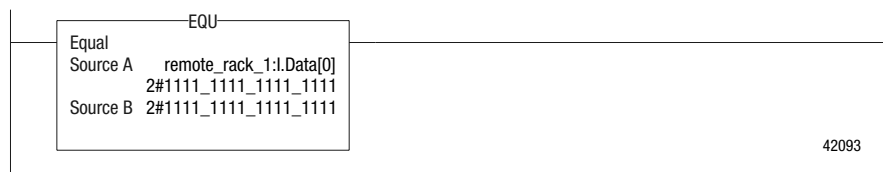
For those instructions that convert SINT or INT values to DINT values, the “Operands” sections in this manual identify the conversion method.

This conversion method:	Converts data by placing:
Sign-extension	the value of the left-most bit (the sign of the value) into each bit position to the left of the existing bits until there are 32 bits.
Zero-fill	zeroes to the left of the existing bits until there are 32 bits

The following example shows the results of converting a value using sign-extension and zero-fill.

This value	2#1111_1111_1111_1111	(-1)
Converts to this value by sign-extension	2#1111_1111_1111_1111_1111_1111_1111_1111	(-1)
Converts to this value by zero-fill	2#0000_0000_0000_0000_1111_1111_1111_1111	(65535)

Because immediate values are always zero-filled, the conversion of a SINT or INT value *may* produce unexpected results. In the following example, the comparison is false because Source A, an INT, converts by sign-extension; while Source B, an immediate value, is zero-filled.



If you use a SINT or INT tag and an immediate value in an instruction that converts data by sign-extension, use one of these methods to handle immediate values:

- Specify any immediate value in the decimal radix
- If you are entering the value in a radix other than decimal, specify all 32 bits of the immediate value. To do so, enter the value of the left-most bit into each bit position to its left until there are 32 bits.
- Create a tag for each operand and use the same data type throughout the instruction. To assign a constant value, either:
 - Enter it into one of the tags
 - Add a MOV instruction that moves the value into one of the tags.
- Use a MEQ instruction to check only the required bits

The following examples show two ways to mix an immediate value with an INT tag. Both examples check the bits of a 1771 I/O module to determine if all the bits are on. Since the input data word of a 1771 I/O module is an INT tag, it is easiest to use a 16-bit constant value.

EXAMPLE

Mixing an INT tag with an immediate value

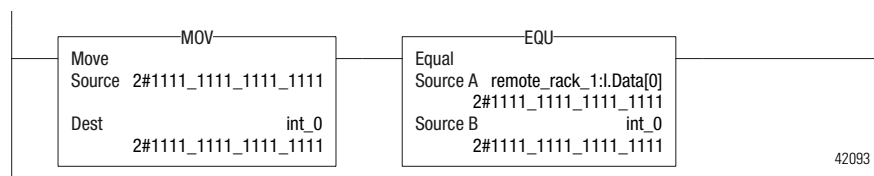
Since *remote_rack_1:I.Data[0]* is an INT tag, the value to check it against is also entered as an INT tag.



EXAMPLE

Mixing an INT tag with an immediate value

Since *remote_rack_1:I.Data[0]* is an INT tag, the value to check it against first moves into *int_0*, also an INT tag. The EQU instruction then compares both tags.



Integer to REAL

The controller stores REAL values in IEEE single-precision, floating-point number format. It uses one bit for the sign of the value, 23 bits for the base value, and eight bits for the exponent (32 bits total). If you mix an integer tag (SINT, INT, or DINT) and a REAL tag as inputs in the same instruction, the controller converts the integer value to a REAL value before the instruction executes.

- A SINT or INT value always converts to the same REAL value.
- A DINT value may not convert to the same REAL value:
 - A REAL value uses up to 24 bits for the base value (23 stored bits plus a “hidden” bit).
 - A DINT value uses up to 32 bits for the value (one for the sign and 31 for the value).
 - If the DINT value requires more than 24 significant bits, it *may not* convert to the same REAL value. If it will not, the controller rounds to the nearest REAL value using 24 significant bits.

DINT to SINT or INT

To convert a DINT value to a SINT or INT value, the controller truncates the upper portion of the DINT and sets the overflow status flag, if necessary. The following example shows the result of a DINT to SINT or INT conversion.

EXAMPLE

Conversion of a DINT to an INT and a SINT

This DINT value:	Converts to this smaller value:	
16#0001_0081 (65,665)	INT:	16#0081 (129)
	SINT:	16#81 (-127)

REAL to an integer

To convert a REAL value to an integer value, the controller rounds the fractional part and truncates the upper portion of the non-fractional part. If data is lost, the controller sets the overflow status flag. Numbers round as follows:

- Numbers other than $x.5$ round to the nearest whole number.
- $X.5$ rounds to the nearest even number.

The following example show the result of converting REAL values to DINT values.

EXAMPLE

Conversion of REAL values to DINT values

This REAL value:	Converts to this DINT value:
-2.5	-2
-1.6	-2
-1.5	-2
-1.4	-1
1.4	1
1.5	2
1.6	2
2.5	2

IMPORTANT

The arithmetic status flags are set based on the value being stored. Instructions that normally do not affect arithmetic status keywords might appear to do so if type conversion occurs because of mixed data types for the instruction parameters. The type conversion process sets the arithmetic status keywords.

Function Block Attributes

Introduction

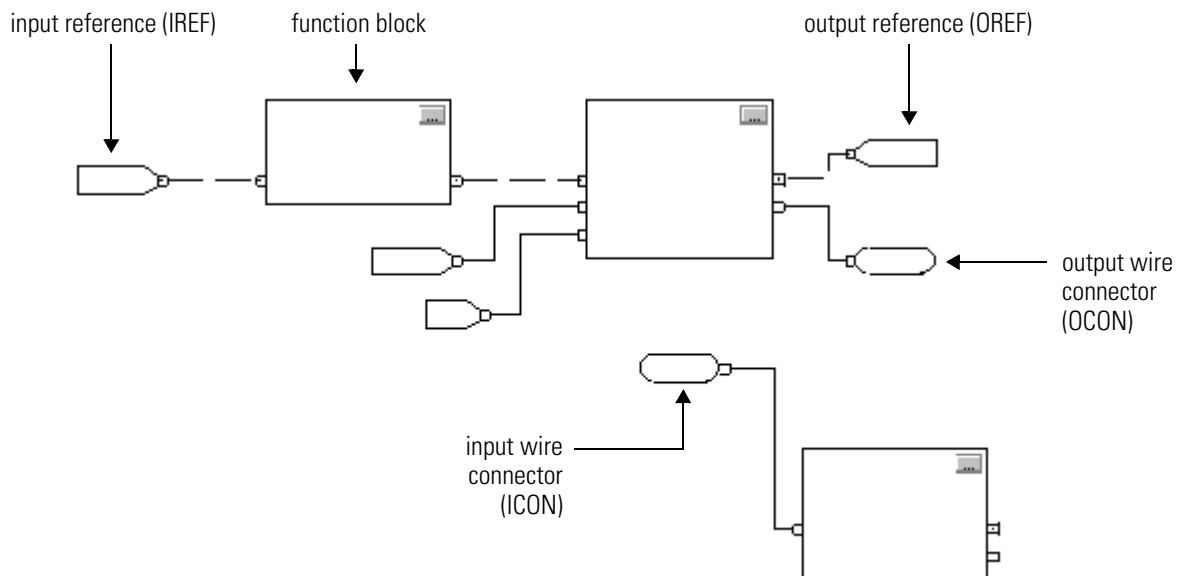
This appendix describes issues that are unique with function block instructions. Review the information in this appendix to make sure you understand how your function block routines will operate.

IMPORTANT

When programming in function block, restrict the range of engineering units to $\pm 10^{+/-15}$ because internal floating point calculations are done using single precision floating point. Engineering units outside of this range may result in a loss of accuracy if results approach the limitations of single precision floating point ($\pm 10^{+/-38}$).

Choose the Function Block Elements

To control a device, use the following elements:

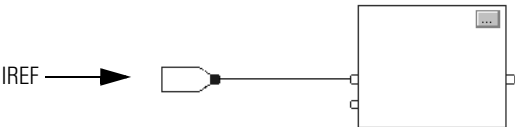


Use the following table to choose your function block elements:

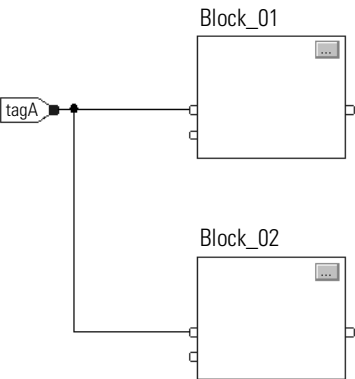
If you want to:	Then use a:
supply a value from an input device or tag	input reference (IREF)
send a value to an output device or tag	output reference (OREF)
perform an operation on an input value or values and produce an output value or values	function block
transfer data between function blocks when they are: <ul style="list-style-type: none">• far apart on the same sheet• on different sheets within the same routine	output wire connector (OCON) and an input wire connector (ICON)
disperse data to several points in the routine	single output wire connector (OCON) and multiple input wire connectors (ICON)

Latching Data

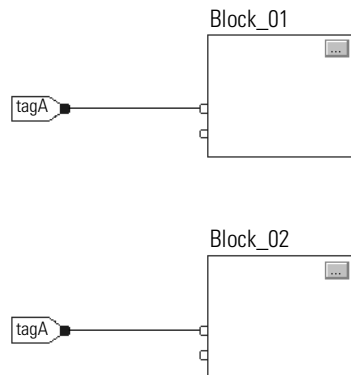
If you use an IREF to specify input data for a function block instruction, the data in that IREF is latched for the scan of the function block routine. The IREF latches data from program-scoped and controller-scoped tags. The controller updates all IREF data at the beginning of each scan.



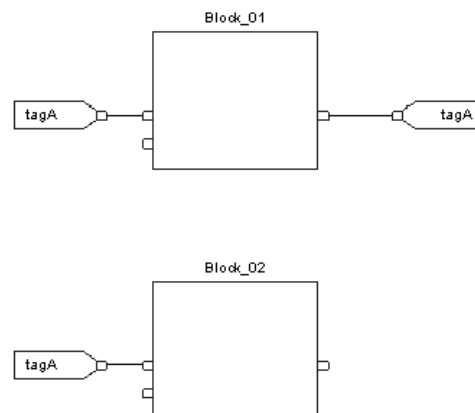
In this example, the value of tagA is stored at the beginning of the routine's execution. The stored value is used when Block_01 executes. The same stored value is also used when Block_02 executes. If the value of tagA changes during execution of the routine, the stored value of tagA in the IREF does not change until the next execution of the routine.



This example is the same as the one above. The value of tagA is stored only once at the beginning of the routine's execution. The routine uses this stored value throughout the routine.



Starting with RSLogix 5000 software, version 11, you can use the same tag in multiple IREFs and an OREF in the same routine. Because the values of tags in IREFs are latched every scan through the routine, all IREFs will use the same value, even if an OREF obtains a different tag value during execution of the routine. In this example, if tagA has a value of 25.4 when the routine starts executing this scan, and Block_01 changes the value of tagA to 50.9, the second IREF wired into Block_02 will still use a value of 25.4 when Block_02 executes this scan. The new tagA value of 50.9 will not be used by any IREFs in this routine until the start of the next scan.



Order of Execution

The RSLogix 5000 programming software automatically determines the order of execution for the function blocks in a routine when you:

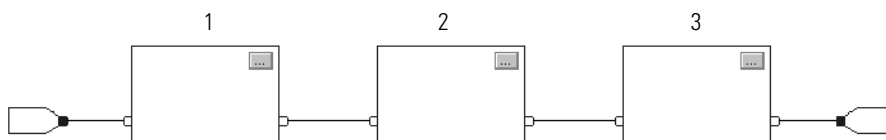
- verify a function block routine
- verify a project that contains a function block routine
- download a project that contains a function block routine

You define execution order by wiring function blocks together and indicating the data flow of any feedback wires, if necessary.

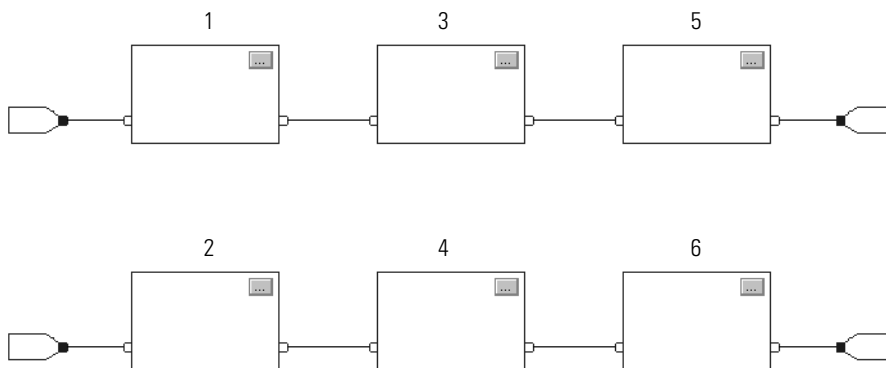
If function blocks are not wired together, it does not matter which block executes first. There is no data flow between the blocks.



If you wire the blocks sequentially, the execution order moves from input to output. The inputs of a block require data to be available before the controller can execute that block. For example, block 2 has to execute before block 3 because the outputs of block 2 feed the inputs of block 3.

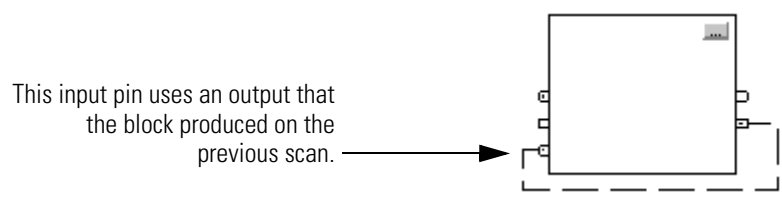


Execution order is only relative to the blocks that are wired together. The following example is fine because the two groups of blocks are not wired together. The blocks within a specific group execute in the appropriate order in relation to the blocks in that group.

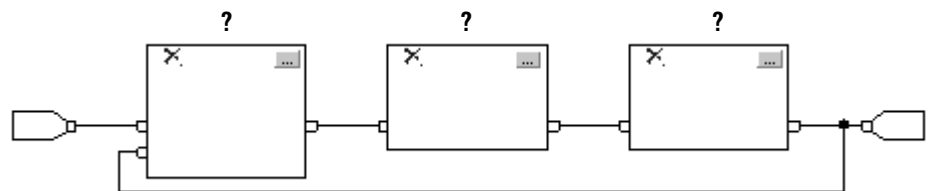


Resolve a Loop

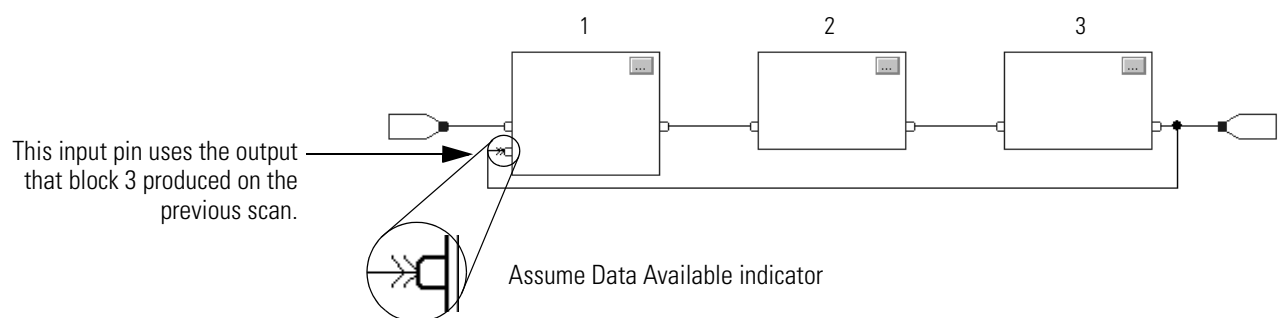
To create a feedback loop around a block, wire an output pin of the block to an input pin of the same block. The following example is OK. The loop contains only a single block, so execution order does not matter.



If a group of blocks are in a loop, the controller cannot determine which block to execute first. In other words, it cannot resolve the loop.

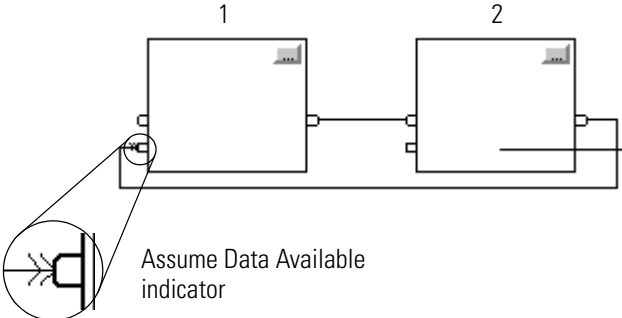
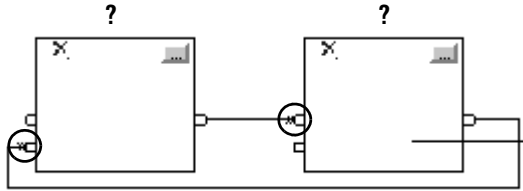


To identify which block to execute first, mark the input wire that creates the loop (the feedback wire) with the *Assume Data Available* indicator. In the following example, block 1 uses the output from block 3 that was produced in the previous execution of the routine.



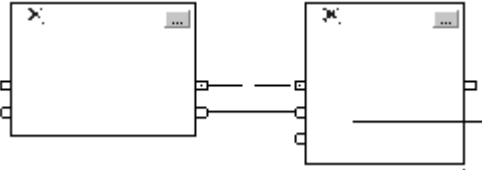
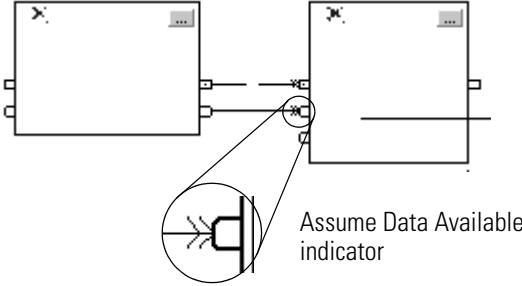
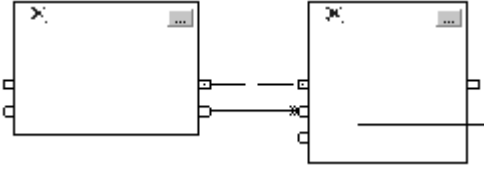
The *Assume Data Available* indicator defines the data flow within the loop. The arrow indicates that the data serves as input to the first block in the loop.

Do not mark all the wires of a loop with the *Assume Data Available* indicator.

This is OK	This is NOT OK
<div><p>Assume Data Available indicator</p><p>The <i>Assume Data Available</i> indicator defines the data flow within the loop.</p></div>	<div><p>The controller cannot resolve the loop because all the wires use the <i>Assume Data Available</i> indicator.</p></div>

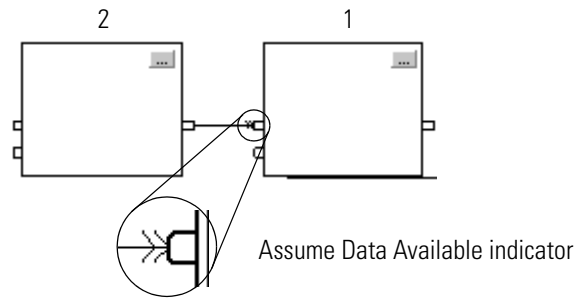
Resolve Data Flow Between Two Blocks

If you use two or more wires to connect two blocks, use the same data flow indicators for all of the wires between the two blocks.

This is OK	This is NOT OK
<div><p>Neither wire uses the <i>Assume Data Available</i> indicator.</p><p>Assume Data Available indicator</p><p>Both wires use the <i>Assume Data Available</i> indicator.</p></div>	<div><p>One wire uses the <i>Assume Data Available</i> indicator while the other wire does not.</p></div>

Create a One Scan Delay

To produce a one scan delay between blocks, use the *Assume Data Available* indicator. In the following example, block 1 executes first. It uses the output from block 2 that was produced in the previous scan of the routine.



Summary

In summary, a function block routine executes in this order:

1. The controller latches all data values in IREFs.
2. The controller executes the other function blocks in the order determined by how they are wired.
3. The controller writes outputs in OREFs.

Function Block Responses to Overflow Conditions

In general, the function block instructions that maintain history do not update history with $\pm\text{NAN}$, or $\pm\text{INF}$ values when an overflow occurs. Each instruction has one of these responses to an overflow condition:

Response 1:		Response 2:	Response 3:	
Blocks execute their algorithm and check the result for $\pm\text{NAN}$ or $\pm\text{INF}$. If $\pm\text{NAN}$ or $\pm\text{INF}$, the block outputs $\pm\text{NAN}$ or $\pm\text{INF}$.		Blocks with output limiting execute their algorithm and check the result for $\pm\text{NAN}$ or $\pm\text{INF}$. The output limits are defined by the HighLimit and LowLimit input parameters. If $\pm\text{INF}$, the block outputs a limited result. If $\pm\text{NAN}$, the output limits are not used and the block outputs $\pm\text{NAN}$.	The overflow condition does not apply. These instructions typically have a boolean output.	
ALM	NTCH	HLL	BAND	OSRI
DEDT	PMUL	INTG	BNOT	RESD
DERV	POSP	PI	BOR	RTOR
ESEL	RLIM	PIDE	BXOR	SETD
FGEN	RMPS	SCL	CUTD	TOFR
HPF	SCRV	SOC	D2SD	TONR
LDL2	SEL		D3SD	
LDLG	SNEG		DFF	
LPF	S RTP		JKFF	
MAVE	SSUM		OSFI	
MAXC	TOT			
MINC	UPDN			
MSTD				
MUX				

Timing Modes

These process control and drives instructions support different timing modes.

DEDT	LDLG	RLIM
DERV	LPF	SCRV
HPF	NTCH	SOC
INTG	PI	TOT
LDL2	PIDE	

There are three different timing modes:

Timing Mode:	Description:						
periodic	<p>Periodic mode is the default mode and is suitable for most control applications. We recommend that you place the instructions that use this mode in a routine that executes in a periodic task. The delta time (DeltaT) for the instruction is determined as follows:</p> <table> <tr> <th>If the instruction executes in a:</th><th>Then DeltaT equals:</th></tr> <tr> <td>periodic task</td><td> <p>period of the task</p> <p>Set the period of the task to whole milliseconds (ms). For DeltaT, the controller truncates any fractional portion of the task's period. For example, if the period of that task = 10.5 ms, the controller sets DeltaT = 10 ms.</p> <p>If you want to use a fractional value for the period of a task, use the oversample timing mode. With the oversample timing mode, set the OversampleDT parameter = period of the task, including any fractional value.</p> </td></tr> <tr> <td>event or continuous task</td><td> <p>elapsed time since the previous execution</p> <p>The controller truncates the elapsed time to whole milliseconds (ms). For example, if the elapsed time = 10.5 ms, the controller sets DeltaT = 10 ms.</p> </td></tr> </table> <p>The update of the process input needs to be synchronized with the execution of the task or sampled 5-10 times faster than the task executes in order to minimize the sampling error between the input and the instruction.</p>	If the instruction executes in a:	Then DeltaT equals:	periodic task	<p>period of the task</p> <p>Set the period of the task to whole milliseconds (ms). For DeltaT, the controller truncates any fractional portion of the task's period. For example, if the period of that task = 10.5 ms, the controller sets DeltaT = 10 ms.</p> <p>If you want to use a fractional value for the period of a task, use the oversample timing mode. With the oversample timing mode, set the OversampleDT parameter = period of the task, including any fractional value.</p>	event or continuous task	<p>elapsed time since the previous execution</p> <p>The controller truncates the elapsed time to whole milliseconds (ms). For example, if the elapsed time = 10.5 ms, the controller sets DeltaT = 10 ms.</p>
If the instruction executes in a:	Then DeltaT equals:						
periodic task	<p>period of the task</p> <p>Set the period of the task to whole milliseconds (ms). For DeltaT, the controller truncates any fractional portion of the task's period. For example, if the period of that task = 10.5 ms, the controller sets DeltaT = 10 ms.</p> <p>If you want to use a fractional value for the period of a task, use the oversample timing mode. With the oversample timing mode, set the OversampleDT parameter = period of the task, including any fractional value.</p>						
event or continuous task	<p>elapsed time since the previous execution</p> <p>The controller truncates the elapsed time to whole milliseconds (ms). For example, if the elapsed time = 10.5 ms, the controller sets DeltaT = 10 ms.</p>						
oversample	<p>In oversample mode, the delta time (DeltaT) used by the instruction is the value written into the OversampleDT parameter of the instruction. Use this mode when the instruction executes in an event or continuous task and the process input does not have a time stamp associated with its updates. If the process input has a time stamp value, use the real time sampling mode instead.</p> <p>Add logic to your program to control when the instruction executes. For example, you can use a timer set to the OversampleDeltaT value to control the execution by using the EnableIn input of the instruction.</p> <p>The process input needs to be sampled 5-10 times faster than the instruction is executed in order to minimize the sampling error between the input and the instruction.</p>						

Timing Mode:	Description:
real time sampling	<p>In the real time sampling mode, the delta time (DeltaT) used by the instruction is the difference between two time stamp values that correspond to the updates of the process input. Use this mode when the instruction executes in an event or continuous task and the process input has a time stamp associated with its updates.</p> <p>The time stamp value is read from the tag name entered for the RTSTimeStamp parameter of the instruction. Normally this tag name is a parameter on the input module associated with the process input.</p> <p>The instruction compares the configured RTSTime value (expected update period) against the calculated DeltaT to determine if every update of the process input is being read by the instruction. If DeltaT is not within 1 millisecond of the configuration time, the instruction sets the RTSMissed status bit to indicate that a problem exists reading updates for the input on the module.</p>

Time-based instructions require a constant value for DeltaT in order for the control algorithm to properly calculate the process output. If DeltaT varies, a discontinuity occurs in the process output. The severity of the discontinuity depends on the instruction and range over which DeltaT varies. A discontinuity occurs if the:

- instruction is not executed during a scan.
- instruction is executed multiple times during a task.
- task is running and the task scan rate or the sample time of the process input changes.
- user changes the time base mode while the task is running.
- Order parameter is changed on a filter block while the task is running. Changing the Order parameter selects a different control algorithm within the instruction.

Common instruction parameters for timing modes

The instructions that support time base modes have these input and output parameters:

Input parameters

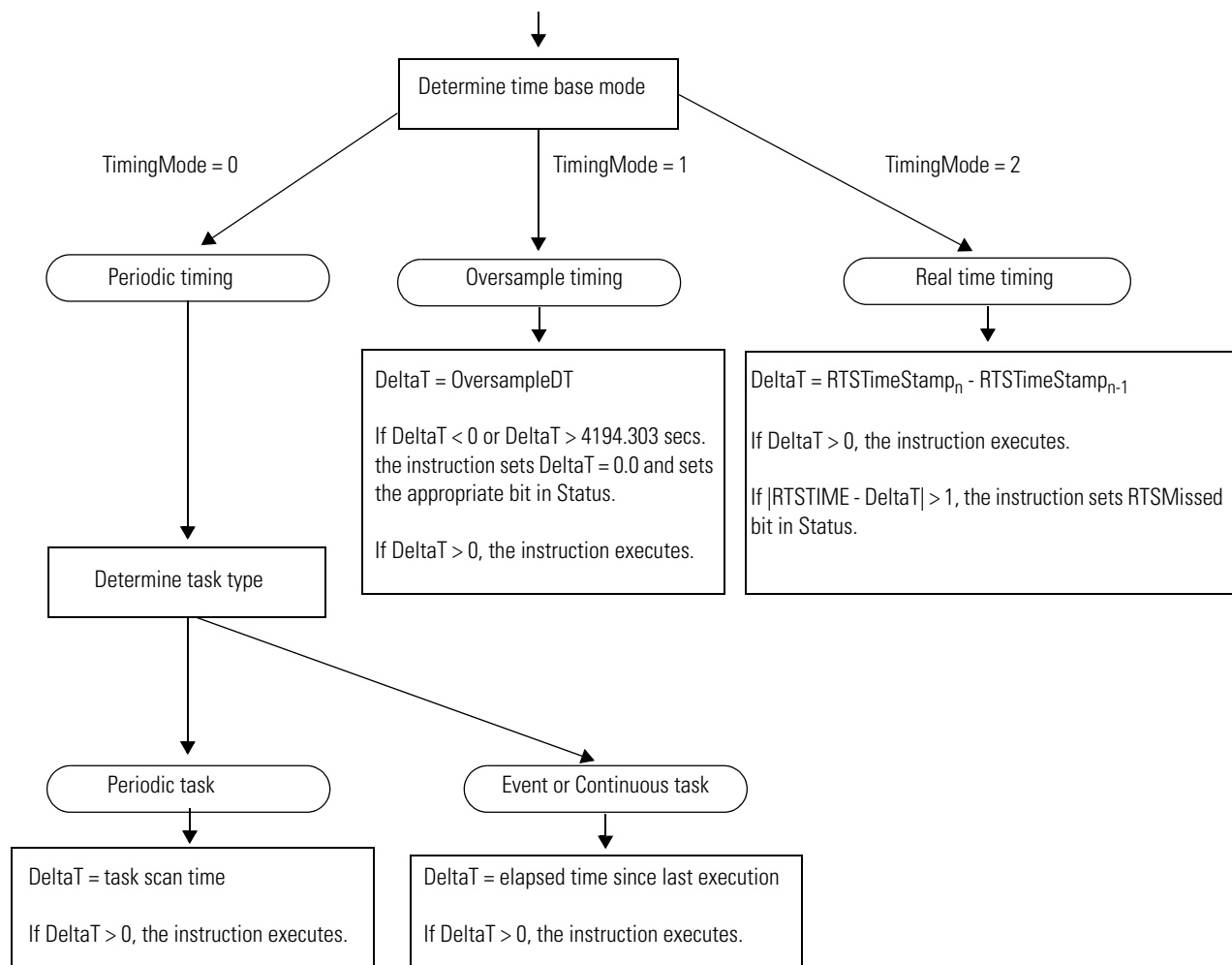
Input Parameter:	Data Type:	Description:								
TimingMode	DINT	<p>Selects timing execution mode.</p> <table><tr><th>Value:</th><th>Description:</th></tr><tr><td>0</td><td>periodic mode</td></tr><tr><td>1</td><td>oversample mode</td></tr><tr><td>2</td><td>real time sampling mode</td></tr></table> <p>valid = 0 to 2 default = 0</p> <p>When TimingMode = 0 and task is periodic, periodic timing is enabled and DeltaT is set to the task scan rate. When TimingMode = 0 and task is event or continuous, periodic timing is enabled and DeltaT is set equal to the elapsed time span since the last time the instruction was executed.</p> <p>When TimingMode = 1, oversample timing is enabled and DeltaT is set to the value of the OversampleDT parameter.</p> <p>When TimingMode = 2, real time sampling timing is enabled and DeltaT is the difference between the current and previous time stamp values read from the module associated with the input.</p> <p>If TimingMode invalid, the instruction sets the appropriate bit in Status.</p>	Value:	Description:	0	periodic mode	1	oversample mode	2	real time sampling mode
Value:	Description:									
0	periodic mode									
1	oversample mode									
2	real time sampling mode									
OversampleDT	REAL	<p>Execution time for oversample timing. The value used for DeltaT is in seconds. If TimingMode = 1, then OversampleDT = 0.0 disables the execution of the control algorithm. If invalid, the instruction sets DeltaT = 0.0 and sets the appropriate bit in Status.</p> <p>valid = 0 to 4194.303 seconds default = 0.0</p>								
RTSTime	DINT	<p>Module update period for real time sampling timing. The expected DeltaT update period is in milliseconds. The update period is normally the value that was used to configure the module's update time. If invalid, the instruction sets the appropriate bit in Status and disables RTSMissed checking.</p> <p>valid = 1 to 32,767ms default = 1</p>								
RTSTimeStamp	DINT	<p>Module time stamp value for real time sampling timing. The time stamp value that corresponds to the last update of the input signal. This value is used to calculate DeltaT. If invalid, the instruction sets the appropriate bit in Status, disables execution of the control algorithm, and disables RTSMissed checking.</p> <p>valid =1 to 32,767ms (wraps from 32767 to 0) 1 count = 1 millisecond default = 0</p>								

Output parameters

Output Parameter:	Data Type:	Description:
DeltaT	REAL	<p>Elapsed time between updates. This is the elapsed time in seconds used by the control algorithm to calculate the process output.</p> <p>Periodic: DeltaT = task scan rate if task is Periodic task, DeltaT = elapsed time since previous instruction execution if task is Event or Continuous task</p> <p>Oversample: DeltaT = OversampleDT</p> <p>Real Time Sampling: DeltaT = (RTTimeStamp_n - RTTimeStamp_{n-1})</p>
Status	DINT	Status of the function block.
TimingModeInv (Status.27)	BOOL	Invalid TimingMode value.
RTSMissed (Status.28)	BOOL	Only used in real time sampling mode. Set when $ABS \Delta T - RTTime > 1 \text{ (.001 second)}$.
RTTimeInv (Status.29)	BOOL	Invalid RTTime value.
RTTimeStampInv (Status.30)	BOOL	Invalid RTTimeStamp value.
DeltaTInv (Status.31)	BOOL	Invalid DeltaT value.

Overview of timing modes

The following diagram shows how an instruction determines the appropriate timing mode.



Program/Operator Control

Several instructions support the concept of Program/Operator control. These instructions include:

- Enhanced Select (ESEL)
- Totalizer (TOT)
- Enhanced PID (PIDE)
- Ramp/Soak (RMPS)
- Discrete 2-State Device (D2SD)
- Discrete 3-State Device (D3SD)

Program/Operator control lets you control these instructions simultaneously from both your user program and from an operator interface device. When in Program control, the instruction is controlled by the Program inputs to the instruction; when in Operator control, the instruction is controlled by the Operator inputs to the instruction.

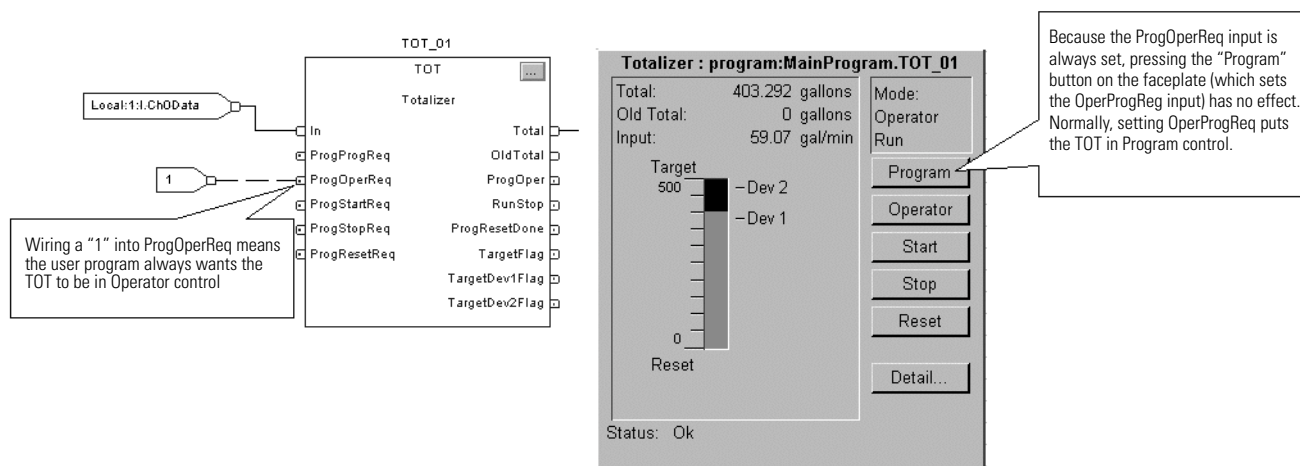
Program or Operator control is determined by using these inputs:

Input:	Description:
.ProgProgReq	A program request to go to Program control.
.ProgOperReq	A program request to go to Operator control.
.OperProgReq	An operator request to go to Program control.
.OperOperReq	An operator request to go to Operator control.

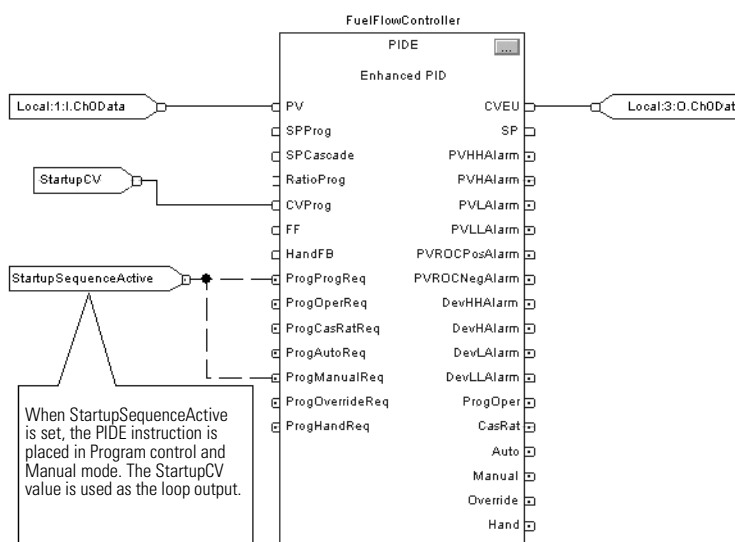
To determine whether an instruction is in Program or Control control, examine the ProgOper output. If ProgOper is set, the instruction is in Program control; if ProgOper is cleared, the instruction is in Operator control.

Operator control takes precedence over Program control if both input request bits are set. For example, if ProgProgReq and ProgOperReq are both set, the instruction goes to Operator control.

The Program request inputs take precedence over the Operator request inputs. This provides the capability to use the ProgProgReq and ProgOperReq inputs to “lock” an instruction in a desired control. For example, let’s assume that a Totalizer instruction will always be used in Operator control, and your user program will never control the running or stopping of the Totalizer. In this case, you could wire a literal value of 1 into the ProgOperReq. This would prevent the operator from ever putting the Totalizer into Program control by setting the OperProgReq from an operator interface device.



Likewise, constantly setting the ProgProgReq can “lock” the instruction into Program control. This is useful for automatic startup sequences when you want the program to control the action of the instruction without worrying about an operator inadvertently taking control of the instruction. In this example, you have the program set the ProgProgReq input during the startup, and then clear the ProgProgReq input once the startup was complete. Once the ProgProgReq input is cleared, the instruction remains in Program control until it receives a request to change. For example, the operator could set the OperOperReq input from a faceplate to take over control of that instruction. The following example shows how to lock an instruction into Program control.



Operator request inputs to an instruction are always cleared by the instruction when it executes. This allows operator interfaces to work with these instructions by merely setting the desired mode request bit. You don't have to program the operator interface to reset the request bits. For example, if an operator interface sets the OperAutoReq input to a PIDE instruction, when the PIDE instruction executes, it determines what the appropriate response should be and clears the OperAutoReq.

Program request inputs are not normally cleared by the instruction because these are normally wired as inputs into the instruction. If the instruction clears these inputs, the input would just get set again by the wired input. There might be situations where you want to use other logic to set the Program requests in such a manner that you want the Program requests to be cleared by the instruction. In this case, you can set the ProgValueReset input and the instruction will always clear the Program mode request inputs when it executes.

In this example, a rung of ladder logic in another routine is used to one-shot latch a ProgAutoReq to a PIDE instruction when a pushbutton is pushed. Because the PIDE instruction automatically clears the Program mode requests, you don't have to write any ladder logic to clear the ProgAutoReq after the routine executes, and the PIDE instruction will receive only one request to go to Auto every time the pushbutton is pressed.

When the TIC101AutoReq Pushbutton is pressed, one-shot latch ProgAutoReq for the PIDE instruction TIC101. TIC101 has been configured with the ProgValueReset input set, so when the PIDE instruction executes, it automatically clears ProgAutoReq.



Notes:

Structured Text Programming

Introduction

This appendix describes issues that are unique with structured text programming. Review the information in this appendix to make sure you understand how your structured text programming will execute.

For information about:	See page:
Structured Text Syntax	C-1
Assignments	C-2
Expressions	C-4
Instructions	C-11
Constructs	C-12
Comments	C-28

Structured Text Syntax

Structured text is a textual programming language that uses statements to define what to execute. Structured text is not case sensitive. Structured text can contain these components:

Term:	Definition:	Examples:
assignment (see page C-2)	Use an assignment statement to assign values to tags. The := operator is the assignment operator. Terminate the assignment with a semi colon ";".	<i>tag := expression;</i>
expression (see page C-4)	An expression is part of a complete assignment or construct statement. An expression evaluates to a number (numerical expression) or to a true or false state (BOOL expression). An expression contains:	
tags	A named area of the memory where data is stored (BOOL, SINT, INT, DINT, REAL, string).	<i>value1</i>
immediates	A constant value.	<i>4</i>
operators	A symbol or mnemonic that specifies an operation within an expression.	<i>tag1 + tag2</i> <i>tag1 >= value1</i>
functions	When executed, a function yields one value. Use parentheses to contain the operand of a function. Even though their syntax is similar, functions differ from instructions in that functions can only be used in expressions. Instructions cannot be used in expressions.	<i>function(tag1)</i>

Term:	Definition:	Examples:
instruction (see page C-11)	<p>An instruction is a standalone statement.</p> <p>An instruction uses parenthesis to contain its operands.</p> <p>Depending on the instruction, there can be zero, one, or multiple operands.</p> <p>When executed, an instruction yields one or more values that are part of a data structure.</p> <p>Terminate the instruction with a semi colon ";".</p> <p>Even though their syntax is similar, instructions differ from functions in that instructions cannot be used in expressions. Functions can only be used in expressions.</p>	<pre>instruction();</pre> <pre>instruction(operand);</pre> <pre>instruction(operand1, operand2, operand3);</pre>
construct (see page C-12)	<p>A conditional statement used to trigger structured text code (i.e, other statements).</p> <p>Terminate the construct with a semi colon ";".</p>	<pre>IF... THEN CASE FOR... DO WHILE... DO REPEAT... UNTIL EXIT</pre>
comment (see page C-28)	<p>Text that explains or clarifies what a section of structured text does.</p> <ul style="list-style-type: none"> • Use comments to make it easier to interpret the structured text. • Comments do not affect the execution of the structured text. • Comments can appear anywhere in structured text. 	<pre>//comment</pre> <pre>(*start of comment . . . end of comment*)</pre> <pre>/*start of comment . . . end of comment*/</pre>

Assignments

Use an assignment to change the value stored within a tag. An assignment has this syntax:

tag := *expression* ;

where:

Component:	Description:									
<i>tag</i>	represents the tag that is getting the new value the tag must be a BOOL, SINT, INT, DINT, or REAL									
:=	is the assignment symbol									
<i>expression</i>	represents the new value to assign to the tag									
<table> <tr> <th>If <i>tag</i> is this data type:</th><th>Use this type of expression:</th></tr> <tr> <td>BOOL</td><td>BOOL expression</td></tr> <tr> <td>SINT</td><td rowspan="4">numeric expression</td></tr> <tr> <td>INT</td></tr> <tr> <td>DINT</td></tr> <tr> <td>REAL</td></tr> </table>		If <i>tag</i> is this data type:	Use this type of expression:	BOOL	BOOL expression	SINT	numeric expression	INT	DINT	REAL
If <i>tag</i> is this data type:	Use this type of expression:									
BOOL	BOOL expression									
SINT	numeric expression									
INT										
DINT										
REAL										
;	ends the assignment									

The *tag* retains the assigned value until another assignment changes the value.

The expression can be simple, such as an immediate value or another tag name, or the expression can be complex and include several operators and/or functions. See the next section “Expressions” on page C-4 for details.

Specify a non-retentive assignment

The non-retentive assignment is different from the regular assignment described above in that the tag in a non-retentive assignment is reset to zero each time the controller:

- enters the RUN mode
- leaves the step of an SFC if you configure the SFC for *Automatic reset* (This applies only if you embed the assignment in the action of the step or use the action to call a structured text routine via a JSR instruction.)

A non-retentive assignment has this syntax:

tag [:=] *expression* ;

where:

Component:	Description:									
<i>tag</i>	represents the tag that is getting the new value the tag must be a BOOL, SINT, INT, DINT, or REAL									
[:=]	is the non-retentive assignment symbol									
<i>expression</i>	represents the new value to assign to the tag									
<table> <tr> <th>If <i>tag</i> is this data type:</th><th>Use this type of expression:</th></tr> <tr> <td>BOOL</td><td>BOOL expression</td></tr> <tr> <td>SINT</td><td rowspan="4">numeric expression</td></tr> <tr> <td>INT</td></tr> <tr> <td>DINT</td></tr> <tr> <td>REAL</td></tr> </table>		If <i>tag</i> is this data type:	Use this type of expression:	BOOL	BOOL expression	SINT	numeric expression	INT	DINT	REAL
If <i>tag</i> is this data type:	Use this type of expression:									
BOOL	BOOL expression									
SINT	numeric expression									
INT										
DINT										
REAL										
;	ends the assignment									

Assign an ASCII character to a string

Use the assignment operator to assign an ASCII character to an element of the DATA member of a string tag. To assign a character, specify the value of the character or specify the tag name, DATA member, and element of the character. For example:

This is OK:	This is <i>not</i> OK.
<code>string1.DATA[0] := 65;</code>	<code>string1.DATA[0] := A;</code>
<code>string1.DATA[0] := string2.DATA[0];</code>	<code>string1 := string2;</code>

To add or insert a string of characters to a string tag, use either of these ASCII string instructions:

To:	Use this instruction:
add characters to the end of a string	CONCAT
insert characters into a string	INSERT

Expressions

An expression is a tag name, equation, or comparison. To write an expression, use any of the following:

- tag name that stores the value (variable)
- number that you enter directly into the expression (immediate value)
- functions, such as: ABS, TRUNC
- operators, such as: +, -, <, >, And, Or

As you write expressions, follow these general rules:

- Use any combination of upper-case and lower-case letter. For example, these three variations of "AND" are acceptable: AND, And, and.
- For more complex requirements, use parentheses to group expressions within expressions. This makes the whole expression easier to read and ensures that the expression executes in the desired sequence. See “Determine the order of execution” on page C-10.

In structured text, you use two types of expressions:

BOOL expression: An expression that produces either the BOOL value of 1 (true) or 0 (false).

- A bool expression uses bool tags, relational operators, and logical operators to compare values or check if conditions are true or false. For example, `tag1 > 65`.
- A simple bool expression can be a single BOOL tag.
- Typically, you use bool expressions to condition the execution of other logic.

Numeric expression: An expression that calculates an integer or floating-point value.

- A numeric expression uses arithmetic operators, arithmetic functions, and bitwise operators. For example, `tag1 + 5`.
- Often, you nest a numeric expression within a bool expression. For example, `(tag1 + 5) > 65`.

Use the following table to choose operators for your expressions:

If you want to:	Then:
Calculate an arithmetic value	"Use arithmetic operators and functions" on page C-6.
Compare two values or strings	"Use relational operators" on page C-7.
Check if conditions are true or false	"Use logical operators" on page C-9.
Compare the bits within values	"Use bitwise operators" on page C-10.

Use arithmetic operators and functions

You can combine multiple operators and functions in arithmetic expressions.

Arithmetic operators calculate new values.

To:	Use this operator:	Optimal data type:
add	+	DINT, REAL
subtract/negate	-	DINT, REAL
multiply	*	DINT, REAL
exponent (x to the power of y)	**	DINT, REAL
divide	/	DINT, REAL
modulo-divide	MOD	DINT, REAL

Arithmetic functions perform math operations. Specify a constant, a non-boolean tag, or an expression for the function.

For:	Use this function:	Optimal data type:
absolute value	<i>ABS (numeric_expression)</i>	DINT, REAL
arc cosine	<i>ACOS (numeric_expression)</i>	REAL
arc sine	<i>ASIN (numeric_expression)</i>	REAL
arc tangent	<i>ATAN (numeric_expression)</i>	REAL
cosine	<i>COS (numeric_expression)</i>	REAL
radians to degrees	<i>DEG (numeric_expression)</i>	DINT, REAL
natural log	<i>LN (numeric_expression)</i>	REAL
log base 10	<i>LOG (numeric_expression)</i>	REAL
degrees to radians	<i>RAD (numeric_expression)</i>	DINT, REAL
sine	<i>SIN (numeric_expression)</i>	REAL
square root	<i>SQRT (numeric_expression)</i>	DINT, REAL
tangent	<i>TAN (numeric_expression)</i>	REAL
truncate	<i>TRUNC (numeric_expression)</i>	DINT, REAL

For example:

Use this format:	Example:	
	For this situation:	You'd write:
<i>value1 operator value2</i>	If <i>gain_4</i> and <i>gain_4_adj</i> are DINT tags and your specification says: "Add 15 to <i>gain_4</i> and store the result in <i>gain_4_adj</i> ."	<i>gain_4_adj := gain_4+15;</i>
<i>operator value1</i>	If <i>alarm</i> and <i>high_alarm</i> are DINT tags and your specification says: "Negate <i>high_alarm</i> and store the result in <i>alarm</i> ."	<i>alarm:= -high_alarm;</i>
<i>function(numeric_expression)</i>	If <i>overtravel</i> and <i>overtravel_POS</i> are DINT tags and your specification says: "Calculate the absolute value of <i>overtravel</i> and store the result in <i>overtravel_POS</i> ."	<i>overtravel_POS := ABS(overtravel);</i>
<i>value1 operator (function((value2+value3)/2))</i>	If <i>adjustment</i> and <i>position</i> are DINT tags and <i>sensor1</i> and <i>sensor2</i> are REAL tags and your specification says: "Find the absolute value of the average of <i>sensor1</i> and <i>sensor2</i> , add the <i>adjustment</i> , and store the result in <i>position</i> ."	<i>position := adjustment + ABS((sensor1 + sensor2)/2);</i>

Use relational operators

Relational operators compare two values or strings to provide a true or false result. The result of a relational operation is a BOOL value:

If the comparison is:	The result is:
true	1
false	0

Use the following relational operators:

For this comparison:	Use this operator:	Optimal Data Type:
equal	=	DINT, REAL, string
less than	<	DINT, REAL, string
less than or equal	<=	DINT, REAL, string
greater than	>	DINT, REAL, string
greater than or equal	>=	DINT, REAL, string
not equal	<>	DINT, REAL, string

For example:

Use this format:	Example:	
	For this situation:	You'd write:
<i>value1 operator value2</i>	If <i>temp</i> is a DINT tag and your specification says: "If <i>temp</i> is less than 100° then..."	IF temp<100 THEN...
<i>stringtag1 operator stringtag2</i>	If <i>bar_code</i> and <i>dest</i> are string tags and your specification says: "If <i>bar_code</i> equals <i>dest</i> then..."	IF bar_code=dest THEN...
<i>char1 operator char2</i> To enter an ASCII character directly into the expression, enter the decimal value of the character.	If <i>bar_code</i> is a string tag and your specification says: "If <i>bar_code</i> .DATA[0] equals 'A' then..."	IF bar_code.DATA[0]=65 THEN...
<i>bool_tag := bool_expressions</i>	If <i>count</i> and <i>length</i> are DINT tags, <i>done</i> is a BOOL tag, and your specification says "If <i>count</i> is greater than or equal to <i>length</i> , you are done counting."	done := (count >= length);

How Strings Are Evaluated

The hexadecimal values of the ASCII characters determine if one string is less than or greater than another string.

- When the two strings are sorted as in a telephone directory, the order of the strings determines which one is greater.

		ASCII Characters	Hex Codes	
l e s s e r ↑	g r e a t e r ↓	1ab	\$31\$61\$62	
		1b	\$31\$62	
		A	\$41	
		AB	\$41\$42	— AB < B
		B	\$42	—
		a	\$61	— a > B
		ab	\$61\$62	

- Strings are equal if their characters match.
- Characters are case sensitive. Upper case "A" (\$41) is *not* equal to lower case "a" (\$61).

For the decimal value and hex code of a character, see the back cover of this manual.

Use logical operators

Logical operators let you check if multiple conditions are true or false. The result of a logical operation is a BOOL value:

If the comparison is:	The result is:
true	1
false	0

Use the following logical operators:

For:	Use this operator:	Data Type:
logical AND	&, AND	BOOL
logical OR	OR	BOOL
logical exclusive OR	XOR	BOOL
logical complement	NOT	BOOL

For example:

Use this format:	Example:	
	For this situation:	You'd write:
<i>BOOLtag</i>	If <i>photoeye</i> is a BOOL tag and your specification says: "If <i>photoeye_1</i> is on then..."	IF photoeye THEN...
NOT <i>BOOLtag</i>	If <i>photoeye</i> is a BOOL tag and your specification says: "If <i>photoeye</i> is off then..."	IF NOT photoeye THEN...
<i>expression1</i> & <i>expression2</i>	If <i>photoeye</i> is a BOOL tag, <i>temp</i> is a DINT tag, and your specification says: "If <i>photoeye</i> is on and <i>temp</i> is less than 100° then..."	IF photoeye & (temp<100) THEN...
<i>expression1</i> OR <i>expression2</i>	If <i>photoeye</i> is a BOOL tag, <i>temp</i> is a DINT tag, and your specification says: "If <i>photoeye</i> is on or <i>temp</i> is less than 100° then..."	IF photoeye OR (temp<100) THEN...
<i>expression1</i> XOR <i>expression2</i>	If <i>photoeye1</i> and <i>photoeye2</i> are BOOL tags and your specification says: "If: <ul style="list-style-type: none"> • <i>photoeye1</i> is on while <i>photoeye2</i> is off or • <i>photoeye1</i> is off while <i>photoeye2</i> is on then..."	IF photoeye1 XOR photoeye2 THEN...
<i>BOOLtag</i> := <i>expression1</i> & <i>expression2</i>	If <i>photoeye1</i> and <i>photoeye2</i> are BOOL tags, <i>open</i> is a BOOL tag, and your specification says: "If <i>photoeye1</i> and <i>photoeye2</i> are both on, set <i>open</i> to true".	open := photoeye1 & photoeye2;

Use bitwise operators

Bitwise operators manipulate the bits within a value based on two values.

For:	Use this operator:	Optimal Data Type:
bitwise AND	&, AND	DINT
bitwise OR	OR	DINT
bitwise exclusive OR	XOR	DINT
bitwise complement	NOT	DINT

For example:

Use this format:	Example:	
	For this situation:	You'd write:
<i>value1 operator value2</i>	If <i>input1</i> , <i>input2</i> , and <i>result1</i> are DINT tags and your specification says: "Calculate the bitwise result of <i>input1</i> and <i>input2</i> . Store the result in <i>result1</i> ."	<i>result1 := input1 AND input2;</i>

Determine the order of execution

The operations you write into an expression are performed in a prescribed order, not necessarily from left to right.

- Operations of equal order are performed from left to right.
- If an expression contains multiple operators or functions, group the conditions in parenthesis "()" . This ensures the correct order of execution and makes it easier to read the expression.

Order:	Operation:
1.	()
2.	function (...)
3.	**
4.	– (negate)
5.	NOT
6.	*, /, MOD
7.	+, - (subtract)
8.	<, <=, >, >=
9.	=, <>
10.	&, AND
11.	XOR
12.	OR

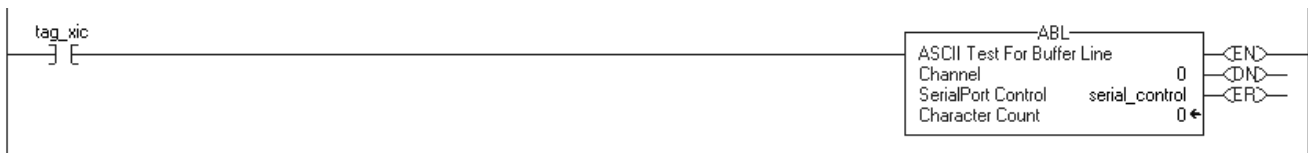
Instructions

Structured text statements can also be instructions. See the Locator Table at the beginning of this manual for a list of the instructions available in structured text. A structured text instruction executes each time it is scanned. A structured text instruction within a construct executes every time the conditions of the construct are true. If the conditions of the construct are false, the statements within the construct are not scanned. There is no rung-condition or state transition that triggers execution.

This differs from function block instructions that use EnableIn to trigger execution. Structured text instructions execute as if EnableIn is always set.

This also differs from relay ladder instructions that use rung-condition-in to trigger execution. Some relay ladder instructions only execute when rung-condition-in toggles from false to true. These are transitional relay ladder instructions. In structured text, instructions will execute each time they are scanned unless you pre-condition the execution of the structured text instruction.

For example, the ABL instruction is a transitional instruction in relay ladder. In this example, the ABL instruction only executes on a scan when *tag_xic* transitions from cleared to set. The ABL instruction does not execute when *tag_xic* stays set or when *tag_xic* is cleared.



In structured text, if you write this example as:

```
IF tag_xic THEN ABL(0,serial_control);

END_IF;
```

the ABL instruction will execute every scan that *tag_xic* is set, not just when *tag_xic* transitions from cleared to set.

If you want the ABL instruction to execute only when *tag_xic* transitions from cleared to set, you have to condition the structured text instruction. Use a one shot to trigger execution.

```
osri_1.InputBit := tag_xic;  
OSRI(osri_1);  
  
IF (osri_1.OutputBit) THEN  
    ABL(0,serial_control);  
END_IF;
```

Constructs

Constructs can be programmed singly or nested within other constructs.

If you want to:	Use this construct:	Available in these languages:	See page:
do something if or when specific conditions occur	IF...THEN	structured text	C-13
select what to do based on a numerical value	CASE...OF	structured text	C-16
do something a specific number of times before doing anything else	FOR...DO	structured text	C-19
keep doing something as long as certain conditions are true	WHILE...DO	structured text	C-22
keep doing something until a condition is true	REPEAT...UNTIL	structured text	C-25

IF...THEN

Use IF...THEN to do something if or when specific conditions occur.

Operands:



```
IF bool_expression THEN
    <statement>;
END_IF;
```

Structured Text

Operand:	Type:	Format:	Enter:
bool_ expression	BOOL	tag expression	BOOL tag or expression that evaluates to a BOOL value (BOOL expression)

Description: The syntax is:



To use ELSIF or ELSE, follow these guidelines:

1. To select from several possible groups of statements, add one or more ELSIF statements.
- Each ELSIF represents an alternative path.

• Specify as many ELSIF paths as you need.

• The controller executes the first true IF or ELSIF and skips the rest of the ELSIFs and the ELSE.
2. To do something when all of the IF or ELSIF conditions are false, add an ELSE statement.

The following table summarizes different combinations of IF, THEN, ELSIF, and ELSE.

If you want to:	And:	Then use this construct
do something if or when conditions are true	do nothing if conditions are false	IF...THEN
	do something else if conditions are false	IF...THEN...ELSE
choose from alternative statements (or groups of statements) based on input conditions	do nothing if conditions are false	IF...THEN...ELSIF
	assign default statements if all conditions are false	IF...THEN...ELSIF...ELSE

Example 1: IF...THEN

If you want this:	Enter this structured text:
IF rejects > 3 then conveyor = off (0) alarm = on (1)	IF rejects > 3 THEN conveyor := 0; alarm := 1; END_IF;

Example 2: IF...THEN...ELSE

If you want this:	Enter this structured text:
If conveyor direction contact = forward (1) then light = off Otherwise light = on	IF conveyor_direction THEN light := 0; ELSE light [:=] 1; END_IF;

The [:=] tells the controller to clear *light* whenever the controller:

- enters the RUN mode
- leaves the step of an SFC if you configure the SFC for *Automatic reset* (This applies only if you embed the assignment in the action of the step or use the action to call a structured text routine via a JSR instruction.)

Example 3: IF...THEN...ELSIF

If you want this:	Enter this structured text:
If sugar low limit switch = low (on) and sugar high limit switch = not high (on) then inlet valve = open (on) Until sugar high limit switch = high (off)	IF Sugar.Low & Sugar.High THEN Sugar.Inlet [:=] 1; ELSIF NOT(Sugar.High) THEN Sugar.Inlet := 0; END_IF;

The [:=] tells the controller to clear *Sugar.Inlet* whenever the controller:

- enters the RUN mode
- leaves the step of an SFC if you configure the SFC for *Automatic reset* (This applies only if you embed the assignment in the action of the step or use the action to call a structured text routine via a JSR instruction.)

Example 4: IF...THEN...ELSIF...ELSE

If you want this:	Enter this structured text:
If tank temperature > 100 then pump = slow If tank temperature > 200 then pump = fast otherwise pump = off	IF tank.temp > 200 THEN pump.fast :=1; pump.slow :=0; pump.off :=0; ELSIF tank.temp > 100 THEN pump.fast :=0; pump.slow :=1; pump.off :=0; ELSE pump.fast :=0; pump.slow :=0; pump.off :=1; END_IF;

CASE...OF

Use CASE to select what to do based on a numerical value.

Operands:



```
CASE numeric_expression OF
    selector1: statement;
    selectorN: statement;
ELSE
    statement;
END_CASE;
```

Structured Text

Operand:	Type:	Format:	Enter:
<i>numeric_expression</i>	SINT INT DINT REAL	tag expression	tag or expression that evaluates to a number (numeric expression)
<i>selector</i>	SINT INT DINT REAL	immediate	same type as <i>numeric_expression</i>

IMPORTANT If you use REAL values, use a range of values for a selector because a REAL value is more likely to be within a range of values than an exact match of one, specific value.

Description: The syntax is:



The syntax for entering the selector values is:

When selector is:	Enter:
one value	<i>value: statement</i>
multiple, distinct values	<i>value1, value2, valueN: <statement></i> Use a comma (,) to separate each value.
a range of values	<i>value1..valueN: <statement></i> Use two periods (..) to identify the range.
distinct values plus a range of values	<i>valuea, valueb, value1..valueN: <statement></i>

The CASE construct is similar to a switch statement in the C or C++ programming languages. However, with the CASE construct the controller executes *only* the statements that are associated with the *first matching* selector value. Execution *always breaks after the statements of that selector* and goes to the END_CASE statement.

Example

If you want this:	Enter this structured text:
If recipe number = 1 then	CASE recipe_number OF
Ingredient A outlet 1 = open (1)	1: Ingredient_A.Outlet_1 :=1;
Ingredient B outlet 4 = open (1)	Ingredient_B.Outlet_4 :=1;
If recipe number = 2 or 3 then	2,3: Ingredient_A.Outlet_4 :=1;
Ingredient A outlet 4 = open (1)	Ingredient_B.Outlet_2 :=1;
Ingredient B outlet 2 = open (1)	
If recipe number = 4, 5, 6, or 7 then	4..7: Ingredient_A.Outlet_4 :=1;
Ingredient A outlet 4 = open (1)	Ingredient_B.Outlet_2 :=1;
Ingredient B outlet 2 = open (1)	
■ If recipe number = 8, 11, 12, or 13 then	8,11..13 Ingredient_A.Outlet_1 :=1;
Ingredient A outlet 1 = open (1)	Ingredient_B.Outlet_4 :=1;
Ingredient B outlet 4 = open (1)	
Otherwise all outlets = closed (0)	ELSE
	Ingredient_A.Outlet_1 [:]=0;
	Ingredient_A.Outlet_4 [:]=0;
	Ingredient_B.Outlet_2 [:]=0;
	Ingredient_B.Outlet_4 [:]=0;
	END_CASE;

The [:=] tells the controller to also clear the outlet tags whenever the controller:

- enters the RUN mode
- leaves the step of an SFC if you configure the SFC for *Automatic reset* (This applies only if you embed the assignment in the action of the step or use the action to call a structured text routine via a JSR instruction.)

FOR...DO

Use the FOR...DO loop to do something a specific number of times before doing anything else.

Operands:



```
FOR count:= initial_value TO
final_value BY increment DO
    <statement>;
END_FOR;
```

Structured Text

Operand:	Type:	Format:	Description:
<i>count</i>	SINT INT DINT	tag	tag to store count position as the FOR...DO executes
<i>initial_value</i>	SINT INT DINT	tag expression immediate	must evaluate to a number specifies initial value for count
<i>final_value</i>	SINT INT DINT	tag expression immediate	specifies final value for count, which determines when to exit the loop
<i>increment</i>	SINT INT DINT	tag expression immediate	(<i>optional</i>) amount to increment count each time through the loop If you don't specify an increment, the count increments by 1.

IMPORTANT

Make sure that you *do not* iterate within the loop too many times in a single scan.

- The controller *does not* execute any other statements in the routine until it completes the loop.
- If the time that it takes to complete the loop is greater than the watchdog timer for the task, a major fault occurs.
- Consider using a different construct, such as IF...THEN.

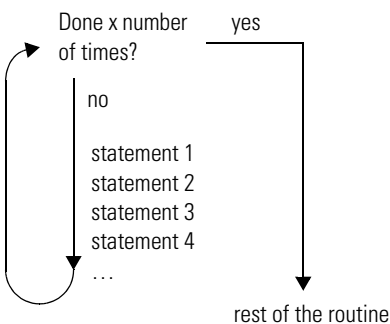
Description: The syntax is:

```
FOR count := initial_value
    TO final_value
optional { BY increment
DO
    <statement>;
optional { IF bool_expression THEN
            EXIT;
            END_IF;
END_FOR;
```

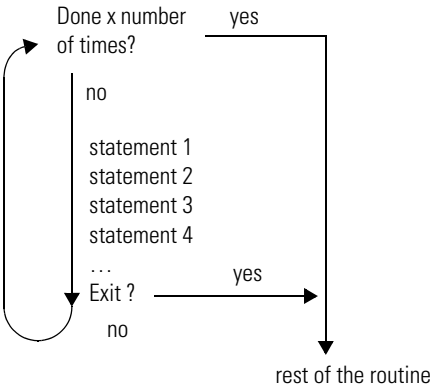
If you don't specify an increment, the loop increments by 1.

If there are conditions when you want to exit the loop early, use other statements, such as an IF...THEN construct, to condition an EXIT statement.

The following diagrams show how a FOR...DO loop executes and how an EXIT statement leaves the loop early.



The FOR...DO loop executes a specific number of times.



To stop the loop before the count reaches the last value, use an EXIT statement.

Example 1:

If you want this:	Enter this structured text:
Clear bits 0 - 31 in an array of BOOLs: 1. Initialize the <i>subscript</i> tag to 0. 2. Clear <i>array[subscript]</i> . For example, when <i>subscript</i> = 5, clear <i>array[5]</i> . 3. Add 1 to <i>subscript</i> . 4. If <i>subscript</i> is ≤ to 31, repeat 2 and 3. Otherwise, stop.	For subscript:=0 to 31 by 1 do array[subscript] := 0; End_for;

Example 2:

If you want this:	Enter this structured text:
<p>A user-defined data type (structure) stores the following information about an item in your inventory:</p> <ul style="list-style-type: none"> • Barcode ID of the item (string data type) • Quantity in stock of the item (DINT data type) <p>An array of the above structure contains an element for each different item in your inventory. You want to search the array for a specific product (use its bar code) and determine the quantity that is in stock.</p> <ol style="list-style-type: none"> 1. Get the size (number of items) of the <i>Inventory</i> array and store the result in <i>Inventory_Items</i> (DINT tag). 2. Initialize the <i>position</i> tag to 0. 3. If <i>Barcode</i> matches the ID of an item in the array, then: <ol style="list-style-type: none"> a. Set the <i>Quantity</i> tag = <i>Inventory[position].Qty</i>. This produces the quantity in stock of the item. b. Stop. <i>Barcode</i> is a string tag that stores the bar code of the item for which you are searching. For example, when <i>position</i> = 5, compare <i>Barcode</i> to <i>Inventory[5].ID</i>. 4. Add 1 to <i>position</i>. 5. If <i>position</i> is \leq to (<i>Inventory_Items</i> - 1), repeat 3 and 4. Since element numbers start at 0, the last element is 1 less than the number of elements in the array. Otherwise, stop. 	<pre> SIZE (Inventory, 0, Inventory_Items) ; For position:=0 to Inventory_Items - 1 do If Barcode = Inventory[position].ID then Quantity := Inventory[position].Qty; Exit; End_if; End_for; </pre>

WHILE...DO

Use the WHILE...DO loop to keep doing something as long as certain conditions are true.

Operands:



```
WHILE bool_expression DO
    <statement>;
END_WHILE;
```

Structured Text

Operand:	Type:	Format:	Enter:
bool_ expression	BOOL	tag expression	BOOL tag or expression that evaluates to a BOOL value

IMPORTANT

Make sure that you *do not* iterate within the loop too many times in a single scan.

- The controller *does not* execute any other statements in the routine until it completes the loop.
- If the time that it takes to complete the loop is greater than the watchdog timer for the task, a major fault occurs.
- Consider using a different construct, such as IF...THEN.

Description: The syntax is:

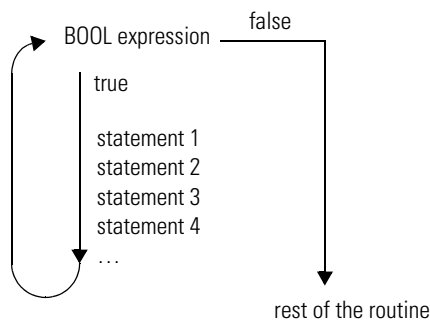
```

      WHILE bool_expression1 DO
          <statement>;
      optional {
          IF bool_expression2 THEN
              EXIT;
          END_IF;
      }
      END_WHILE;
```

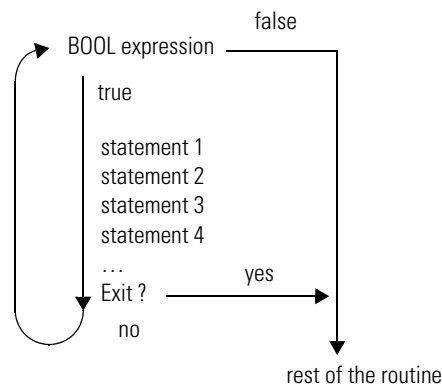
← statements to execute while *bool_expression1* is true

← If there are conditions when you want to exit the loop early, use other statements, such as an IF...THEN construct, to condition an EXIT statement.

The following diagrams show how a WHILE...DO loop executes and how an EXIT statement leaves the loop early.



While the *bool_expression* is true, the controller executes only the statements within the WHILE...DO loop.



To stop the loop before the conditions are true, use an EXIT statement.

Example 1:

If you want this:	Enter this structured text:
The WHILE...DO loop evaluates its conditions first. If the conditions are true, the controller then executes the statements within the loop.	<pre>pos := 0;</pre>
This differs from the REPEAT...UNTIL loop because the REPEAT...UNTIL loop executes the statements in the construct and then determines if the conditions are true before executing the statements again. The statements in a REPEAT...UNTIL loop are always executed at least once. The statements in a WHILE...DO loop might never be executed.	<pre>While ((pos <= 100) & structarray[pos].value <> targetvalue)) do pos := pos + 2; String_tag.DATA[pos] := SINT_array[pos]; end_while;</pre>

Example 2:

If you want this:	Enter this structured text:
<p>Move ASCII characters from a SINT array into a string tag. (In a SINT array, each element holds one character.) Stop when you reach the carriage return.</p> <ol style="list-style-type: none">1. Initialize <i>Element_number</i> to 0.2. Count the number of elements in <i>SINT_array</i> (array that contains the ASCII characters) and store the result in <i>SINT_array_size</i> (DINT tag).3. If the character at <i>SINT_array[element_number]</i> = 13 (decimal value of the carriage return), then stop.4. Set <i>String_tag[element_number]</i> = the character at <i>SINT_array[element_number]</i>.5. Add 1 to <i>element_number</i>. This lets the controller check the next character in <i>SINT_array</i>.6. Set the Length member of <i>String_tag</i> = <i>element_number</i>. (This records the number of characters in <i>String_tag</i> so far.)7. If <i>element_number</i> = <i>SINT_array_size</i>, then stop. (You are at the end of the array and it does not contain a carriage return.)8. Go to 3.	<pre>element_number := 0; SIZE(SINT_array, 0, SINT_array_size); While SINT_array[element_number] <> 13 do String_tag.DATA[element_number] := SINT_array[element_number]; element_number := element_number + 1; String_tag.LEN := element_number; If element_number = SINT_array_size then exit; end_if; end_while;</pre>

REPEAT...UNTIL

Use the REPEAT...UNTIL loop to keep doing something until conditions are true.

Operands:



```
REPEAT
    <statement>;
UNTIL bool_expression
END_REPEAT;
```

Structured Text

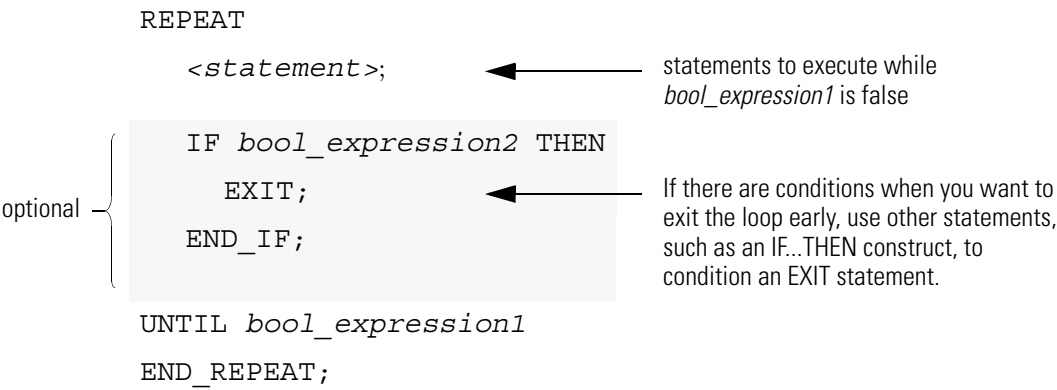
Operand:	Type:	Format:	Enter:
bool_expression	BOOL	tag expression	BOOL tag or expression that evaluates to a BOOL value (BOOL expression)

IMPORTANT

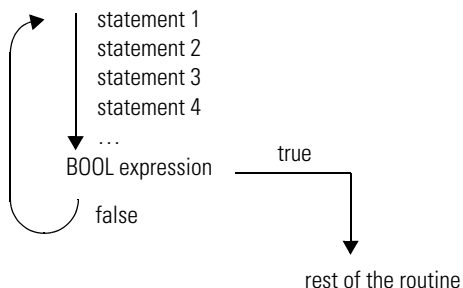
Make sure that you *do not* iterate within the loop too many times in a single scan.

- The controller *does not* execute any other statements in the routine until it completes the loop.
- If the time that it takes to complete the loop is greater than the watchdog timer for the task, a major fault occurs.
- Consider using a different construct, such as IF...THEN.

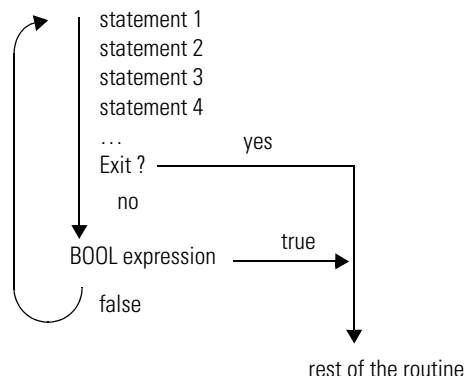
Description: The syntax is:



The following diagrams show how a REPEAT...UNTIL loop executes and how an EXIT statement leaves the loop early.



While the *bool_expression* is false, the controller executes only the statements within the REPEAT...UNTIL loop.



To stop the loop before the conditions are false, use an EXIT statement.

Example 1:

If you want this:	Enter this structured text:
<p>The REPEAT...UNTIL loop executes the statements in the construct and then determines if the conditions are true before executing the statements again.</p> <p>This differs from the WHILE...DO loop because the WHILE...DO The WHILE...DO loop evaluates its conditions first. If the conditions are true, the controller then executes the statements within the loop. The statements in a REPEAT...UNTIL loop are always executed at least once. The statements in a WHILE...DO loop might never be executed.</p>	<pre> pos := -1; REPEAT pos := pos + 2; UNTIL ((pos = 101) OR (structarray[pos].value = targetvalue)) end_repeat; </pre>

Example 2:

If you want this:	Enter this structured text:
<p>Move ASCII characters from a SINT array into a string tag. (In a SINT array, each element holds one character.) Stop when you reach the carriage return.</p> <ol style="list-style-type: none"> 1. Initialize <i>Element_number</i> to 0. 2. Count the number of elements in <i>SINT_array</i> (array that contains the ASCII characters) and store the result in <i>SINT_array_size</i> (DINT tag). 3. Set <i>String_tag[element_number]</i> = the character at <i>SINT_array[element_number]</i>. 4. Add 1 to <i>element_number</i>. This lets the controller check the next character in <i>SINT_array</i>. 5. Set the Length member of <i>String_tag</i> = <i>element_number</i>. (This records the number of characters in <i>String_tag</i> so far.) 6. If <i>element_number</i> = <i>SINT_array_size</i>, then stop. (You are at the end of the array and it does not contain a carriage return.) 7. If the character at <i>SINT_array[element_number]</i> = 13 (decimal value of the carriage return), then stop. Otherwise, go to 3. 	<pre> element_number := 0; SIZE(SINT_array, 0, SINT_array_size); Repeat String_tag.DATA[element_number] := SINT_array[element_number]; element_number := element_number + 1; String_tag.LEN := element_number; If element_number = SINT_array_size then exit; end_if; Until SINT_array[element_number] = 13 end_repeat; </pre>

Comments

To make your structured text easier to interpret, add comments to it.

- Comments let you use plain language to describe how your structured text works.
- Comments do not affect the execution of the structured text.

To add comments to your structured text:

To add a comment:	Use one of these formats:
on a single line	<i>//comment</i>
at the end of a line of structured text	<i>(*comment*)</i> <i>/*comment*/</i>
within a line of structured text	<i>(*comment*)</i> <i>/*comment*/</i>
that spans more than one line	<i>(*start of comment . . . end of comment*)</i> <i>/*start of comment . . . end of comment*/</i>

For example:

Format:	Example:
<i>//comment</i>	At the beginning of a line <i>//Check conveyor belt direction</i> IF conveyor_direction THEN... At the end of a line ELSE <i>//If conveyor isn't moving, set alarm light</i> light := 1; END_IF;
<i>(*comment*)</i>	Sugar.Inlet[:=]1; <i>(*open the inlet*)</i> IF Sugar.Low <i>(*low level LS*)</i> & Sugar.High <i>(*high level LS*)</i> THEN... <i>(*Controls the speed of the recirculation pump. The speed depends on the temperature in the tank.*)</i> IF tank.temp > 200 THEN...
<i>/*comment*/</i>	Sugar.Inlet:=0; <i>/*close the inlet*/</i> IF bar_code=65 <i>/*A*/</i> THEN... <i>/*Gets the number of elements in the Inventory array and stores the value in the Inventory_Items tag*/</i> SIZE(Inventory,0,Inventory_Items);

A

ABL instruction 16-5
ABS instruction 5-29
absolute value 5-29
ACB instruction 16-8
ACL instruction 16-10
ACS instruction 13-14
ADD instruction 5-6
addition 5-6
advanced math instructions
 introduction 14-1
 LN 14-2
 LOG 14-4
 XPY 14-6
AFI instruction 10-23
AHL instruction 16-12
alarms 12-28
all mode 7-2
always false instruction 10-23
AND instruction 6-23
arc cosine 13-14
arc sine 13-11
arc tangent 13-17
ARD instruction 16-16
arithmetic operators
 structured text C-6
arithmetic status flags
 overflow B-8
ARL instruction 16-19
array instructions
 AVE 7-38
 BSL 8-2
 BSR 8-5
 COP 7-28
 CPS 7-28
 DDT 12-10
 FAL 7-7
 FBC 12-2
 FFL 8-8
 FFU 8-14
 file/misc. 7-1
 FLL 7-34
 FSC 7-19
 LFL 8-20
 LFU 8-26
 mode of operation 7-2
 RES 2-36
 sequencer 9-1
 shift 8-1
 SIZE 7-53
 SQI 9-2
 SQL 9-10

SQO 9-6
SRT 7-43
STD 7-48

ASCII

structured text assignment C-4

ASCII chars in buffer 16-8
ASCII clear buffer 16-10
ASCII handshake lines 16-12
ASCII instructions

ABL 16-5
ACB 16-8
ACL 16-10
AHL 16-12
ARD 16-16
ARL 16-19
AWA 16-23
AWT 16-28
CONCAT 17-3
DELETE 17-5
DTOS 18-8
FIND 17-7
INSERT 17-9
LOWER 18-14
MID 17-11
RTOS 18-10
STOD 18-4
STOR 18-6
SWPB 6-19
UPPER 18-12

ASCII read 16-16
ASCII read line 16-19
ASCII test for buffer line 16-5
ASCII write 16-28
ASCII write append 16-23
ASN instruction 13-11

assignment

ASCII character C-4
non-retentive C-3
retentive C-2

assume data available B-5, B-6, B-7

ATN instruction 13-17

attributes

converting data types A-1
immediate values A-1

AVE instruction 7-38

average 7-38

AWA instruction 16-23

AWT instruction 16-28

B

BAND 6-35

- bit field distribute** 6-11
- bit field distribute with target** 6-14
- bit instructions**
 - introduction 1-1
 - ONS 1-12
 - OSF 1-17
 - OSFI 1-22
 - OSR 1-15
 - OSRI 1-19
 - OTE 1-6
 - OTL 1-8
 - OTU 1-10
 - XIO 1-4
- bit shift left** 8-2
- bit shift right** 8-5
- bitwise AND** 6-23
- bitwise exclusive OR** 6-29
- bitwise NOT** 6-32
- bitwise operators**
 - structured text C-10
- bitwise OR** 6-26
- BNOT** 6-44
- BOOL expression**
 - structured text C-4
- Boolean AND** 6-35
- Boolean Exclusive OR** 6-41
- Boolean NOT** 6-44
- Boolean OR** 6-38
- BOR** 6-38
- break** 11-5
- BRK instruction** 11-5
- BSL instruction** 8-2
- BSR instruction** 8-5
- BTD instruction** 6-11
- BTDT instruction** 6-14
- BXOR** 6-41

C

- cache**
 - connection 3-31
- CASE** C-16
- clear** 6-17
- CLR instruction** 6-17
- CMP instruction** 4-2
- comments**
 - structured text C-28
- common attributes** A-1
 - converting data types A-1
 - immediate values A-1
- compare** 4-2

- compare instructions**
 - CMP 4-2
 - EQU 4-7
 - expression format 4-5, 7-25
 - GEQ 4-11
 - GRT 4-15
 - introduction 4-1
 - LEQ 4-19
 - LES 4-23
 - LIM 4-27
 - MEQ 4-33
 - NEQ 4-38
 - order of operation 4-5, 7-26
 - valid operators 4-4, 7-25
- COMPARE structure** 12-3, 12-11

- compute** 5-2

- compute instructions**
 - ABS 5-29
 - ADD 5-6
 - CPT 5-2
 - DIV 5-15
 - expression format 5-4, 7-17
 - introduction 5-1
 - MOD 5-19
 - MUL 5-12
 - NEG 5-26
 - order of operation 5-5, 7-18
 - SQR 5-23
 - SUB 5-9
 - valid operators 5-4, 7-17

- CONCAT instruction** 17-3

- configuring** 3-15
 - MSG instruction 3-15
 - PID instruction 12-26

- connection**
 - cache 3-31

- connector**
 - function block diagram B-1

- construct**
 - structured text C-12

- CONTROL structure** 7-8, 7-19, 7-39, 7-43, 7-48, 8-2, 8-5, 8-8, 8-14, 8-20, 8-26, 9-2, 9-6, 9-10

- control structure** 10-15

- CONTROLLER object** 3-37

- CONTROLLERDEVICE object** 3-37

- conversion instructions**
 - DEG 15-2
 - FRD 15-9
 - introduction 15-1
 - RAD 15-4

- TOD 15-6
- TRN 15-11
- convert to BCD** 15-6
- convert to integer** 15-9
- converting data types** A-1
- COP instruction** 7-28
- copy** 7-28
- COS instruction** 13-5
- cosine** 13-5
- count down** 2-28
- count up** 2-24
- count up/down** 2-32
- counter instructions**
 - CTD 2-28
 - CTU 2-24
 - CTUD 2-32
 - introduction 2-1
 - RES 2-36
- COUNTER structure** 2-24, 2-28
- CPS instruction** 7-28
- CPT instruction** 5-2
- CST object** 3-39
- CTD instruction** 2-28
- CTU instruction** 2-24
- CTUD instruction** 2-32

D

- data transitional** 12-18
- DDT instruction**
 - operands 12-10
 - search mode 12-12
- deadband** 12-38
- DEG instruction** 15-2
- degree** 15-2
- DELETE instruction** 17-5
- description**
 - structured text C-28
- DF1 object** 3-40
- diagnostic detect** 12-10
- DINT to String** 18-8
- DIV instruction** 5-15
- division** 5-15
- document**
 - structured text C-28
- DTOS instruction** 18-8
- DTR instruction** 12-18

E

- elements**

- SIZE instruction 7-53
- end of transition instruction** 10-25
- EOT instruction** 10-25
- EQU instruction** 4-7
- equal to** 4-7
- error codes**
 - ASCII 16-4
 - MSG instruction 3-8
- EVENT instruction** 10-31
- event task**
 - configure 3-51
 - trigger via consumed tag 3-57
 - trigger via EVENT instruction 10-31
- examine if open** 1-4
- execution order** B-4
- exponential** 14-6
- expression**
 - BOOL expression
 - structured text C-4
 - numeric expression
 - structured text C-4
 - order of execution
 - structured text C-10
 - structured text
 - arithmetic operators C-6
 - bitwise operators C-10
 - functions C-6
 - logical operators C-9
 - overview C-4
 - relational operators C-7
- expressions**
 - format 4-5, 5-4, 7-17, 7-25
 - order of operation 4-5, 5-5, 7-18, 7-26
 - valid operators 4-4, 5-4, 7-17, 7-25

F

- FAL instruction**
 - mode of operation 7-2
 - operands 7-7
- FAULTLOG object** 3-43
- FBC instruction**
 - operands 12-2
 - search mode 12-4
- FBD_BIT_FIELD_DISTRIBUTE structure** 6-14
- FBD_BOOLEAN_AND structure** 6-35
- FBD_BOOLEAN_NOT structure** 6-44
- FBD_BOOLEAN_OR structure** 6-38
- FBD_BOOLEAN_XOR structure** 6-41

- FBD_COMPARE structure** 4-8, 4-12, 4-16, 4-20, 4-24, 4-39
 - FBD_CONVERT structure** 15-6, 15-9
 - FBD_COUNTER structure** 2-32
 - FBD_LIMIT structure** 4-28
 - FBD_LOGICAL structure** 6-24, 6-27, 6-30, 6-32
 - FBD_MASK_EQUAL structure** 4-34
 - FBD_MASKED_MOVE structure** 6-8
 - FBD_MATH structure** 5-7, 5-10, 5-13, 5-16, 5-20, 5-26, 14-7
 - FBD_MATH_ADVANCED structure** 5-23, 5-29, 13-2, 13-5, 13-8, 13-11, 13-14, 13-17, 14-2, 14-4, 15-2, 15-4
 - FBD_ONESHOT structure** 1-19, 1-22
 - FBD_TIMER structure** 2-14, 2-17, 2-20
 - FBD_TRUNCATE structure** 15-11
 - feedback loop**
 - function block diagram B-5
 - feedforward** 12-39
 - FFL instruction** 8-8
 - FFU instruction** 8-14
 - FIFO load** 8-8
 - FIFO unload** 8-14
 - file arithmetic and logic** 7-7
 - file bit comparison** 12-2
 - file fill** 7-34
 - file instructions. See array instructions**
 - file search and compare** 7-19
 - FIND instruction** 17-7
 - Find String** 17-7
 - FLL instruction** 7-34
 - FOR instruction** 11-2
 - for/break instructions**
 - BRK 11-5
 - FOR 11-2
 - introduction 11-1
 - RET 11-6
 - FOR...DO** C-19
 - FRD instruction** 15-9
 - FSC instruction**
 - mode of operation 7-2
 - operands 7-19
 - function block diagram**
 - choose elements B-1
 - create a scan delay B-7
 - resolve a loop B-5
 - resolve data flow between blocks B-6
 - functions**
 - structured text C-6
- ## G
- GEQ instruction** 4-11
 - get system value** 3-34
 - greater than** 4-15
 - greater than or equal to** 4-11
 - GRT instruction** 4-15
 - GSV instruction**
 - objects 3-36
 - operands 3-34
- ## I
- ICON** B-1
 - IF...THEN** C-13
 - immediate output instruction** 3-57
 - immediate values** A-1
 - incremental mode** 7-5
 - inhibit**
 - task 3-51
 - input reference** B-1
 - input wire connector** B-1
 - input/output instructions**
 - GSV 3-34
 - introduction 3-1
 - IOT 3-57
 - MSG 3-2
 - SSV 3-34
 - INSERT instruction** 17-9
 - Insert String** 17-9
 - instructions**
 - advanced math 14-1
 - array
 - ASCII conversion 18-1
 - ASCII serial port 16-1
 - ASCII string manipulation 17-1
 - bit 1-1
 - compare 4-1
 - compute 5-1
 - conversion 15-1
 - counter 2-1
 - for/break 11-1
 - input/output 3-1
 - logical 6-1
 - math conversion 15-1
 - move 6-1
 - program control 10-1
 - sequencer 9-1
 - serial port 16-1
 - shift 8-1

- special 12-1
- string conversion 18-1
- string manipulation 17-1
- timer 2-1
- trigonometric 13-1
- IOT instruction** 3-57
- IREF** B-1

J

- JMP instruction** 10-2
- JSR instruction** 10-4
- jump** 10-2
- jump to subroutine** 10-4
- JXR instruction**
 - control structure 10-15

L

- label** 10-2
- latching data** B-2
- LBL instruction** 10-2
- LEQ instruction** 4-19
- LES instruction** 4-23
- less than** 4-23
- less than or equal to** 4-19
- LFL instruction** 8-20
- LFU instruction** 8-26
- LIFO load** 8-20
- LIFO unload** 8-26
- LIM instruction** 4-27
- limit** 4-27
- LN instruction** 14-2
- log**
 - base 10 14-4
 - natural 14-2
- log base 10** 14-4
- LOG instruction** 14-4
- logical instructions**
 - AND 6-23
 - introduction 6-1
 - NOT 6-32
 - OR 6-26
 - XOR 6-29
- logical operators**
 - structured text C-9
- lower case** 18-14
- LOWER instruction** 18-14

M

- masked equal to** 4-33

- masked move** 6-5
- masked move with target** 6-8
- masks** 12-19
- master control reset** 10-19
- math conversion instructions**
 - DEG 15-2
 - FRD 15-9
 - introduction 15-1
 - RAD 15-4
 - TOD 15-6
 - TRN 15-11
- math operators**
 - structured text C-6
- MCR instruction** 10-19
- MEQ instruction** 4-33
- message** 3-2
 - cach connections 3-31
 - programming guidelines 3-33
- MESSAGE object** 3-44
- MESSAGE structure** 3-2
- MID instruction** 17-11
- Middle String** 17-11
- mixing data types** A-1
- MOD instruction** 5-19
- mode of operation** 7-2
- MODULE object** 3-46
- modulo division** 5-19
- MOTIONGROUP object** 3-47
- MOV instruction** 6-3
- move** 6-3
- move instructions**
 - BTD 6-11
 - BTDT 6-14
 - CLR 6-17
 - introduction 6-1
 - MOV 6-3
 - MVM 6-5
 - MVMT 6-8
- move/logical instructions**
 - BAND 6-35
 - BNOT 6-44
 - BOR 6-38
 - BXOR 6-41
- MSG instruction** 3-15
 - cache connection 3-31
 - communication method 3-30
 - error codes 3-8
 - operands 3-2
 - programming guidelines 3-33
 - structure 3-2
- MUL instruction** 5-12

multiplication 5-12
MVM instruction 6-5
MVMT instruction 6-8

N

natural log 14-2
NEG instruction 5-26
negate 5-26
NEQ instruction 4-38
no operation 10-24
NOP instruction 10-24
not equal to 4-38
NOT instruction 6-32
numeric expression C-4
numerical mode 7-3

O

objects

CONTROLLER 3-37
 CONTROLLERDEVICE 3-37
 CST 3-39
 DF1 3-40
 FAULTLOG 3-43
 GSV/SSV instruction 3-36
 MESSAGE 3-44
 MODULE 3-46
 MOTIONGROUP 3-47
 PROGRAM 3-48
 ROUTINE 3-49
 SERIALPORT 3-49
 TASK 3-51
 WALLCLOCKTIME 3-53

OCON B-1

one shot 1-12
one shot falling 1-17
one shot falling with input 1-22
one shot rising 1-15
one shot rising with input 1-19
ONS instruction 1-12
operators 4-4, 5-4, 7-17, 7-25

order of execution
 structured text C-10

OR instruction 6-26

order of execution B-4

structured text expression C-10

order of operation 4-5, 5-5, 7-18, 7-26

OREF B-1

OSF instruction 1-17

OSFI instruction 1-22

OSR instruction 1-15

OSRI instruction 1-19

OTE instruction 1-6

OTL instruction 1-8

OTU instruction 1-10

output

enable or disable end-of-task processing
 3-51

update immediately 3-57

output biasing 12-39

output energize 1-6

output latch 1-8

output reference B-1

output unlatch 1-10

output wire connector B-1

overflow conditions B-8

overlap

check for task overlap 3-51

P

pause SFC instruction 10-27

PID instruction

alarms 12-28
 configuring 12-26
 deadband 12-38
 feedforward 12-39
 operands 12-21
 output biasing 12-39
 scaling 12-29
 tuning 12-27

PID structure 12-22

postscan

structured text C-3

product codes 3-37

program control instructions

AFI 10-23
 EOT 10-25
 EVENT 10-31
 introduction 10-1
 JMP 10-2
 JSR 10-4
 LBL 10-2
 MCR 10-19
 NOP 10-24
 RET 10-4
 SBR 10-4
 TND 10-17
 UID 10-21
 UIE 10-21

PROGRAM object 3-48

program/operator control

overview B-14

proportional, integral, and derivative
12-21

R

RAD instruction 15-4
radians 15-4
REAL to String 18-10
relational operators
 structured text C-7
REPEAT...UNTIL C-25
RES instruction 2-36
reset 2-36
reset SFC instruction 10-29
RESULT structure 12-3, 12-11
RET instruction 10-4, 11-6
retentive timer on 2-10
retentive timer on with reset 2-20
return 10-4, 11-6
ROUTINE object 3-49
RTO instruction 2-10
RTOR instruction 2-20
RTOS instruction 18-10

S

SBR instruction 10-4
scaling 12-29
scan delay
 function block diagram B-7
search mode 12-4, 12-12
search string 17-7
sequencer input 9-2
sequencer instructions
 introduction 9-1
 SQL 9-2
 SQL 9-10
 SQO 9-6
sequencer load 9-10
sequencer output 9-6
serial port instructions
 ABL 16-5
 ACB 16-8
 ACL 16-10
 AHL 16-12
 ARD 16-16
 ARL 16-19
 AWA 16-23
 AWT 16-28
 introduction 16-1

SERIAL_PORT_CONTROL structure
 16-2, 16-4, 16-5, 16-8, 16-13,
 16-17, 16-20, 16-24, 16-29

SERIALPORT object 3-49

set system value 3-34

SFP instruction 10-27

SFR instruction 10-29

shift instructions

BSL 8-2
 BSR 8-5
 FFL 8-8
 FFU 8-14
 introduction 8-1
 LFL 8-20
 LFU 8-26

SIN instruction 13-2

sine 13-2

size in elements 7-53

SIZE instruction 7-53

sort 7-43

special instructions

DDT 12-10
 DTR 12-18
 FBC 12-2
 introduction 12-1
 PID 12-21
 SFP 10-27
 SFR 10-29

SQL instruction 9-2

SQL instruction 9-10

SQO instruction 9-6

SQR instruction 5-23

square root 5-23

SRT instruction 7-43

SSV instruction

objects 3-36
 operands 3-34

standard deviation 7-48

status

task 3-51

STD instructions 7-48

STOD instruction 18-4

STOR instruction 18-6

string

evaluation in structured text C-8

String Concatenate 17-3

string conversion instructions

DTOS 18-8
 introduction 18-1
 LOWER 18-14
 RTOS 18-10

- STOD 18-4
 - STOR 18-6
 - SWPB 6-19
 - UPPER 18-12
 - string data type** 16-3, 17-2, 18-3
 - String Delete** 17-5
 - string manipulation instructions**
 - CONCAT 17-3
 - DELETE 17-5
 - FIND 17-7
 - INSERT 17-9
 - introduction 17-1
 - MID 17-11
 - STRING structure** 16-3, 17-2, 18-3
 - String To DINT** 18-4
 - String To REAL** 18-6
 - structured text**
 - arithmetic operators C-6
 - assign ASCII character C-4
 - assignment C-2
 - bitwise operators C-10
 - CASE C-16
 - comments C-28
 - components C-1
 - constructs C-12
 - evaluation of strings C-8
 - expression C-4
 - FOR...DO C-19
 - functions C-6
 - IF...THEN C-13
 - logical operators C-9
 - non-retentive assignment C-3
 - numeric expression C-4
 - relational operators C-7
 - REPEAT...UNTIL C-25
 - WHILE...DO C-22
 - structures**
 - COMPARE 12-3, 12-11
 - CONTROL 7-8, 7-19, 7-39, 7-43, 7-48, 8-2, 8-5, 8-8, 8-14, 8-20, 8-26, 9-2, 9-6, 9-10
 - COUNTER 2-24, 2-28
 - FBD_BIT_FIELD_DISTRIBUTE 6-14
 - FBD_BOOLEAN_AND 6-35
 - FBD_BOOLEAN_NOT 6-44
 - FBD_BOOLEAN_OR 6-38
 - FBD_BOOLEAN_XOR 6-41
 - FBD_COMPARE 4-8, 4-12, 4-16, 4-20, 4-24, 4-39
 - FBD_CONVERT 15-6, 15-9
 - FBD_COUNTER 2-32
 - FBD_LIMIT 4-28
 - FBD_LOGICAL 6-24, 6-27, 6-30, 6-32
 - FBD_MASK_EQUAL 4-34
 - FBD_MASKED_MOVE 6-8
 - FBD_MATH 5-7, 5-10, 5-13, 5-16, 5-20, 5-26, 14-7
 - FBD_MATH_ADVANCED 5-23, 5-29, 13-2, 13-5, 13-8, 13-11, 13-14, 13-17, 14-2, 14-4, 15-2, 15-4
 - FBD_ONESHOT 1-19, 1-22
 - FBD_TIMER 2-14, 2-17, 2-20
 - FBD_TRUNCATE 15-11
 - MESSAGE 3-2
 - PID 12-22
 - RES instruction 2-36
 - RESULT 12-3, 12-11
 - SERIAL_PORT_CONTROL 16-2, 16-4, 16-5, 16-8, 16-13, 16-17, 16-20, 16-24, 16-29
 - STRING 16-3, 17-2, 18-3
 - string 16-3, 17-2, 18-3
 - TIMER 2-2, 2-6, 2-10
 - SUB instruction** 5-9
 - subroutine** 10-4
 - subtraction** 5-9
 - swap byte** 6-19
 - SWPB instruction** 6-19
 - synchronous copy** 7-28
- ## T
- TAN instruction** 13-8
 - tangent** 13-8
 - task**
 - configure programmatically 3-51
 - inhibit 3-51
 - monitor 3-51
 - trigger event task 10-31
 - trigger via consumed tag 3-57
 - TASK object** 3-51
 - temporary end** 10-17
 - timeout**
 - configure for event task 3-51
 - timer instructions**
 - introduction 2-1
 - RES 2-36
 - RTO 2-10
 - RTOR 2-20
 - TOF 2-6
 - TOFR 2-17
 - TON 2-2
 - TONR 2-14

timer off delay 2-6
timer off delay with reset 2-17
timer on delay 2-2
timer on delay with reset 2-14
TIMER structure 2-2, 2-6, 2-10
timing modes B-9
TND instruction 10-17
TOD instruction 15-6
TOF instruction 2-6
TOFR instruction 2-17
TON instruction 2-2
TONR instruction 2-14
trigger event task 10-31
trigger event task instruction 10-31
trigonometric instructions
 ACS 13-14
 ASN 13-11
 ATN 13-17
 COS 13-5
 introduction 13-1
 SIN 13-2
 TAN 13-8
TRN instruction 15-11
truncate 15-11
tuning 12-27

U

UID instruction 10-21
UIE instruction 10-21
unresolved loop
 function block diagram B-5
update output 3-57
upper case 18-12
UPPER instruction 18-12
user interrupt disable 10-21
user interrupt enable 10-21

W

WALLCLOCKTIME object 3-53
WHILE...DO C-22

X

X to the power of Y 14-6
XIO instruction 1-4
XOR instruction 6-29
XPY instruction 14-6



How Are We Doing?

Your comments on our technical publications will help us serve you better in the future.
Thank you for taking the time to provide us feedback.

You can complete this form and mail it back to us, visit us online at www.ab.com/manuals, or email us at RADocumentComments@ra.rockwell.com

Pub. Title/Type Logix5000™ Controllers General Instructions

Cat. No. 1756 ControlLogix®, 1769 CompactLogix™, 1789 SoftLogix™, 1794 FlexLogix™, PowerFlex 700S with DriveLogix
Pub. No. 1756-RM003G-EN-P
Pub. Date June 2003
Part No. 957726-86

Please complete the sections below. Where applicable, rank the feature (1=needs improvement, 2=satisfactory, and 3=outstanding).

Overall Usefulness	1	2	3	How can we make this publication more useful for you?
Completeness (all necessary information is provided)	1	2	3	Can we add more information to help you?
				procedure/step illustration feature
				example guideline other
				explanation definition
Technical Accuracy (all provided information is correct)	1	2	3	Can we be more accurate?
				text illustration
Clarity (all provided information is easy to understand)	1	2	3	How can we make things clearer?
Other Comments				You can add additional comments on the back of this form.

Your Name _____ Location/Phone _____
Your Title/Function _____ Would you like us to contact you regarding your comments?
____ No, there is no need to contact me
____ Yes, please call me
____ Yes, please email me at _____
____ Yes, please contact me via _____

Return this form to: Allen-Bradley Marketing Communications, 1 Allen-Bradley Dr., Mayfield Hts., OH 44124-9705
Phone: 440-646-3176 Fax: 440-646-3525 Email: RADocumentComments@ra.rockwell.com

PLEASE FASTEN HERE (DO NOT STAPLE)

Other Comments

PLEASE FOLD HERE

PLEASE REMOVE

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 18235 CLEVELAND OH

POSTAGE WILL BE PAID BY THE ADDRESSEE



**Rockwell
Automation**

1 ALLEN-BRADLEY DR
MAYFIELD HEIGHTS OH 44124-9705



ASCII Character Codes

Character	Dec	Hex	Character	Dec	Hex	Character	Dec	Hex	Character	Dec	Hex
[ctrl-@] NUL	0	\$00	SPACE	32	\$20	@	64	\$40	'	96	\$60
[ctrl-A] SOH	1	\$01	!	33	\$21	A	65	\$41	a	97	\$61
[ctrl-B] STX	2	\$02	"	34	\$22	B	66	\$42	b	98	\$62
[ctrl-C] ETX	3	\$03	#	35	\$23	C	67	\$43	c	99	\$63
[ctrl-D] EOT	4	\$04	\$	36	\$24	D	68	\$44	d	100	\$64
[ctrl-E] ENQ	5	\$05	%	37	\$25	E	69	\$45	e	101	\$65
[ctrl-F] ACK	6	\$06	&	38	\$26	F	70	\$46	f	102	\$66
[ctrl-G] BEL	7	\$07	'	39	\$27	G	71	\$47	g	103	\$67
[ctrl-H] BS	8	\$08	(40	\$28	H	72	\$48	h	104	\$68
[ctrl-I] HT	9	\$09)	41	\$29	I	73	\$49	i	105	\$69
[ctrl-J] LF	10	\$1 (\$0A)	*	42	\$2A	J	74	\$4A	j	106	\$6A
[ctrl-K] VT	11	\$0B	+	43	\$2B	K	75	\$4B	k	107	\$6B
[ctrl-L] FF	12	\$0C	,	44	\$2C	L	76	\$4C	l	108	\$6C
[ctrl-M] CR	13	\$r (\$0D)	-	45	\$2D	M	77	\$4D	m	109	\$6D
[ctrl-N] SO	14	\$0E	.	46	\$2E	N	78	\$4E	n	110	\$6E
[ctrl-O] SI	15	\$0F	/	47	\$2F	O	79	\$4F	o	111	\$6F
[ctrl-P] DLE	16	\$10	0	48	\$30	P	80	\$50	p	112	\$70
[ctrl-Q] DC1	17	\$11	1	49	\$31	Q	81	\$51	q	113	\$71
[ctrl-R] DC2	18	\$12	2	50	\$32	R	82	\$52	r	114	\$72
[ctrl-S] DC3	19	\$13	3	51	\$33	S	83	\$53	s	115	\$73
[ctrl-T] DC4	20	\$14	4	52	\$34	T	84	\$54	t	116	\$74
[ctrl-U] NAK	21	\$15	5	53	\$35	U	85	\$55	u	117	\$75
[ctrl-V] SYN	22	\$16	6	54	\$36	V	86	\$56	v	118	\$76
[ctrl-W] ETB	23	\$17	7	55	\$37	W	87	\$57	w	119	\$77
[ctrl-X] CAN	24	\$18	8	56	\$38	X	88	\$58	x	120	\$78
[ctrl-Y] EM	25	\$19	9	57	\$39	Y	89	\$59	y	121	\$79
[ctrl-Z] SUB	26	\$1A	:	58	\$3A	Z	90	\$5A	z	122	\$7A
ctrl-[ESC	27	\$1B	;	59	\$3B	[91	\$5B	{	123	\$7B
[ctrl-\] FS	28	\$1C	<	60	\$3C	\	92	\$5C		124	\$7C
ctrl-] GS	29	\$1D	=	61	\$3D]	93	\$5D	}	125	\$7D
[ctrl-^] RS	30	\$1E	>	62	\$3E	^	94	\$5E	~	126	\$7E
[ctrl-_] US	31	\$1F	?	63	\$3F	_	95	\$5F	DEL	127	\$7F

Rockwell Automation Support

Rockwell Automation provides technical information on the web to assist you in using our products. At <http://support.rockwellautomation.com>, you can find technical manuals, a knowledge base of FAQs, technical and application notes, sample code and links to software service packs, and a MySupport feature that you can customize to make the best use of these tools.

For an additional level of technical phone support for installation, configuration and troubleshooting, we offer TechConnect Support programs. For more information, contact your local distributor or Rockwell Automation representative, or visit <http://support.rockwellautomation.com>.

Installation Assistance

If you experience a problem with a hardware module within the first 24 hours of installation, please review the information that's contained in this manual. You can also contact a special Customer Support number for initial help in getting your module up and running:

United States	1.440.646.3223 Monday – Friday, 8am – 5pm EST
Outside United States	Please contact your local Rockwell Automation representative for any technical support issues.

New Product Satisfaction Return

Rockwell tests all of our products to ensure that they are fully operational when shipped from the manufacturing facility. However, if your product is not functioning and needs to be returned:

United States	Contact your distributor. You must provide a Customer Support case number (see phone number above to obtain one) to your distributor in order to complete the return process.
Outside United States	Please contact your local Rockwell Automation representative for return procedure.

www.rockwellautomation.com

Corporate Headquarters

Rockwell Automation, 777 East Wisconsin Avenue, Suite 1400, Milwaukee, WI, 53202-5302 USA, Tel: (1) 414.212.5200, Fax: (1) 414.212.5201

Headquarters for Allen-Bradley Products, Rockwell Software Products and Global Manufacturing Solutions

Americas: Rockwell Automation, 1201 South Second Street, Milwaukee, WI 53204-2496 USA, Tel: (1) 414.382.2000, Fax: (1) 414.382.4444

Europe: Rockwell Automation SA/NV, Vorstlaan/Boulevard du Souverain 36-BP 3A/B, 1170 Brussels, Belgium, Tel: (32) 2 663 0600, Fax: (32) 2 663 0640

Asia Pacific: Rockwell Automation, 27/F Citicorp Centre, 18 Whitfield Road, Causeway Bay, Hong Kong, Tel: (852) 2887 4788, Fax: (852) 2508 1846

Headquarters for Dodge and Reliance Electric Products

Americas: Rockwell Automation, 6040 Ponders Court, Greenville, SC 29615-4617 USA, Tel: (1) 864.297.4800, Fax: (1) 864.281.2433

Europe: Rockwell Automation, Brühlstraße 22, D-74834 Elztal-Dallau, Germany, Tel: (49) 6261 9410, Fax: (49) 6261 17741

Asia Pacific: Rockwell Automation, 55 Newton Road, #11-01/02 Revenue House, Singapore 307987, Tel: (65) 351 6723, Fax: (65) 355 1733



Allen-Bradley

Logix5000™ Controllers General Instructions

Reference Manual