Reference Manual

Allen-Bradley

Logix5000 Controllers General Instructions Reference Manual

Catalog Numbers 1756 ControlLogix, 1768 CompactLogix, 1769 CompactLogix, 1789 SoftLogix, 1794 FlexLogix, PowerFlex 700S with DriveLogix

Allen-Bradley • Rockwell Software

Rockwell Automation

Important User Information

Solid-state equipment has operational characteristics differing from those of electromechanical equipment. Safety Guidelines for the Application, Installation and Maintenance of Solid State Controls (publication SGI-1.1 available from your local Rockwell Automation® sales office or online at http://www.rockwellautomation.com/literature/) describes some important differences between solid-state equipment and hard-wired electromechanical devices. Because of this difference, and also because of the wide variety of uses for solid-state equipment, all persons responsible for applying this equipment must satisfy themselves that each intended application of this equipment is acceptable.

In no event will Rockwell Automation, Inc. be responsible or liable for indirect or consequential damages resulting from the use or application of this equipment.

The examples and diagrams in this manual are included solely for illustrative purposes. Because of the many variables and requirements associated with any particular installation, Rockwell Automation, Inc. cannot assume responsibility or liability for actual use based on the examples and diagrams.

No patent liability is assumed by Rockwell Automation, Inc. with respect to use of information, circuits, equipment, or software described in this manual.

Throughout this manual, when necessary, we use notes to make you aware of safety considerations.

	**WARNING:** Identifies information about practices or circumstances that can cause an explosion in a hazardous environment, which may lead to personal injury or death, property damage, or economic loss.
	**ATTENTION:** Identifies information about practices or circumstances that can lead to personal injury or death, property damage, or economic loss. Attentions help you identify a hazard, avoid a hazard, and recognize the consequence.
	**SHOCK HAZARD:** Labels may be on or inside the equipment, for example, a drive or motor, to alert people that dangerous voltage may be present.
	**BURN HAZARD:** Labels may be on or inside the equipment, for example, a drive or motor, to alert people that surfaces may reach dangerous temperatures.
IMPORTANT	Identifies information that is critical for successful application and understanding of the product.

This manual contains new and updated information. Changes throughout this revision are marked by change bars, as shown to the right of this paragraph.

Topic	Page
ALMD instruction operand descriptions and graphics for Relay Ladder Logic and Function Block Diagram languages	44
ALARM_DIGITAL structure input parameter descriptions	45
Shelve operands added to ALARM_DIGITAL structure output parameter descriptions	47
RSLinx software and alarm log data access. Replaced RSView software with FactoryTalkView software.	48
Relay Ladder action for Prescan condition	49
Structured Text and Function Block action for Prescan condition, EnableIn and Postscan condition	51
Corrected Relay Ladder Logix graphic with Shelved parameter	53
Corrected Function Block Diagram ALMD tag graphic	54
Analog alarm operand descriptions	55
ALARM_ANALOG structure input parameter descriptions	56
Shelve operands added to ALARM_ANALOG input parameter descriptions	57
Shelve operands added to ALARM_ANALOG output parameter descriptions	61
Relay Ladder action descriptions changed	65
Structured Text and Function Block action includes xShelved parameters and various other changes	66
Corrected RLL and FBD ALMA tag graphics	71
Corrected ALMD and ALMA tag graphic and Configuration tab graphics	72
Corrected ALMD and ALMA Messages tab graphics	74
Corrected ALMD and ALMA Status tab graphics	77
Added updated buffer alarm information	77
Added ProgUnshelve, OperShelve and OperSuppress elements to the Alarm Structure Elements in the Programmatically Access Alarm Information section	78
Corrected ALMA tag graphic	79
Corrected text for when an alarm is Disabled per version 21.00.00 changes, added Shelve, Suppress and Disable alarms information with state model for alarms	79
Updated State Alarms graphics	81
Controller memory use: added digital alarm memory consumption	83
Controller memory use: added analog alarm memory consumption	84
Scan execution times for analog and digital alarm instructions	85
CIP Generic messages	171
Watchdog and battery bit descriptions corrected for MajorFaultBits and MinorFaultBits	198
Important table added for ** operand and REALs for the CPT instruction.	257
Information added for ** operand for the CPT instruction.	260
Removed code 51 from DTOS Structured Text Fault Condition table	627
Removed code 51 from RTOS Structured Text Fault Condition table	629

Notes:

Chapter 2

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

Chapter 3

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

Chapter 4

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

Chapter 5

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

Chapter 6

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

Chapter 7

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

Chapter 8

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

Chapter 9

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

Chapter 10

Sequencer Instructions (SQI, SQO, SQL)

Chapter 11

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

Chapter 14

Trigonometric Instructions (SIN, COS, TAN, ASN, ASIN, ACS, ACOS, ATN, ATAN)

Chapter 15

Advanced Math Instructions (LN, LOG, XPY)

Chapter 16

Math Conversion Instructions (DEG, RAD, TOD, FRD, TRN, TRUNC)

Chapter 17

ASCII Serial Port Instructions (ABL, ACB, ACL, AHL, ARD, ARL, AWA, AWT)

Chapter 18

ASCII String Instructions (CONCAT, DELETE, FIND, INSERT, MID)

Chapter 19

ASCII Conversion Instructions (STOD, STOR, DTOS, RTOS, UPPER, LOWER)

Appendix C

Structured Text Programming

Index

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
Coordinate	Motion Coordinate System User Manual, publication MOTION-UM002
Motion	Logix5000 Controllers Motion Instructions Reference Manual, publication MOTION-RM002
PhaseManager™	PhaseManager User Manual, publication LOGIX-UM001
Process control	Logix5000 Controllers Process Control and Drives Instructions Reference Manual, publication 1756-RM006

Instruction	Location	Languages
ABL ASCII Test For Buffer Line	679	Relay ladder Structured text
ABS Absolute Value	286	Relay ladder Structured text Function block
ACB ASCII Chars in Buffer	582	Relay ladder Structured text
ACL ASCII Clear Buffer	584	Relay ladder Structured text
ACS Arc Cosine	540	Relay ladder Structured text Function block
ADD Add	261	Relay ladder Structured text Function block
AFI Always False Instruction	468	Relay ladder
AHL ASCII Handshake Lines	586	Relay ladder Structured text
ALM Alarm	Process control	Structured text Function block
ALMA Analog Alarm	55	Relay ladder Structured text Function block
ALMD Digital Alarm	44	Relay ladder Structured text Function block
AND Bitwise AND	312	Relay ladder Structured text Function block

Instruction	Location	Languages
ARD ASCII Read	590	Relay ladder Structured text
ARL ASCII Read Line	594	Relay ladder Structured text
ASN Arc Sine	537	Relay ladder Structured text Function block
ATN Arc Tangent	543	Relay ladder Structured text Function block
AVE File Average	375	Relay ladder
AWA ASCII Write Append	598	Relay ladder Structured text
AWT ASCII Write	602	Relay ladder Structured text
BAND Boolean AND	327	Structured text Function block
BNOT Boolean NOT	336	Structured text Function block
BOR Boolean OR	330	Structured text Function block
BPT Breakpoints	635	Relay ladder
BRK Break	485	Relay ladder
BSL Bit Shift Left	398	Relay ladder
BSR Bit Shift Right	402	Relay ladder
BTD Bit Field Distribute	299	Relay ladder
BTDT Bit Field Distribute with Target	302	Structured text Function block
BTR Message	176	Relay ladder Structured text
BTW Message	176	Relay ladder Structured text
BXOR Boolean Exclusive OR	333	Structured text Function block
CC Coordinated Control	Process control	Structured text Function block
CLR Clear	306	Relay ladder Structured text

Instruction	Location	Languages
CMP Compare	215	Relay ladder
CONCAT String Concatenate	610	Relay ladder Structured text
COP Copy File	365	Relay ladder Structured text
COS Cosine	531	Relay ladder Structured text Function block
CPS Synchronous Copy File	365	Relay Ladder Structured text
CPT Compute	257	Relay ladder
CTD Count Down	144	Relay ladder
CTU Count Up	140	Relay ladder
CTUD Count Up/Down	148	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	497	Relay ladder
DEDT Deadtime	Process control	Structured text Function block
DEG Degrees	560	Relay ladder Structured text Function block
DELETE String Delete	612	Relay ladder Structured text
DERV Derivative	Process control	Structured text Function block
DFF D Flip-Flop	Process control	Structured text Function block
DIV Divide	271	Relay ladder Structured text Function block
DTOS DINT to String	627	Relay ladder Structured text
DTR Data Transitional	504	Relay ladder
EOT End of Transition	470	Relay ladder Structured text

Instruction	Location	Languages
EQU Equal to	220	Relay ladder Structured text Function block
ESEL Enhanced Select	Process control	Structured text Function block
EVENT Trigger Event Task	476	Relay ladder Structured text
FAL File Arithmetic and Logic	345	Relay ladder
FBC File Bit Comparison	490	Relay ladder
FFL FIFO Load	406	Relay ladder
FFU FIFO Unload	412	Relay ladder
FGEN Function Generator	Process control	Structured text Function block
FIND Find String	614	Relay ladder Structured text
FLL File Fill	371	Relay ladder
FOR For	482	Relay ladder
FRD Convert to Integer	569	Relay ladder Function block
FSC File Search and Compare	357	Relay ladder
GEQ Greater than or Equal to	224	Relay ladder Structured text Function block
GRT Greater Than	228	Relay ladder Structured text Function block
GSV Get System Value	188	Relay ladder Structured text
HLL High/Low Limit	Process control	Structured text Function block
HPF High Pass Filter	Process control	Structured text Function block
ICON Input Wire Connector	653	Function block
IMC Internal Model Control	Process control	Structured text Function block
INSERT Insert String	616	Relay ladder Structured text

Instruction	Location	Languages
INTG Integrator	Process control	Structured text Function block
IOT Immediate Output	209	Relay ladder Structured text
IREF Input Reference	653	Function block
JKFF JK Flip-Flop	Process control	Structured text Function block
JMP Jump to Label	447	Relay ladder
JSR Jump to Subroutine	449	Relay ladder Structured text Function block
JXR Jump to External Routine	459	Relay ladder
LBL Label	447	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	232	Relay ladder Structured text Function block
LES Less Than	236	Relay ladder Structured text Function block
LFL LIFO Load	418	Relay ladder
LFU LIFO Unload	424	Relay ladder
LIM Limit	240	Relay ladder Function block
LN Natural Log	548	Relay ladder Structured text Function block
LOG Log Base 10	551	Relay ladder Structured text Function block
LOWER Lower Case	633	Relay ladder Structured text
LPF Low Pass Filter	Process control	Structured text Function block
MAAT Motion Apply Axis Tuning	Motion	Relay ladder Structured text

Instruction	Location	Languages
MAFR Motion Axis Fault Reset	Motion	Relay ladder Structured text
MAG Motion Axis Gear	Motion	Relay ladder Structured text
MAHD Motion Apply Hookup Diagnostics	Motion	Relay ladder Structured text
MAH Motion Axis Home	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
MASD Motion Axis Shutdown	Motion	Relay ladder Structured text
MAS Motion Axis Stop	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
MCCD Motion Coordinated Change Dynamics	Coordinate	Relay ladder Structured text
MCCM Motion Coordinated Circular Move	Coordinate	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	Coordinate	Relay ladder Structured text
MCR Master Control Reset	464	Relay ladder

Instruction	Location	Languages
MCSD Motion Coordinated Shutdown	Coordinate	Relay ladder Structured text
MCS Motion Coordinated Stop	Coordinate	Relay ladder Structured Text
MCSR Motion Coordinated Shutdown Reset	Coordinate	Relay ladder Structured text
MCT Motion Coordinated Transform	Coordinate	Relay ladder Structured text
MCTP Motion Calculate Transform Position	Coordinate	Relay ladder Structured text
MDAC Motion Master Driven Axis Control	Motion	Relay ladder Structured text
MDCC Motion Master Driven Coordinated Control	Motion	Relay ladder Structured text
MDF Motion Direct Drive Off	Motion	Relay ladder Structured text
MDOC Motion Disarm Output Cam	Motion	Relay ladder Structured text
MDO Motion Direct Drive On	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	246	Relay ladder Structured text Function Block
MGSD Motion Group Shutdown	Motion	Relay ladder Structured text
MGS Motion Group Stop	Motion	Relay ladder Structured text
MGSP Motion Group Strobe Position	Motion	Relay ladder Structured text
MGSR Motion Group Shutdown Reset	Motion	Relay ladder Structured text
MID Middle String	618	Relay ladder Structured text
MINC Minimum Capture	Process control	Structured text Function block
MMC Modular Multivariable Control	Process control	Structured text Function block

Instruction	Location	Languages
MOD Modulo	276	Relay ladder Structured text Function block
MOV Move	291	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	154	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	268	Relay ladder Structured text Function block
MUX Multiplexer	Process control	Function block
MVM Masked Move	293	Relay ladder
MVMT Masked Move with Target	296	Structured text Function block
NEG Negate	283	Relay ladder Structured text Function block
NEQ Not Equal to	251	Relay ladder Structured text Function block
NOP No Operation	469	Relay ladder
NOT Bitwise NOT	324	Relay ladder Structured text Function block
NTCH Notch Filter	Process control	Structured text Function block
OCON Output Wire Connector	653	Function block
ONS One Shot	102	Relay ladder
OR Bitwise OR	316	Relay ladder Structured text Function block
OREF Output Reference	653	Function block
OSFI One Shot Falling with Input	112	Structured text Function block
OSF One Shot Falling	107	Relay ladder
OSRI One Shot Rising with Input	109	Structured text Function block
OSR One Shot Rising	105	Relay ladder
OTE Output Energize	96	Relay ladder
OTL Output Latch	98	Relay ladder
OTU Output Unlatch	100	Relay ladder
PATT Attach to Equipment Phase	PhaseManager	Relay ladder Structured text
PCLF Equipment Phase Clear Failure	PhaseManager	Relay ladder Structured text
PCMD Equipment Phase Command	PhaseManager	Relay ladder Structured text
PDET Detach from Equipment Phase	PhaseManager	Relay ladder Structured text
PFL Equipment Phase Failure	PhaseManager	Relay ladder Structured text
PIDE Enhanced PID	Process control	Structured text Function block
PID Proportional Integral Derivative	507	Relay ladder Structured text
PI Proportional + Integral	Process control	Structured text Function block
PMUL Pulse Multiplier	Process control	Structured text Function block
POSP Position Proportional	Process control	Structured text Function block
POVR Equipment Phase Override Command	PhaseManager	Relay ladder Structured text
PPD Equipment Phase Paused	PhaseManager	Relay ladder Structured text

Instruction	Location	Languages
PRNP Equipment Phase New Parameters	PhaseManager	Relay ladder Structured text
PSC Phase State Complete	PhaseManager	Relay ladder Structured text
PXRQ Equipment Phase External Request	PhaseManager	Relay ladder Structured text
RAD Radians	563	Relay ladder Structured text Function block
RESD Reset Dominant	Process control	Structured text Function block
RES Reset	152	Relay ladder
RET Return	449 and 486	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	136	Relay ladder
RTOR Retentive Timer On with Reset	136	Structured text Function block
RTOS REAL to String	629	Relay ladder Structured text
SBR Subroutine	449	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	472	Relay ladder Structured text
SFR SFC Reset	474	Relay ladder Structured text
SIN Sine	528	Relay ladder Structured text Function block
SIZE Size In Elements	392	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	432	Relay ladder
SQL Sequencer Load	440	Relay ladder
SQO Sequencer Output	436	Relay ladder
SQR Square Root	280	Relay ladder Function block
SQRT Square Root	280	Structured text
SRT File Sort	380	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	188	Relay ladder Structured text
STD File Standard Deviation	385	Relay ladder
STOD String To DINT	623	Relay ladder Structured text
STOR String To REAL	625	Relay ladder Structured text
SUB Subtract	265	Relay ladder Structured text Function block
SWPB Swap Byte	308	Relay ladder Structured text
TAN Tangent	534	Relay ladder Structured text Function block
TND Temporary End	462	Relay ladder
TOD Convert to BCD	566	Relay ladder Function block
TOFR Timer Off Delay with Reset	132	Structured text Function block

Instruction	Location	Languages
TOF Timer Off Delay	120	Relay ladder
TONR Timer On Delay with Reset	128	Structured text Function block
TON Timer On Delay	116	Relay ladder
TOT Totalizer	Process control	Structured text Function block
TPT Tracepoints	639	Relay ladder
TRN Truncate	571	Relay ladder Function block
TRUNC Truncate	571	Structured text
UID User Interrupt Disable	466	Relay ladder Structured text
UIE User Interrupt Enable	466	Relay ladder Structured text
UPDN Up/Down Accumulator	Process control	Structured text Function block
UPPER Upper Case	631	Relay ladder Structured text
XIC Examine If Closed	90	Relay ladder
XIO Examine If Open	93	Relay ladder
XOR Bitwise Exclusive OR	320	Relay ladder Structured text Function block
XPY X to the Power of Y	554	Relay ladder Structured text Function block

Notes:

Introduction

This manual provides a programmer with details about each available instruction for a Logix-based controller.

Who Should Use This Manual

You should be familiar with how the Logix-based controller stores and processes data.

Novice programmers should read all of 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 is one of a set of related manuals that shows common procedures for programming and operating Logix5000™ controllers. For a complete list of common procedures manuals, see the Logix5000 Controllers Common Procedures Programming Manual, publication 1756-PM001.

The term Logix5000 controller refers to any controller that is based on the Logix5000 operating system, such as:

- CompactLogix™ controllers
- ControlLogix® controllers
- DriveLogix™ controllers
- FlexLogix™ controllers
- SoftLogix™5800 controllers

Table 1 - Description of Instruction Format

Section	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.

Table 1 - Description of Instruction Format

Section	Information
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.

Icon	Programming Language
	Relay ladder
	Structured text
	Function block

Common Information for All Instructions

The Logix5000 instruction set has some common attributes.

Information	Appendix
Common attributes	Common Attributes defines: • Arithmetic status flags • Data types • Keywords
Function block attributes	Function Block Attributes defines: • Program and operator control • Timing modes

Conventions and Related Terms

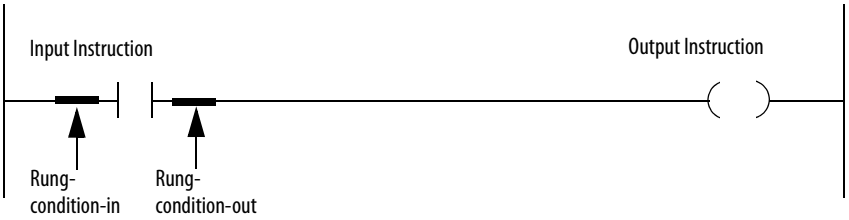
This manual uses set and clear to define the status of bits (Boolean) and values (non-Boolean).

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

- 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 $+/-10^{+/-15}$ because internal floating point calculations are done by 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 ($+/-10^{+/-38}$).

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 were 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.

Studio 5000 Environment

The Studio 5000™ Engineering and Design Environment combines engineering and design elements into a common environment. The first element in the Studio 5000 environment is the Logix Designer application. The Logix Designer application is the rebranding of RSLogix™ 5000 software and will continue to be the product to program Logix5000™ controllers for discrete, process, batch, motion, safety, and drive-based solutions.

Additional Resources

See these manuals and documents for more information about using motionmodules in a Logix5000 control system.

Resource	Description
Logix5000 Controllers Common Procedures Programming Manual, publication 1756-PM001	Provides links to a collection of programming manuals that describe how you can use procedures that are common to all Logix5000 controller projects.
Motion Coordinate System User Manual, publication MOTION-UM002	Provides information to create a coordinate system by using Logix5000 motion modules.
Logix5000 Controllers Motion Instructions Reference Manual, publication MOTION-RM002	Provides a programmer with details about the motion instructions that are available for a Logix5000 controller.
PhaseManager User Manual, publication LOGIX-UM001	Shows you how to set up and program a Logix5000 controller to use equipment phases.
Logix5000 Controllers Process Control and Drives Instructions Reference Manual, publication 1756-RM006	Provides a programmer with details about each available instruction for a Logix-based controller with regard to process control and drives.
PlantPAx Automation System Reference Manual, publication PROCES-RM001	Provides information about how to set up a PlantPAx system.

You can view or download Rockwell Automation publications at http:/www.rockwellautomation.com/literature/. To order paper copies of technical documentation, contact your local Allen-Bradley distributor or Rockwell Automation sales representative.

Notes:

FactoryTalk Alarms and Events Logix-based Instructions
(ALMD, ALMA)

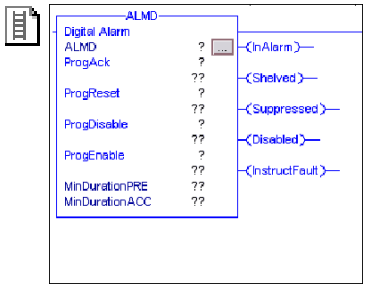
These Logix-based alarm instructions are available in relay ladder, structured text, and function block diagram. These instructions detect alarm conditions and publish alarm events that can be displayed and logged through FactoryTalkView Alarms and Events servers.

If you want to	Use this instruction	Available in	Page
Detect alarms based on Boolean (true/false) conditions	ALMD	Relay ladder Structured text Function block	44
Detect alarms based on the level or rate of change of a value	ALMA	Relay ladder Structured text Function block	55

Digital Alarm (ALMD)

The ALMD instruction detects alarms based on Boolean (true/false) conditions. Program (Prog) and operator (Oper) control parameters provide an interface for alarm commands.

Operands:



Relay Ladder

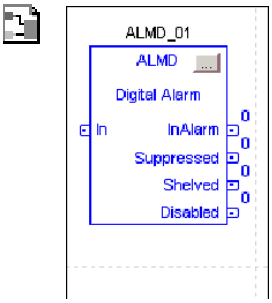
In relay ladder, the alarm condition input (In) is obtained from the rung condition.

Operand	Type	Format	Description
ALMD tag	ALARM_DIGITAL	Structure	Tag using ALMD structure.

Structured Text

```
ALMD(ALMD, In, ProgAck,
ProgReset, ProgDisable, ProgEnable);
```

The operands are the same as those for the relay ladder ALMD instruction, with one exception as indicated above.



Function Block

Operand	Type	Format	Description
ALMD tag	ALARM_DIGITAL	Structure	Tag using ALMD structure.

ALARM_DIGITAL Structure

Input Parameter	Data Type	Description
EnableIn	BOOL	**Relay Ladder** Corresponds to the rung state. Does not affect processing. **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 always executes.
In	BOOL	The digital signal input to the instruction. Default is cleared. **Relay Ladder** **Function Block** Follows the rung condition. Set if the rung condition is true. Cleared if the rung condition is false. **Structured Text** Copied from instruction operand.
InFault	BOOL	Bad health indicator for the input. The user application may set InFault to indicate the input signal has an error. When set, the instruction sets InFaulted (Status.1). When cleared, the instruction clears InFaulted (Status .1). In either case, the instruction continues to evaluate In for alarm conditions. Default is cleared (good health).
Condition	BOOL	Specifies how alarm is activated. When Condition is set, the alarm condition is activated when In is set. When Condition is cleared, the alarm condition is activated when In is cleared. Default is set.
AckRequired	BOOL	Specifies whether alarm acknowledgement is required. When set, acknowledgement is required. When cleared, acknowledgement is not required and Acked is always set. Default is set.
Latched	BOOL	Specifies whether the alarm is latched. Latched alarms remain InAlarm when the alarm condition becomes false, until a Reset command is received. When set, the alarm is latched. When cleared, the alarm is unlatched. A latched alarm can be reset only when the alarm condition is false. Default is cleared.
ProgAck	BOOL	For Relay Ladder Logic, on transition from cleared to set, acknowledges alarm (if acknowledgement is required). Default is cleared. **Relay Ladder** Copies value from the instruction operand. **Structured Text** Copies value from the instruction operand.
OperAck	BOOL	Set by the operator interface to acknowledge the alarm. Takes effect only if the alarm is unacknowledged. The instruction clears this parameter. Default is cleared.
ProgReset	BOOL	For Relay Ladder Logic, on transition from cleared to set, acknowledges alarm (if acknowledgement is required). Default is cleared. **Relay Ladder** Copied from the instruction operand. **Structured Text** Copied from the instruction operand.
OperReset	BOOL	Set by the operator interface to reset the latched alarm. Takes effect only if the latched alarm is InAlarm and the alarm condition is false. The alarm instruction clears this parameter. Default is cleared.
ProgSuppress	BOOL	Set by the user program to suppress the alarm. Default is cleared.
OperSuppress	BOOL	Set by the operator interface to suppress the alarm. The alarm instruction clears this parameter. Default is cleared.

Input Parameter	Data Type	Description
ProgUnsuppress	BOOL	Set by the user program to unsuppress the alarm. Takes precedence over Suppress commands. Default is cleared.
OperUnsuppress	BOOL	Set by the operator interface to unsuppress the alarm. Takes precedence over Suppress commands. The alarm instruction clears this parameter. Default is cleared.
OperShelve	BOOL	Set by the operator interface to shelve or reshelve the alarm. Takes action on transition from cleared to set. The instruction clears this parameter. Unshelve commands take precedence over Shelve commands. Connecting a push button to the OperShelve tag: The alarm instruction only processes the OperShelve tag on transition from cleared to set to prevent unwanted reshelving of the alarm. For example, if an operator presses a push button to shelve the alarm while the ProgUnshelve tag is set, the alarm is not shelved because the ProgUnshelve tag takes precedence. To shelve the alarm, the operator can release and press the push button again once ProgUnshelve is cleared.
ProgUnshelve	BOOL	Set by the user program to unshelve the alarm. Takes precedence over Shelve commands. Default is cleared. For more information on shelving an alarm, see the description for the OperShelve parameter.
OperUnshelve	BOOL	Set by the operator interface to unshelve the alarm. The alarm instruction clears this parameter. Takes precedence over Shelve commands. Default is cleared. For more information on shelving an alarm, see the description for the OperShelve parameter.
ProgDisable	BOOL	Set by the user program to disable the alarm. Takes action on transition from cleared to set. Default is cleared. **Relay Ladder** Copied from the instruction operand. **Structured Text** Copied from the instruction operand.
OperDisable	BOOL	Set by the operator interface to disable the alarm. The alarm instruction clears this parameter. Default is cleared.
ProgEnable	BOOL	Set by the user program to enable the alarm. Takes precedence over a Disable command. Takes action on transition from cleared to set. Default is cleared. **Relay Ladder** Copied from the instruction operand. **Structured Text** Copied from the instruction operand.
OperEnable	BOOL	Set by the operator interface to enable the alarm. Takes precedence over Disable command. The alarm instruction clears this parameter. Default is cleared.
AlarmCountReset	BOOL	Set by the operator interface to reset the alarm count to zero. The alarm instruction clears this parameter. Default is cleared.
UseProgTime	BOOL	Specifies whether to use the controller's clock or the ProgTime value to timestamp alarm state change events. When set, the ProgTime value provides timestamp. When cleared, the controller's clock provides timestamp. Default is cleared.
ProgTime	LINT	If UseProgTime is set, this value is used to provide the timestamp value for all events. This lets the application apply timestamps obtained from the alarm source, such as a sequence-of-events input module.
Severity	DINT	Severity of the alarm. This does not affect processing of alarms by the controller, but can be used for sorting and filtering functions at the alarm subscriber. Valid = 1...1000 (1000 = most severe; 1 = least severe). Default = 500.

Input Parameter	Data Type	Description
MinDurationPRE	DINT	Minimum duration preset (milliseconds) for the alarm condition to remain true before the alarm is marked as InAlarm and alarm notification is sent to clients. The controller collects alarm data as soon as the alarm condition is detected, so no data is lost while waiting to meet the minimum duration. Valid = 0...2,147,483,647. Default = 0. You do not want to use some other tag on the MinDurationPRE faceplate to control the state, then enter the ALMD tag .MinDurationPRE on the instruction faceplate, for example, myalmd.MinDurationPRE.
ShelveDuration	DINT	Length of time in minutes to shelve an alarm. Set by the operator interface to unshelve the alarm. The alarm instruction clears this parameter. Takes precedence over Shelve commands. Default is cleared. Minimum time is one minute. Maximum time is defined by MaxShelveDuration.
MaxShelveDuration	DINT	Maximum time duration in minutes for which an alarm can be shelved. For more information on shelving an alarm, see the description for ShelveDuration parameter.

Output Parameter	Data Type	Description
EnableOut	BOOL	Enable output.
InAlarm	BOOL	Alarm active status. Set when the alarm is active. Cleared when the alarm is not active (normal status).
Acked	BOOL	Alarm acknowledged status. Set when the alarm is acknowledged. Cleared when the alarm is not acknowledged. Acked is always set when AckRequired is cleared.
InAlarmUnack	BOOL	Combined alarm active and acknowledged status. Set when the alarm is active (InAlarm is set) and unacknowledged (Acked is cleared). Cleared when the alarm is normal (inactive), acknowledged, or both.
Suppressed	BOOL	Suppressed status of the alarm. Set when the alarm is suppressed. Cleared when the alarm is not suppressed.
Shelved	BOOL	Set by the operator to shelve the alarm. Takes action on transition from cleared to set. Cleared when alarm is unshelved.
Disabled	BOOL	Disabled status of the alarm. Set when the alarm is disabled. Cleared when the alarm is enabled.
Commissioned	BOOL	Reserved for future use.
MinDurationACC	DINT	Elapsed time since the alarm was detected. When this value reaches MinDurationPRE, the alarm becomes active (InAlarm is set), and a notification is sent to clients.
AlarmCount	DINT	Number of times the alarm has been activated (InAlarm is set). If the maximum value is reached, the counter leaves the value at the maximum count value.
InAlarmTime	LINT	Timestamp of alarm detection.
AckTime	LINT	Timestamp of alarm acknowledgement. If the alarm does not require acknowledgement, this timestamp is equal to alarm time.
RetToNormalTime	LINT	Timestamp of alarm returning to a normal state.
AlarmCountResetTime	LINT	Timestamp indicating when the alarm count was reset.
ShelveTime	LINT	Timestamp indicating when the alarm was shelved the last time. Each time the alarm is shelved, the timestamp is set to the current time. For more information on shelving an alarm, see the description for the Shelved parameter.
UnshelveTime	LINT	Timestamp indicating when the alarm is going to be unshelved. This value is set every time the alarm is shelved (even if the alarm has already been shelved). The timestamp is determined by adding the ShelveDuration to the current time. If the alarm is unshelved programmatically or by an operator, then the value is set to the current time. For more information on shelving an alarm, see the description for the Shelved parameter.
Status	DINT	Combined status indicators: Status.0 = InstructFault. Status.1= InFaulted. Status.2 = SeverityInv.

Output Parameter	Data Type	Description
InstructFault (Status.0)	BOOL	Instruction error conditions exist. This is not a minor or major controller error. Check the remaining status bits to determine what occurred.
InFaulted (Status.1)	BOOL	User program has set InFault to indicate bad quality input data. Alarm continues to evaluate In for alarm condition.
SeverityInv (Status.2)	BOOL	Alarm severity configuration is invalid. If severity <1, the instruction uses Severity = 1. If severity >1000, the instruction uses Severity = 1000.

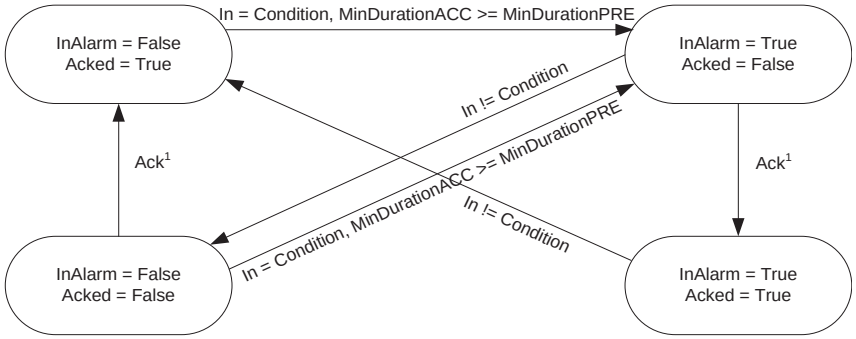
Description: The ALMD instruction detects alarms based on Boolean (true/false) conditions.

The ALMD instruction provides additional functionality when used with RSLinx® Enterprise and FactoryTalk View SE software. You can display alarms in the Alarm Summary, Alarm Banner, Alarm Status Explorer, and Alarm Log Viewer displays in FactoryTalk View SE software.

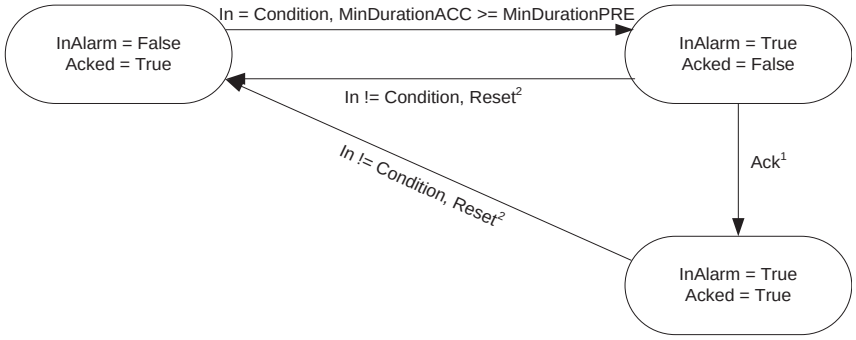
RSLinx Enterprise software subscribes to Alarm Log in the controller. If a connection to RSLinx Enterprise software is lost, the controller continues to store alarm data into the log and data can be accessed when the communication is restored.

State Diagrams When Acknowledgement Required

Latched = False



Latched = True

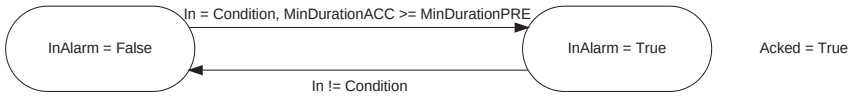


1. Alarm can be acked several different ways: ProgAck, OperAck, clients (Studio 5000 software, FactoryTalkView software.

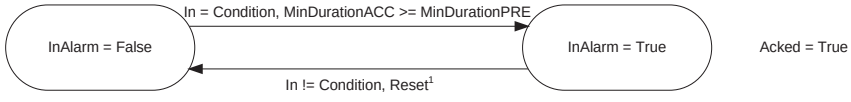
2. Alarm can be reset several different ways: ProgReset, OperReset, clients (Studio 5000 software, FactoryTalkView software.

State Diagrams When Acknowledgment Not Required

Latched = False



Latched = True



1. Alarm can be acked several different ways: ProgReset, OperReset, clients (Studio 5000 software, FactoryTalk View software.

Arithmetic Status Flags: None

Fault Conditions:

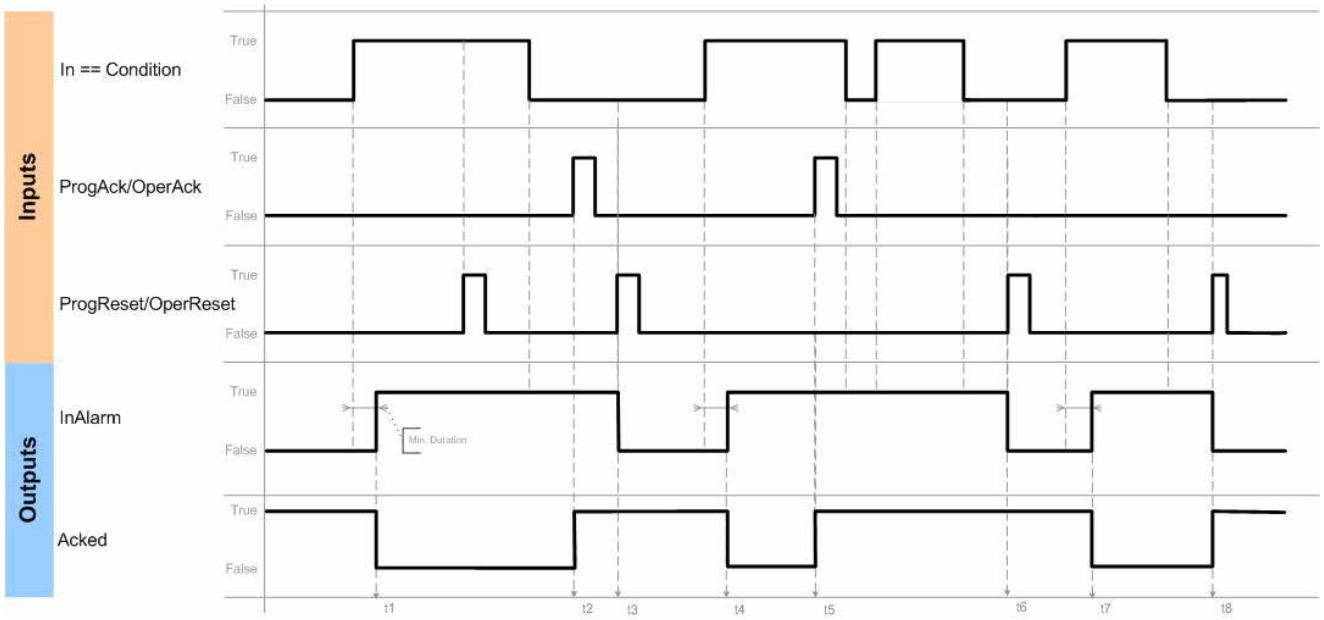
None

Execution:

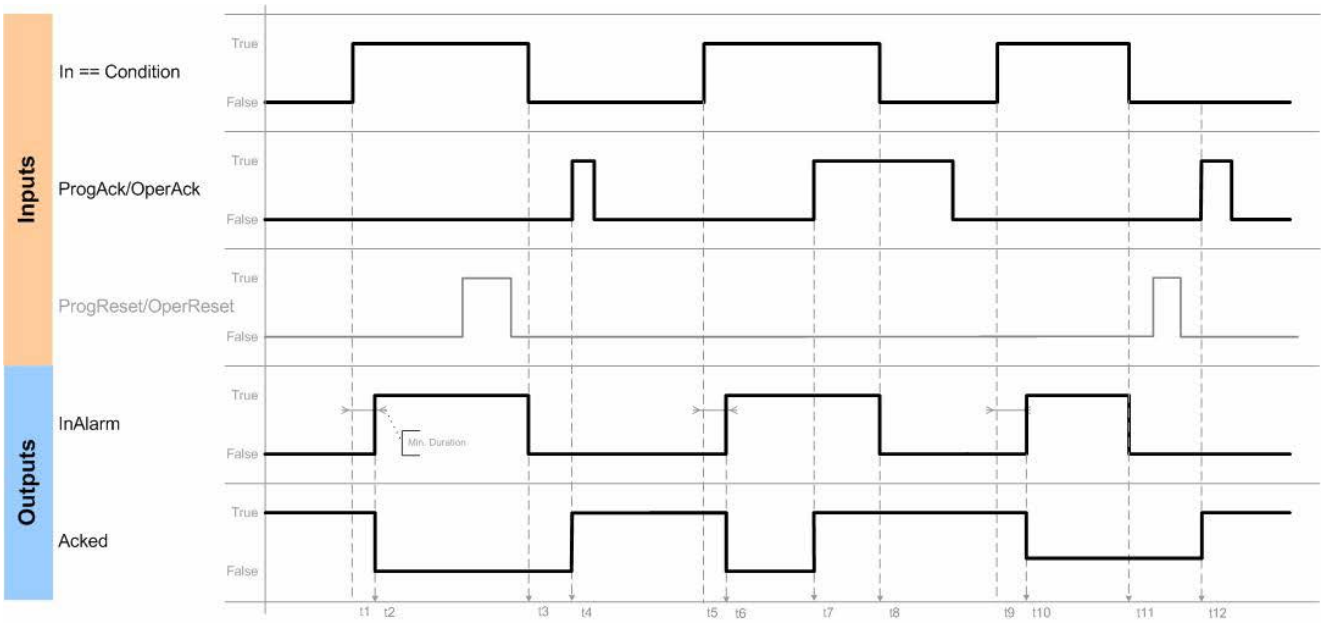
Condition	Relay Ladder Action
Prescan	The rung-condition-out is set to false. InAlarm and Shelved are cleared and Acked is set. All operator requests and timestamps are cleared.
Rung-condition-in is false	The rung-condition-out is set to false. EnableIn and EnableOut are cleared. The In parameter is cleared, and the instruction evaluates to determine the alarm state.
Rung-condition-in is true	The rung-condition-out is set to true. EnableIn and EnableOut are set. The In parameter is set, and the instruction evaluates to determine the alarm state.
Postscan	The rung-condition-out is set to false.

Condition	Function Block Action	Structured Text Action
Prescan	All operator requests and timestamps are cleared. InAlarm, Shelved, and EnableOut are cleared and Acked is set.	All operator requests and timestamps are cleared. InAlarm, Shelved and EnableOut are cleared and Acked is set.
Instruction first scan	No action taken.	No action taken.
Instruction first run	No action taken.	No action taken.
EnableIn is cleared	The instruction does not execute. All operator requests are cleared. EnableOut is cleared.	The instruction executes. EnableOut is cleared.
EnableIn is set	The instruction executes. EnableOut is set.	The instruction executes. EnableOut is always set.
Postscan	EnableOut is cleared.	EnableOut is cleared.

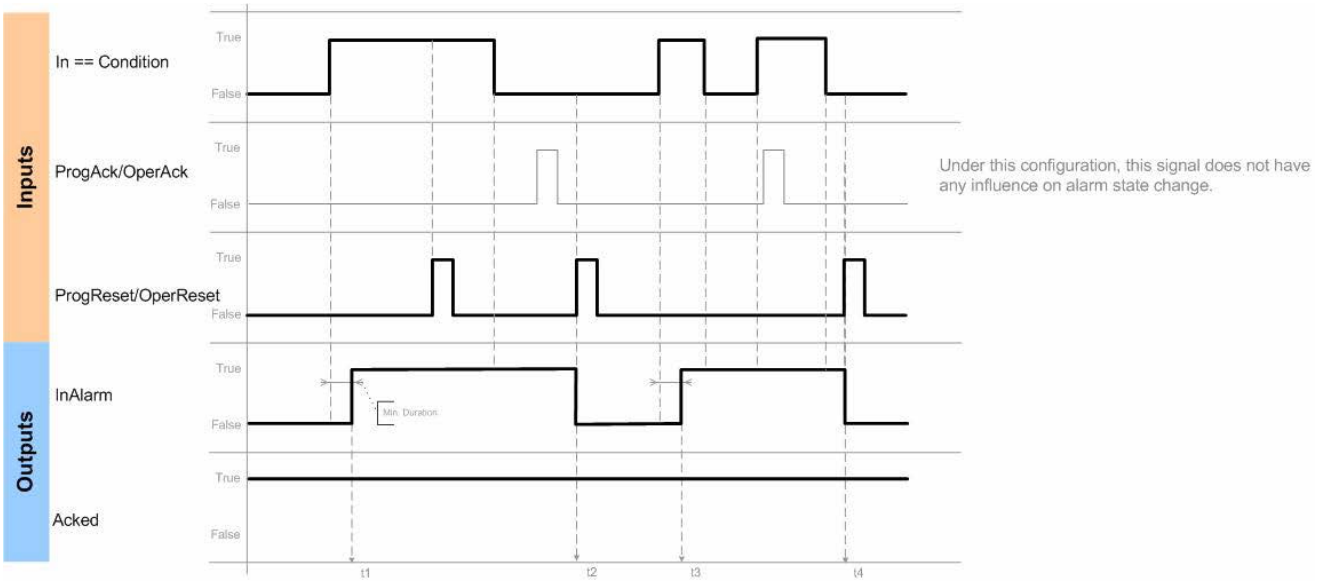
ALMD Alarm Acknowledge Required and Latched



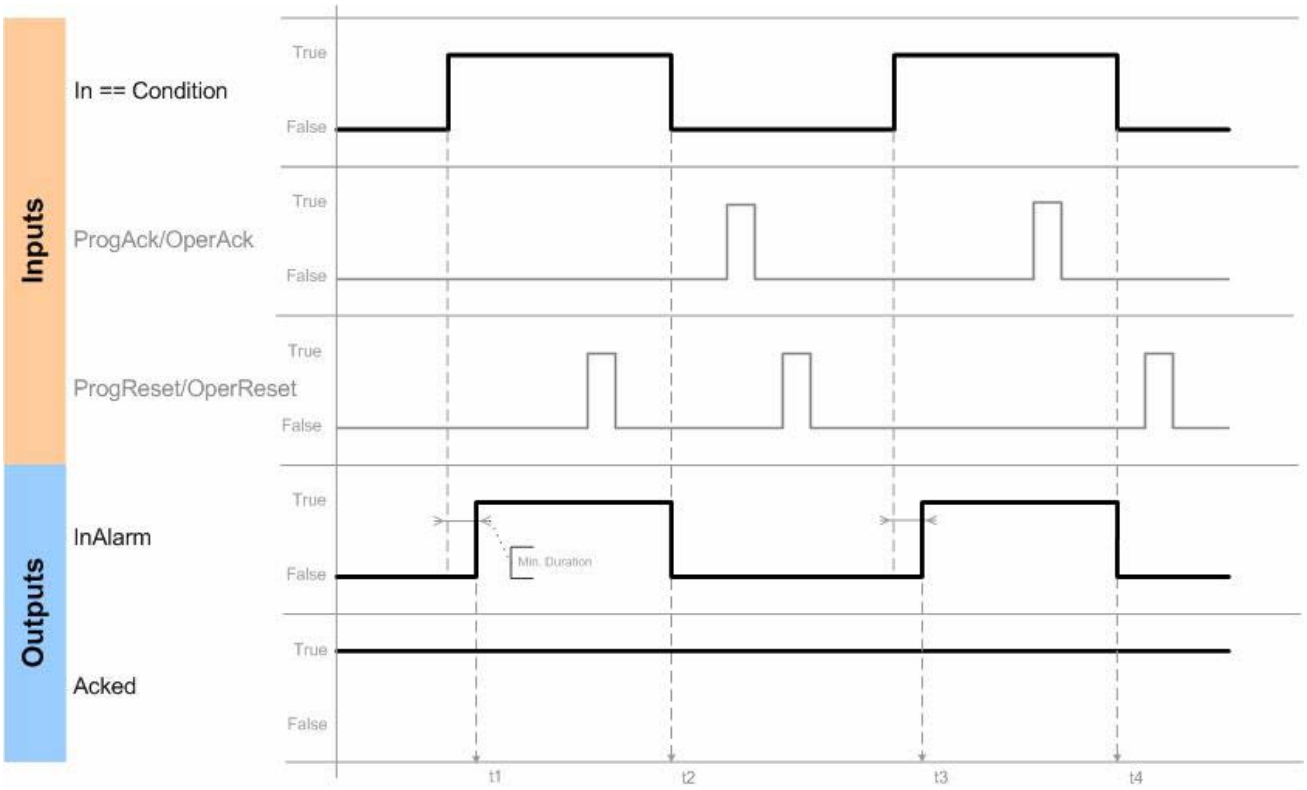
ALMD Alarm Acknowledge Required and Not Latched



ALMD Alarm Acknowledge Not Required and Latched

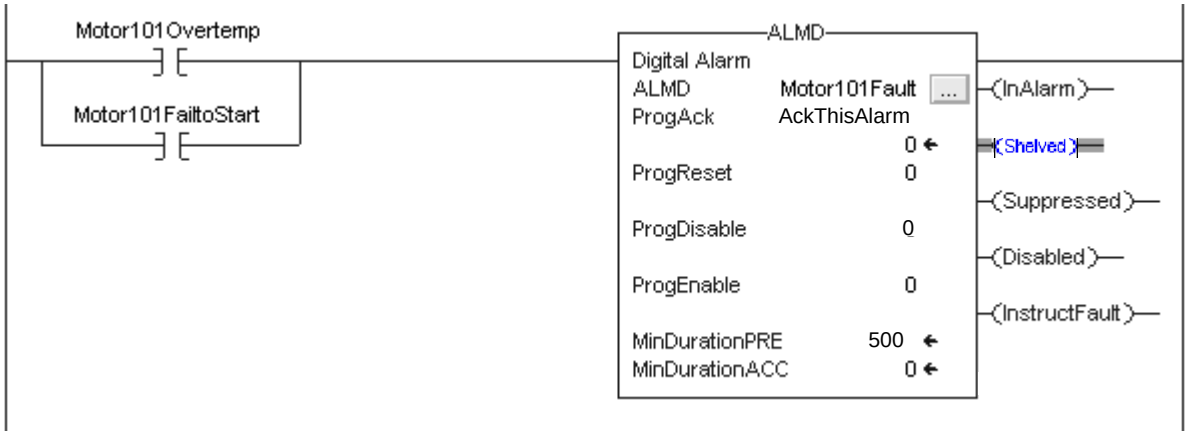


ALMD Alarm Acknowledge Not Required and Not Latched



Example: Two motor failure signals are combined such that if either one occurs, a motor fault alarm is activated. Programmatically acknowledge the alarm with a cleared-to-set transition of the AckThisAlarm tag value. The application logic must clear AckThisAlarm.

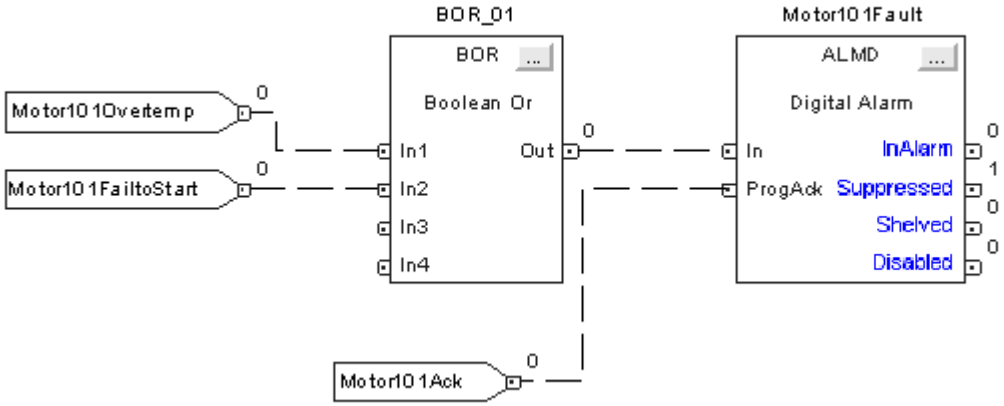
Relay Ladder



Structured Text

```
Motor101FaultConditions := Motor101Overtemp OR
Motor101FailToStart;

ALMD(Motor101Fault,Motor101FaultConditions,
Motor101Ack,0,0,0 );
```

Function Block

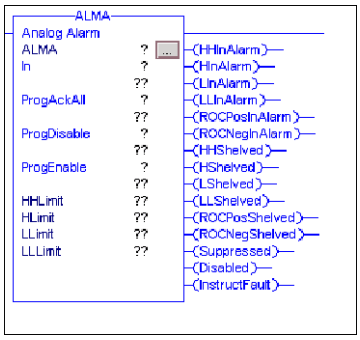


Analog Alarm (ALMA)

The ALMA instruction detects alarms based on the level or rate of change of an analog value. Program (Prog) and operator (Oper) control parameters provide an interface for alarm commands.

Operands:

Relay Ladder



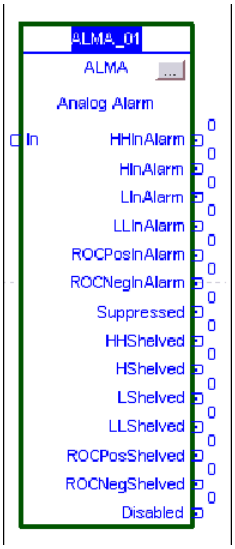
Operand	Type	Format	Description
ALMA tag	ALARM_ANALOG	Structure	ALMA structure.

Structured Text

```
ALMA(ALMA, In, ProgAckAll,
ProgDisable, ProgEnable);
```

The operands are the same as those for the relay ladder ALMD instruction, with a few exceptions as indicated above.

Function Block



Operand	Type	Format	Description
ALMA tag	ALARM_ANALOG	Structure	ALMA structure.

ALARM_ANALOG Structure

Input Parameter	Data Type	Description
EnableIn	BOOL	**Relay Ladder** Corresponds to the rung state. If cleared, the instruction does not execute and outputs are not updated. **Structured Text** If cleared, the instruction does not execute and outputs are not updated. Default is set. **Function Block** If cleared, the instruction does not execute and outputs are not updated. Default is set.
In	REAL	The alarm input value, which is compared with alarm limits to detect alarm conditions. Default = 0.0. **Relay Ladder** Copied from the instruction operand. **Structured Text** Copied from instruction operand.
InFault	BOOL	Bad health indicator for the input. The user application may set InFault to indicate the input signal has an error. When set, the instruction sets InFaulted (Status.1). When cleared, the instruction clears InFaulted (Status.1). In either case, the instruction continues to evaluate In for alarm conditions. Default is cleared (good health).
HHEnabled	BOOL	High High alarm condition detection. Set to enable detection of the High High alarm condition. Clear to disable detection of the High High alarm condition. Default is set.
HEnabled	BOOL	High alarm condition detection. Set to enable detection of the High alarm condition. Clear to disable detection of the High alarm condition. Default is set.
LEnabled	BOOL	Low alarm condition detection. Set to enable detection of the Low alarm condition. Clear to disable detection of the Low alarm condition. Default is set.
LLEnabled	BOOL	Low Low alarm condition detection. Set to enable detection of the Low Low alarm condition. Clear to disable detection of the Low Low alarm condition. Default is set.
AckRequired	BOOL	Specifies whether alarm acknowledgement is required. When set, acknowledgement is required. When cleared, acknowledgement is not required and HHAcked, HAcked, LAcked, LLAcked, ROCPosAcked, and ROCNegAcked are always set. Default is set.
ProgAckAll	BOOL	For Relay Ladder Logic, on transition from cleared to set, acknowledges alarm (if acknowledgement allowed). Default is cleared. **Relay Ladder** Copies value from the instruction operand. **Structured Text** Copies value from the instruction operand.
OperAckAll	BOOL	Set by the operator interface to acknowledge all conditions of this alarm. Takes effect only if an alarm condition is unacknowledged. The alarm instruction clears this parameter. Default is cleared.
HHProgAck	BOOL	High High program acknowledge. Set by the user program to acknowledge a High High condition. Requires a cleared-to-set transition while the alarm condition is unacknowledged. Default is cleared.
HHOperAck	BOOL	High High operator acknowledge. Set by the operator interface to acknowledge a High High condition. Requires a cleared-to-set transition while the alarm condition is unacknowledged. The alarm instruction clears this parameter. Default is cleared.
HProgAck	BOOL	High program acknowledge. Set by the user program to acknowledge a High condition. Requires a cleared-to-set transition while the alarm condition is unacknowledged. Default is cleared.

Input Parameter	Data Type	Description
HOperAck	BOOL	High operator acknowledge. Set by the operator interface to acknowledge a High condition. Requires a cleared-to-set transition while the alarm condition is Unacknowledged. The alarm instruction clears this parameter. Default is cleared.
LProgAck	BOOL	Low program acknowledge. Set by the user program to acknowledge a Low condition. Requires a cleared-to-set transition while the alarm condition is unacknowledged. Default is cleared.
LOperAck	BOOL	Low operator acknowledge. Set by the operator interface to acknowledge a Low condition. Requires a cleared-to-set transition while the alarm condition is unacknowledged. The alarm instruction clears this parameter. Default is cleared.
LLProgAck	BOOL	Low Low program acknowledge. Set by the user program to acknowledge a Low Low condition. Requires a cleared-to-set transition while the alarm condition is unacknowledged. Default is cleared.
LLOperAck	BOOL	Low Low operator acknowledge. Set by the operator interface to acknowledge a Low Low condition. Requires a cleared-to-set transition while the alarm condition is unacknowledged. The alarm instruction clears this parameter. Default is cleared.
ROCPosProgAck	BOOL	Positive rate of change program acknowledge. Set by the user program to acknowledge a positive rate-of-change condition. Requires a cleared-to-set transition while the alarm condition is unacknowledged. Default is cleared.
ROCPosOperAck	BOOL	Positive rate of change operator acknowledge. Set by the operator interface to acknowledge a positive rate-of-change condition. Requires a cleared-to-set transition while the alarm condition is unacknowledged. The alarm instruction clears this parameter. Default is cleared.
ROCNegProgAck	BOOL	Negative rate of change program acknowledge. Set by the user program to acknowledge a negative rate-of-change condition. Requires a cleared-to-set transition while the alarm condition is unacknowledged. Default is cleared.
ROCNegOperAck	BOOL	Negative rate of change operator acknowledge. Set by the operator interface to acknowledge a negative rate-of-change condition. Requires a cleared-to-set transition while the alarm condition is unacknowledged. The alarm instruction clears this parameter. Default is cleared.
ProgSuppress	BOOL	Set by the user program to suppress the alarm. Default is cleared.
OperSuppress	BOOL	Set by the operator interface to suppress the alarm. The alarm instruction clears this parameter. Default is cleared.
ProgUnsuppress	BOOL	Set by the user program to unsuppress the alarm. Takes precedence over Suppress commands. Default is cleared.
OperUnsuppress	BOOL	Set by the operator interface to unsuppress the alarm. Takes precedence over Suppress commands. The alarm instruction clears this parameter. Default is cleared.
HHOperShelve	BOOL	High-high operator shelve. Set by the operator to shelve or reshelve a high-high condition. Takes action on transition from cleared to set. The alarm instruction clears this parameter. Default is cleared. Connecting a push button to the OperShelve tag: The alarm instruction only processes the OperShelve tag on transition from cleared to set to prevent unwanted reshelving of the alarm. For example, if an operator presses a push button to shelve the alarm while the ProgUnshelve tag is set, the alarm is not shelved because the ProgUnshelve tag takes precedence. To shelve the alarm, the operator can release and press the push button again once ProgUnshelve is cleared.
HOperShelve	BOOL	High operator shelve. Set by the operator interface to shelve or reshelve a high condition. Takes action on transition from cleared to set. The alarm instruction clears this parameter. Default is cleared. Unshelve commands take precedence over Shelve commands. Connecting a push button to the OperShelve tag: The alarm instruction only processes the OperShelve tag on transition from cleared to set to prevent unwanted reshelving of the alarm. For example, if an operator presses a push button to shelve the alarm while the ProgUnshelve tag is set, the alarm is not shelved because the ProgUnshelve tag takes precedence. To shelve the alarm, the operator can release and press the push button again once ProgUnshelve is cleared.

Input Parameter	Data Type	Description
LOperShelve	BOOL	Low operator shelve. Set by the operator interface to shelve or reshelve a low condition. Requires a change from a cleared state in one program scan to a set state in the next program scan. The alarm instruction clears this parameter. Default is cleared. Unshelve commands take precedence over Shelve commands. Connecting a push button to the OperShelve tag.
LLOperShelve	BOOL	Low-low operator shelve. Set by the operator interface to shelve or reshelve a low-low condition. Requires a change from a cleared state in one program scan to a set state in the next program scan. The alarm instruction clears this parameter. Default is cleared. Unshelve commands take precedence over Shelve commands. Connecting a push button to the OperShelve tag.
ROCPosOperShelve	BOOL	Positive rate-of-change operator shelve. Set by the operator interface to shelve or reshelve a positive rate-of-change condition. Requires a change from a cleared state in one program scan to a set state in the next program scan. The alarm instruction clears this parameter. Default is cleared. Unshelve commands take precedence over Shelve commands. Connecting a push button to the OperShelve tag.
ROCNegOperShelve	BOOL	Negative rate-of-change operator shelve. Set by the operator interface to shelve or reshelve a negative rate-of-change condition. Requires a change from a cleared state in one program scan to a set state in the next program scan. The alarm instruction clears this parameter. Default is cleared. Unshelve commands take precedence over Shelve commands. Connecting a push button to the OperShelve tag.
ProgUnshelveAll	BOOL	Set by the user program to unshelve all conditions on this alarm. If both shelve and unshelve are set, unshelve commands take precedence over shelve commands. Default is cleared.
HHOperUnshelve	BOOL	High-high operator unshelve. Set by the operator interface to unshelve a high-high condition. The alarm instruction clears this parameter. If both shelve and unshelve are set, unshelve commands take precedence over shelve commands. Default is cleared.
HOperUnshelve	BOOL	High operator unshelve. Set by the operator interface to unshelve a high condition. The alarm instruction clears this parameter. If both shelve and unshelve are set, unshelve commands take precedence over shelve commands. Default is cleared.
LOperUnshelve	BOOL	Low operator unshelve. Set by the operator interface to unshelve a low condition. The alarm instruction clears this parameter. If both shelve and unshelve are set, unshelve commands take precedence over shelve commands. Default is cleared.
LLOperUnshelve	BOOL	Low-low operator unshelve. Set by the operator interface to unshelve a low-low condition. The alarm instruction clears this parameter. If both shelve and unshelve are set, unshelve commands take precedence over shelve commands. Default is cleared.
ROCPosOperUnshelve	BOOL	Positive rate-of-change operator unshelve. Set by the operator interface to unshelve a positive rate-of-change condition. The alarm instruction clears this parameter. If both shelve and unshelve are set, unshelve commands take precedence over shelve commands. Default is cleared.
ROCNegOperUnshelve	BOOL	Negative rate-of-change operator unshelve. Set by the operator interface to unshelve a negative rate-of-change condition. The alarm instruction clears this parameter. If both shelve and unshelve are set, unshelve commands take precedence over shelve commands. Default is cleared.
ProgDisable	BOOL	Set by the user program to disable the alarm. On transition from cleared to set, disables the alarm. Default is cleared. **Relay Ladder** Copied from the instruction operand. **Structured Text** Copied from the instruction operand.
OperDisable	BOOL	Set by the operator interface to disable the alarm. The alarm instruction clears this parameter. Default is cleared.
ProgEnable	BOOL	Set by the user program to enable the alarm. Takes precedence over a Disable command. Takes action on transition from cleared to set. Default is cleared. **Relay Ladder** Copied from the instruction operand. **Structured Text** Copied from the instruction operand.
OperEnable	BOOL	Set by the operator interface to enable the alarm. Takes precedence over Disable command. The alarm instruction clears this parameter. Default is cleared.

Input Parameter	Data Type	Description
AlarmCountReset	BOOL	Set by the user program to reset the alarm counts for all conditions. Default is cleared.
HHMinDurationEnable	BOOL	High-high minimum duration enable. Set to enable minimum duration timer when detecting the high-high condition. Default is enabled.
HMinDurationEnable	BOOL	High minimum duration enable. Set to enable minimum duration timer when detecting the high condition. Default is enabled.
LMinDurationEnable	BOOL	Low minimum duration enable. Set to enable minimum duration timer when detecting the low condition. Default is enabled.
LLMinDurationEnable	BOOL	Low-low minimum duration enable. Set to enable minimum duration timer when detecting the low-low condition. Default is enabled.
HHLimit	REAL	High High alarm limit. Valid = HLimit < HHLimit < maximum positive float. Default = 0.0.
HHSeverity	DINT	Severity of the High High alarm condition. This does not affect processing of alarms by the controller, but can be used for sorting and filtering functions at the alarm subscriber. Valid = 1...1000 (1000 = most severe; 1 = least severe). Default = 500.
HLimit	REAL	High alarm limit. Valid = LLimit < HLimit < HHLimit. Default = 0.0.
HSeverity	DINT	Severity of the High alarm condition. This does not affect processing of alarms by the controller, but can be used for sorting and filtering functions at the alarm subscriber. Valid = 1...1000 (1000 = most severe; 1 = least severe). Default = 500.
LLimit	REAL	Low alarm limit. Valid = LLLimit < LLimit < HLimit. Default = 0.0.
LSeverity	DINT	Severity of the Low alarm condition. This does not affect processing of alarms by the controller, but can be used for sorting and filtering functions at the alarm subscriber. Valid = 1...1000 (1000 = most severe; 1 = least severe). Default = 500.
LLLimit	REAL	Low Low alarm limit. Valid = maximum negative float < LLLimit < LLimit. Default = 0.0
LLSeverity	DINT	Severity of the Low Low alarm condition. This does not affect processing of alarms by the controller, but can be used for sorting and filtering functions at the alarm subscriber. Valid = 1...1000 (1000 = most severe; 1 = least severe). Default = 500.
MinDurationPRE	DINT	Minimum duration preset (milliseconds) for an alarm level condition to remain true before the condition is marked as InAlarm and alarm notification is sent to clients. The controller collects alarm data as soon as the alarm condition is detected, so no data is lost while waiting to meet the minimum duration. Does not apply to rate-of-change conditions. MinDurationPRE applies only to the first excursion from normal in either direction. For example, once the High condition times out, the High High condition will become active immediately, while a low condition will wait for the time-out period. Valid = 0...2,147,483,647. Default = 0.
ShelveDuration	DINT	Time duration (in minutes) for which a shelved alarm will be shelved. Minimum time is one minute. Maximum time is defined by MaxShelveDuration.
MaxShelveDuration	DINT	Maximum time duration (in minutes) for which an alarm can be shelved.

Input Parameter	Data Type	Description
Deadband	REAL	Deadband for detecting that High High, High, Low, and Low Low alarm levels have returned to normal. A non-zero Deadband can reduce alarm condition chattering if the In value is continually changing but remaining near the level condition threshold. The Deadband value does not affect the transition to the InAlarm (active) state. Once a level condition is active, but before the condition will return to the inactive (normal) state, the In value must either: • drop below the threshold minus the deadband (for High and High High conditions). OR • rise above the threshold plus the deadband (for Low and Low Low conditions). The Deadband is not used to condition the Minimum Duration time measurement. Valid = 0 ≤ Deadband < Span from first enabled low alarm to the first enabled high alarm. Default = 0.0.
ROCPosLimit	REAL	Limit for an increasing rate-of-change in units per second. Detection is enabled for any value > 0.0 if ROCPeriod is also > 0.0. Valid = 0.0...maximum possible float. Default = 0.0.
ROCPosSeverity	DINT	Severity of the increasing rate-of-change condition. This does not affect processing of alarms by the controller, but can be used for sorting and filtering functions at the alarm subscriber. Valid = 1...1000 (1000 = most severe; 1 = least severe). Default = 500.
ROCNegLimit	REAL	Limit for a decreasing rate-of-change in units per second. Detection is enabled for any value > 0.0 if ROCPeriod is also > 0.0. Valid = 0.0...maximum possible float. Default = 0.0.
ROCNegSeverity	DINT	Severity of the decreasing rate-of-change condition. This does not affect processing of alarms by the controller, but can be used for sorting and filtering functions at the alarm subscriber. Valid = 1...1000 (1000 = most severe; 1 = least severe). Default = 500.
ROCPeriod	REAL	Time period in seconds for calculation (sampling interval) of the rate of change value. Each time the sampling interval expires, a new sample of In is stored, and ROC is recalculated. Rate-of-change detection is enabled for any value > 0.0. Valid = 0.0...maximum possible float. Default = 0.0.

Output Parameter	Data Type	Description
EnableOut	BOOL	Enable output.
InAlarm	BOOL	Alarm active status. Set when any alarm condition is active. Cleared when all alarm conditions are not active (normal status).
AnyInAlarmUnack	BOOL	Combined alarm active and acknowledged status. Set when any alarm condition is detected and unacknowledged. Cleared when all alarm conditions are normal (inactive), acknowledged, or both.
HHInAlarm	BOOL	High High alarm condition status. Set when a High High condition exists. Cleared when no High High condition exists.
HInAlarm	BOOL	High alarm condition status. Set when a High condition exists. Cleared when no High condition exists.
LInAlarm	BOOL	Low alarm condition status. Set when a Low condition exists. Cleared when no Low condition exists.
LLInAlarm	BOOL	Low Low alarm condition status. Set when a Low Low condition exists. Cleared when no Low Low condition exists.
ROCPosInAlarm	BOOL	Positive rate-of-change alarm condition status. Set when a positive rate-of-change condition exists. Cleared when no positive rate-of-change condition exists.
ROCNegInAlarm	BOOL	Negative rate-of-change alarm condition status. Set when a negative rate-of-change condition exists. Cleared when no negative rate-of-change condition exists.
ROC	REAL	Calculated rate-of-change of the In value. This value is updated when the instruction is scanned following each elapsed ROCPeriod. The ROC value is used to evaluate the ROCPosInAlarm and ROCNegInAlarm conditions. ROC = (current sample of In – previous sample of In) / ROCPeriod.
HHAcked	BOOL	High High condition acknowledged status. Set when a High High condition is acknowledged. Always set when AckRequired is cleared. Cleared when a High High condition is not acknowledged.
HAcked	BOOL	High condition acknowledged status. Set when a High condition is acknowledged. Always set when AckRequired is cleared. Cleared when a High condition is not acknowledged.

Output Parameter	Data Type	Description
LAcked	BOOL	Low condition acknowledged status. Set when a Low condition is acknowledged. Always set when AckRequired is cleared. Cleared when a Low condition is not acknowledged.
LLAcked	BOOL	Low Low condition acknowledged status. Set when a Low Low condition is acknowledged. Always set when AckRequired is cleared. Cleared when a Low Low condition is not acknowledged.
ROCPosAcked	BOOL	Positive rate-of-change condition acknowledged status. Set when a positive rate-of-change condition is acknowledged. Always set when AckRequired is cleared. Cleared when a positive rate-of-change condition is not acknowledged.
ROCNegAcked	BOOL	Negative rate-of-change condition acknowledged status. Set when a negative rate-of-change condition is acknowledged. Always set when AckRequired is cleared. Cleared when a negative rate-of-change condition is not acknowledged.
HHInAlarmUnack	BOOL	Combined High High condition active and unacknowledged status. Set when the High High condition is active (HHInAlarm is set) and unacknowledged. Cleared when the High High condition is normal (inactive), acknowledged, or both.
HInAlarmUnack	BOOL	Combined High condition active and unacknowledged status. Set when the High condition is active (HInAlarm is set) and unacknowledged. Cleared when the High condition is normal (inactive), acknowledged, or both.
LInAlarmUnack	BOOL	Combined Low condition active and unacknowledged status. Set when the Low condition is active (LInAlarm is set) and unacknowledged. Cleared when the Low condition is normal (inactive), acknowledged, or both.
LLInAlarmUnack	BOOL	Combined Low Low condition active and unacknowledged status. Set when the Low Low condition is active (LLInAlarm is set) and unacknowledged. Cleared when the Low Low condition is normal (inactive), acknowledged, or both.
ROCPosInAlarmUnack	BOOL	Combined positive rate-of-change condition active and unacknowledged status. Set when the positive rate-of-change condition is active (ROCPosInAlarm is set) and unacknowledged. Cleared when the positive rate-of-change condition is normal (inactive), acknowledged, or both.
ROCNegInAlarmUnack	BOOL	Combined negative rate-of-change condition active and unacknowledged status. Set when the negative rate-of-change condition is active (ROCNegInAlarm is set) and unacknowledged. Cleared when the negative rate-of-change condition is normal (inactive), acknowledged, or both.
Suppressed	BOOL	Suppressed status of the alarm. Set when the alarm is suppressed. Cleared when the alarm is not suppressed.
HHShelved	BOOL	High-high condition shelved status. Set when a high-high condition is shelved. Cleared when high-high condition is unshelved.
HShelved	BOOL	High condition shelved status. Set when a high condition is shelved. Cleared when high condition is unshelved.
LShelved	BOOL	Low condition shelved status. Set when a low condition is shelved. Cleared when low condition is unshelved.
LLShelved	BOOL	Low-low condition shelved status. Set when a low-low condition is shelved. Cleared when low-low condition is unshelved.
ROCPosShelved	BOOL	Positive rate-of-change condition shelved status. Set when a positive rate-of-change condition is shelved. Cleared when positive rate-of-change condition is unshelved.
ROCNegShelved	BOOL	Negative rate-of-change condition shelved status. Set when a negative rate-of-change condition is shelved. Cleared when negative rate-of-change condition is unshelved.
Commissioned	BOOL	Reserved for future use.
Disabled	BOOL	Disabled status of the alarm. Set when the alarm is disabled. Cleared when the alarm is enabled.

Output Parameter	Data Type	Description
MinDurationACC	DINT	Elapsed time since an alarm condition was detected. When this value reaches MinDurationPRE, the alarm becomes active (InAlarm is set), and alarm notification is sent to clients. Does not apply to rate of change conditions or to conditions for which the minimum duration detection is disabled. If the alarm condition becomes inactive before MinDurationACC reaches MinDurationPRE, then the MinDuration timer is stopped. MinDurationACC is not reset so it keeps its last value. MinDurationACC is reset when an alarm level condition is newly detected and thus the MinDuration timer is started again. The MinDuration timer can be stopped before expiration if EnableIn is cleared or the alarm is disabled. In all of these cases, MinDurationACC keeps its last value and is not reset until the MinDuration timer is started again.
HHInAlarmTime	LINT	Timestamp when the ALMA instruction detected that the In value exceeded the High High condition limit for the most recent transition to the active state.
HHAlarmCount	DINT	The number of times the High High condition has been activated. If the maximum value is reached, the counter leaves the value at the maximum count value.
HInAlarmTime	LINT	Timestamp when the ALMA instruction detected that the In value exceeded the High condition limit for the most recent transition to the active state.
HAlarmCount	DINT	The number of times the High condition has been activated. If the maximum value is reached, the counter leaves the value at the maximum count value.
LInAlarmTime	LINT	Timestamp when the ALMA instruction detected that the In value exceeded the Low condition limit for the most recent transition to the active state.
LAlarmCount	DINT	The number of times the Low condition has been activated. If the maximum value is reached, the counter leaves the value at the maximum count value.
LLInAlarmTime	LINT	Timestamp when the ALMA instruction detected that the In value exceeded the Low Low condition limit for the most recent transition to the active state.
LLAlarmCount	DINT	The number of times the Low Low condition has been activated. If the maximum value is reached, the counter leaves the value at the maximum count value.
ROCPosInAlarmTime	LINT	Timestamp when the ALMA instruction detected that the In value exceeded the positive rate-of-change condition limit for the most recent transition to the active state.
ROCPosInAlarmCount	DINT	The number of times the positive rate-of-change condition has been activated. If the maximum value is reached, the counter leaves the value at the maximum count value.
ROCNegInAlarmTime	LINT	Timestamp when the ALMA instruction detected that the In value exceeded the negative rate-of-change condition limit for the most recent transition to the active state.
ROCNegAlarmCount	DINT	The number of times the negative rate-of-change condition has been activated. If the maximum value is reached, the counter leaves the value at the maximum count value.
AckTime	LINT	Timestamp of most recent condition acknowledgement. If the alarm does not require acknowledgement, this timestamp is equal to most recent condition alarm time.
RetToNormalTime	LINT	Timestamp of alarm returning to a normal state.
AlarmCountResetTime	LINT	Timestamp indicating when the alarm count was reset.
ShelveTime	LINT	Timestamp indicates when an alarm condition was shelved the last time. Each time alarm condition is shelved the timestamp is set to current time.
UnshelveTime	LINT	Timestamp indicating when all alarm conditions are going to be unshelved. Value is set only when no alarm condition is shelved yet or when an alarm condition is reshelved. The timestamp is determined as sum of the ShelveDuration time period and current time. If an alarm condition is unshelved programmatically or by an operator and no other alarm condition is shelved, then value is set to the current time.
Status	DINT	Combined status indicators: Status.0 = InstructFault. Status.1 = InFaulted. Status.2 = SeverityInv. Status.3 = AlarmLimitsInv. Status.4 = DeadbandInv. Status.5 = ROCPosLimitInv. Status.6 = ROCNegLimitInv. Status.7 = ROCPeriodInv.
InstructFault (Status.0)	BOOL	Instruction error conditions exist. This is not a minor or major controller error. Check the remaining status bits to determine what occurred.

Output Parameter	Data Type	Description
InFaulted (Status.1)	BOOL	User program has set InFault to indicate bad quality input data. Alarm continues to evaluate In for alarm conditions.
SeverityInv (Status.2)	BOOL	Alarm severity configuration is invalid. If severity <1, the instruction uses Severity = 1. If severity >1000, the instruction uses Severity = 1000.
AlarmLimitsInv (Status.3)	BOOL	Alarm Limit configuration is invalid (for example, LLimit < LLLimit). If invalid, the instruction clears all level conditions active bits. Until the fault is cleared, no new level conditions can be detected.
DeadbandInv (Status.4)	BOOL	Deadband configuration is invalid. If invalid, the instruction uses Deadband = 0.0. Valid = 0 ≤ Deadband < Span from first enabled low alarm to the first enabled high alarm.
ROCPosLimitInv (Status.5)	BOOL	Positive rate-of-change limit invalid. If invalid, the instruction uses ROCPosLimit = 0.0, which disables positive rate-of-change detection.
ROCNegLimitInv (Status.6)	BOOL	Negative rate-of-change limit invalid. If invalid, the instruction uses ROCNegLimit = 0.0, which disables negative rate-of-change detection.
ROCPeriodInv (Status.7)	BOOL	Rate-of-change period invalid. If invalid, the instruction uses ROCPeriod = 0.0, which disables rate-of-change detection.

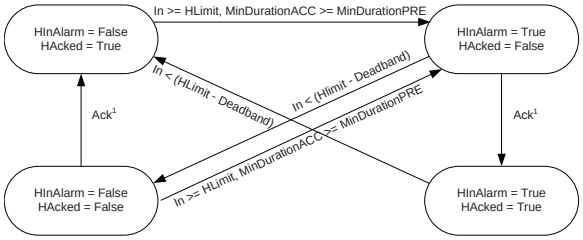
Description

The ALMA instruction detects alarms based on the level or rate of change of a value.

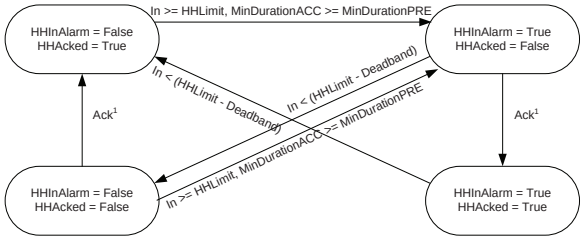
The ALMA instruction provides additional functionality when used with FactoryTalk View SE software. You can display alarms in the Alarm Summary, Alarm Banner, Alarm Status Explorer, and Alarm Log Viewer displays in FactoryTalk View SE software.

FactoryTalk Alarms and Events server subscribes to alarms in the controller. Use the output parameters to monitor the instruction to see the alarm subscription status and to display alarm status changes. If a connection to the FactoryTalk Alarms and Events server is lost, the controller can briefly buffer alarm data until the connection is restored.

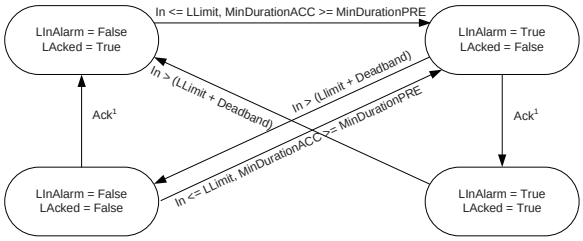
State Diagrams When Acknowledgement Required



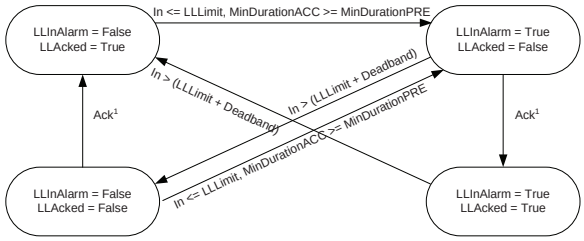
[1] H alarm condition can be acked by several different ways: HProgAck, HOperAck, ProgAckAll, OperAckAll, clients (Studio 5000 software, FactoryTalk View software).



[1] HH alarm condition can be acked by several different ways: HHProgAck, HHOperAck, ProgAckAll, OperAckAll, clients (Studio 5000 software, FactoryTalk View software).



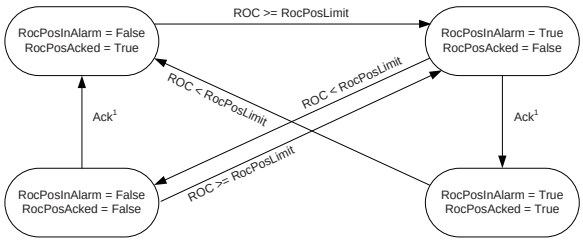
[1] L alarm condition can be acked by several different ways: LProgAck, LOperAck, ProgAckAll, OperAckAll, clients (Studio 5000 software, FactoryTalk View software).



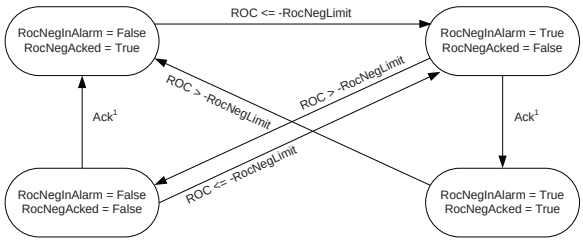
[1] LL alarm condition can be acked by several different ways: LLProgAck, LLOperAck, ProgAckAll, OperAckAll, clients (Studio 5000 software, FactoryTalk View software).

$$ROC = \frac{\ln(CurrentSample) - \ln(PreviousSample)}{ROCPeriod}$$

Where a new sample is collected on the next scan after the ROCPeriod has elapsed.

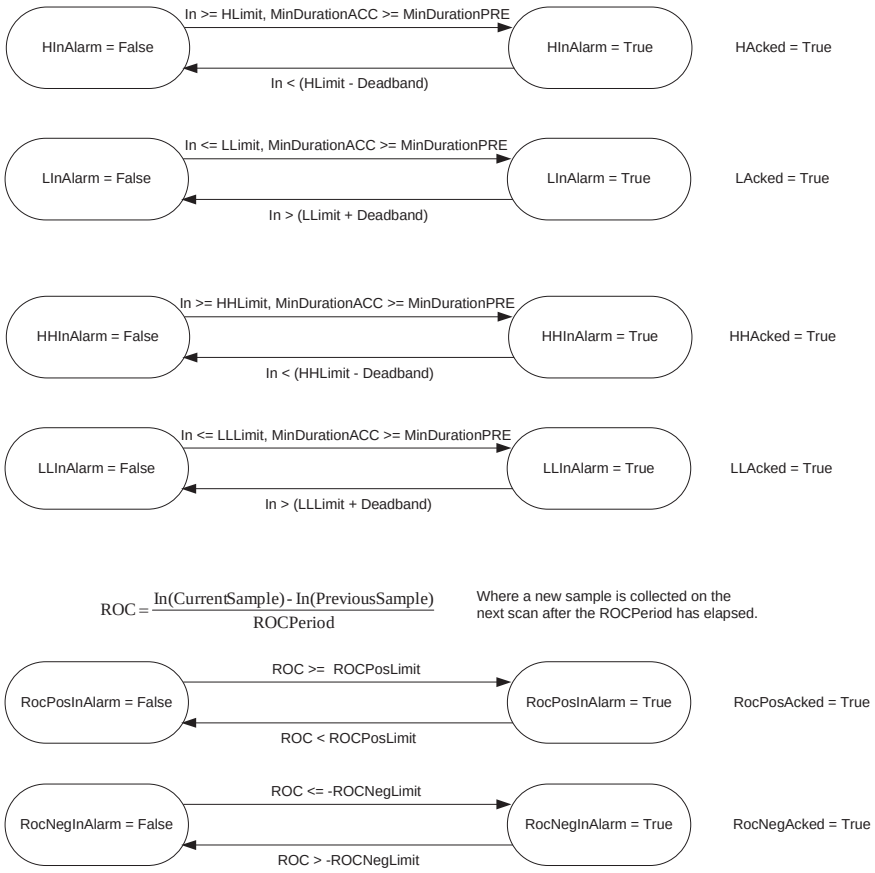


[1] ROCPos alarm condition can be acked by several different ways: RocPosProgAck, RocPosOperAck, ProgAckAll, OperAckAll, clients (Studio 5000 software, FactoryTalk View software).



[1] ROCNeg alarm condition can be acked by several different ways: RocNegProgAck, RocNegOperAck, ProgAckAll, OperAckAll, clients (Studio 5000 software, FactoryTalk View software).

State Diagrams When Acknowledgement Not Required



Arithmetic Status Flags: Arithmetic status flags are set for the ROC output.

Fault Conditions:

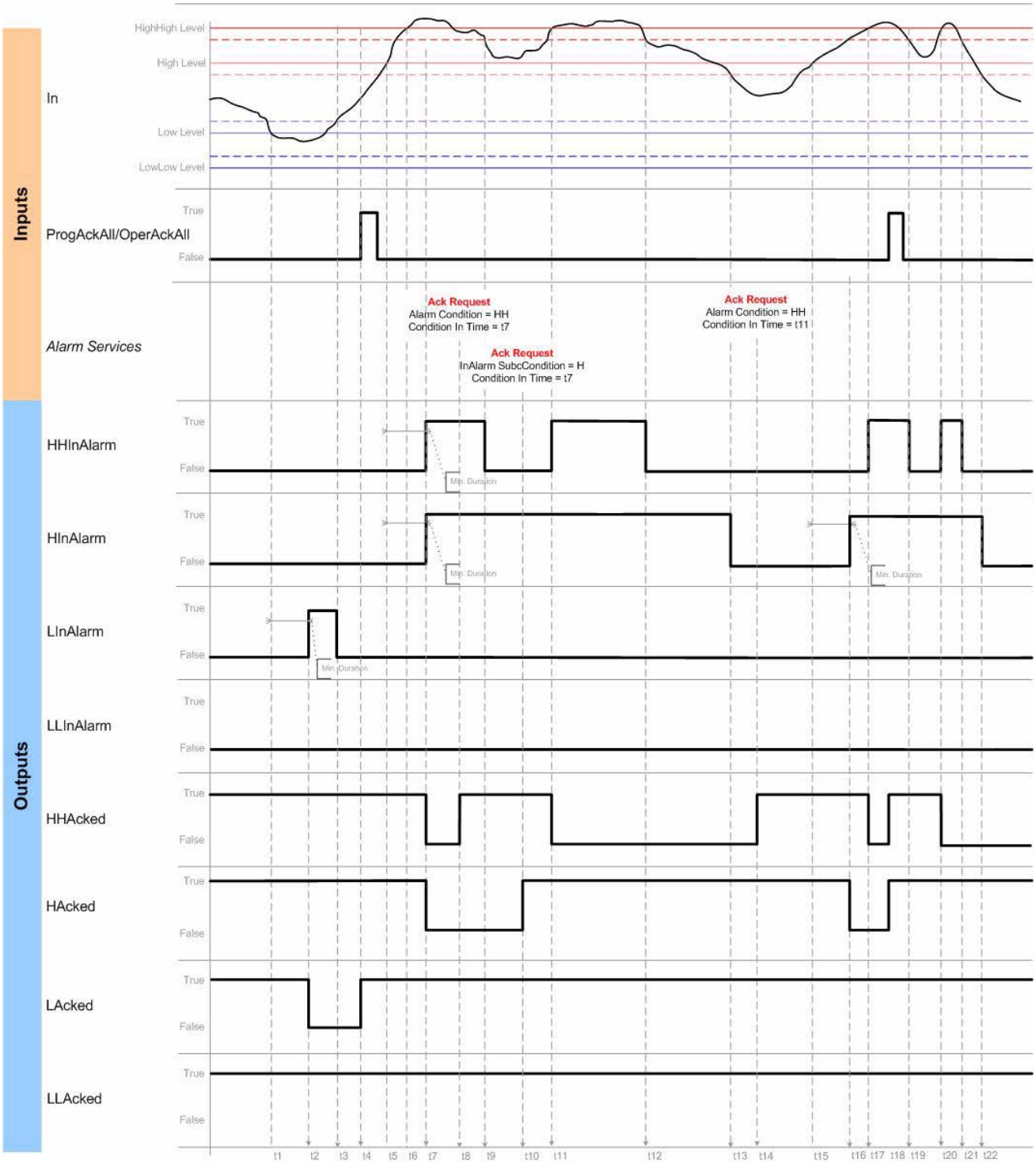
Minor Fault	Fault Type	Fault Code
ROC overflow	4	4

Execution:

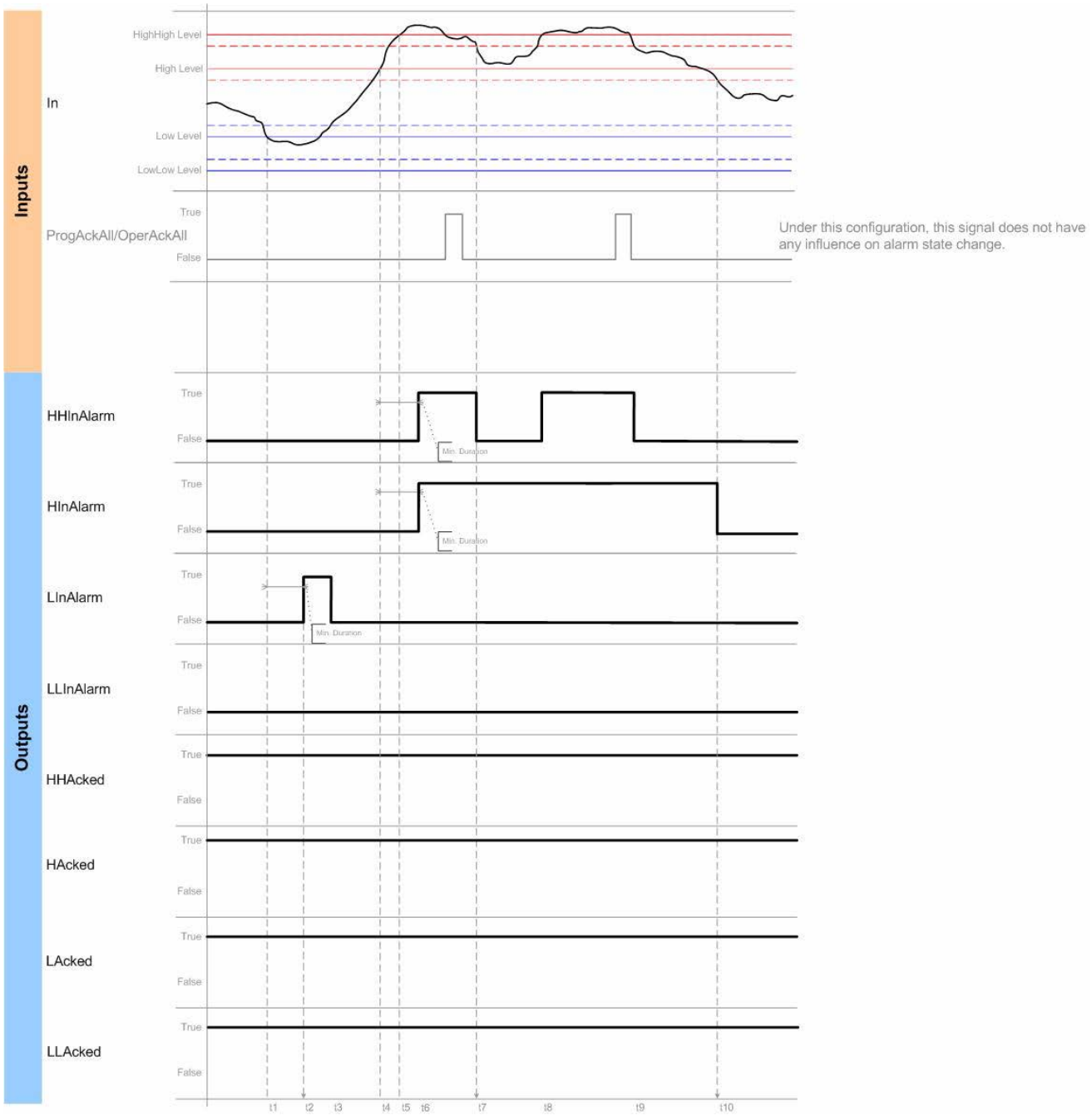
Condition	Relay Ladder Action
Prescan	The rung-condition-out is set to false. All of the *x*InAlarm parameters and xShelved are cleared and all alarm conditions are acknowledged. All operator requests and timestamps are cleared.
Rung-condition-in is false	The instruction does not execute. EnableOut is cleared. The rung-condition-out is set to false.
Rung-condition-in is true	The instruction executes. EnableOut is set. The rung-condition-out is set to true.
Postscan	The rung-condition-out is set to false.

Condition	Function Block Action	Structured Text Action
Prescan	All operator requests and timestamps are cleared. All of the *x*InAlarm and xShelved parameters are cleared. EnableOut is cleared. All alarm conditions are acknowledged.	All operator requests and timestamps are cleared. All of the *x*InAlarm and xShelved parameters are cleared. EnableOut is cleared. All alarm conditions are acknowledged.
Instruction first scan	No action taken.	No action taken.
Instruction first run	No action taken.	No action taken.
EnableIn is false	The instruction does not execute. All operator requests are cleared. EnableOut is cleared.	The instruction does not execute. EnableOut is cleared.
EnableIn is true	The instruction executes. EnableOut is set.	The instruction executes. EnableOut is always set to true.
Postscan	EnableOut is cleared.	EnableOut is cleared to false.

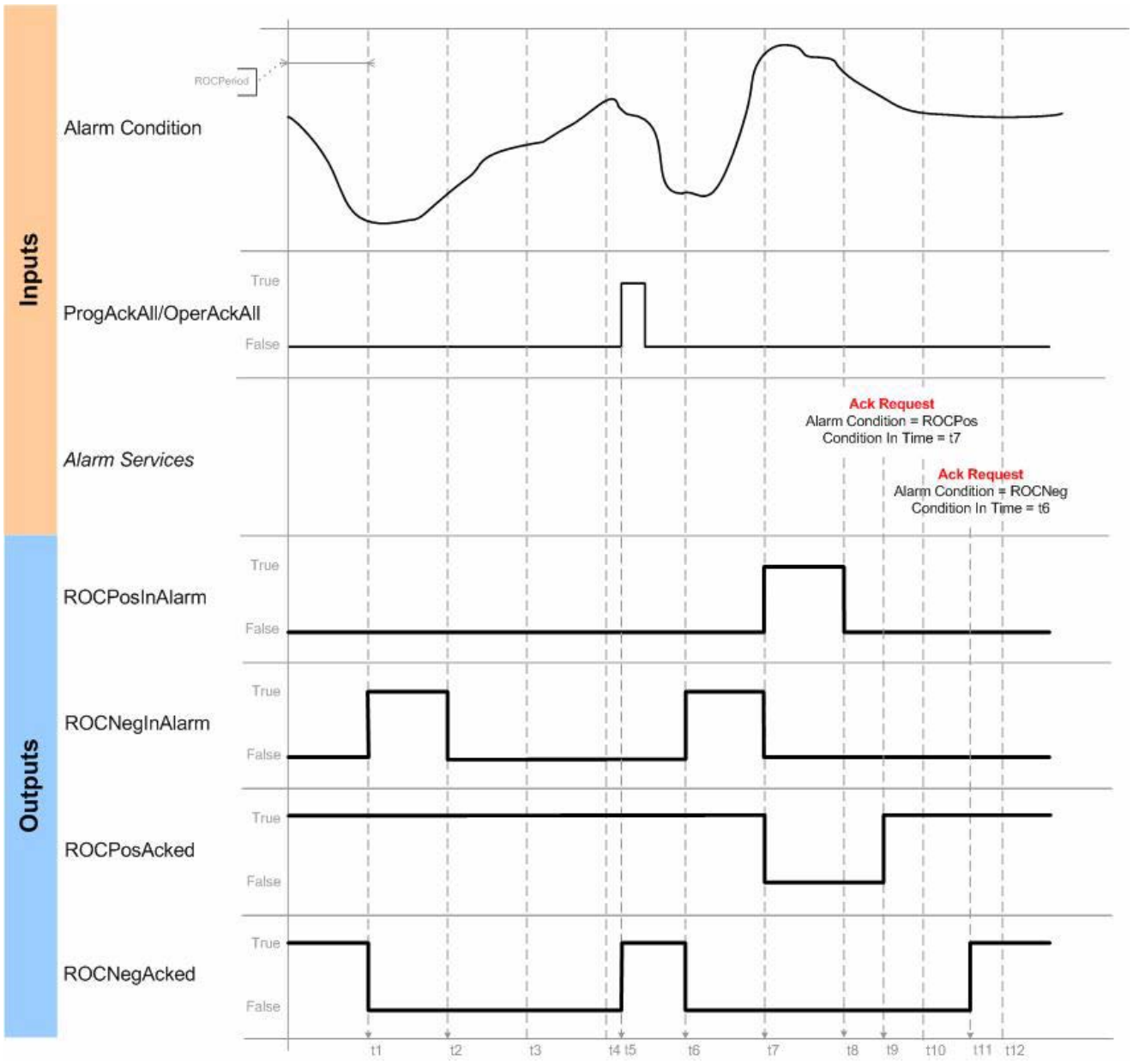
ALMA Level Condition Acknowledge Required



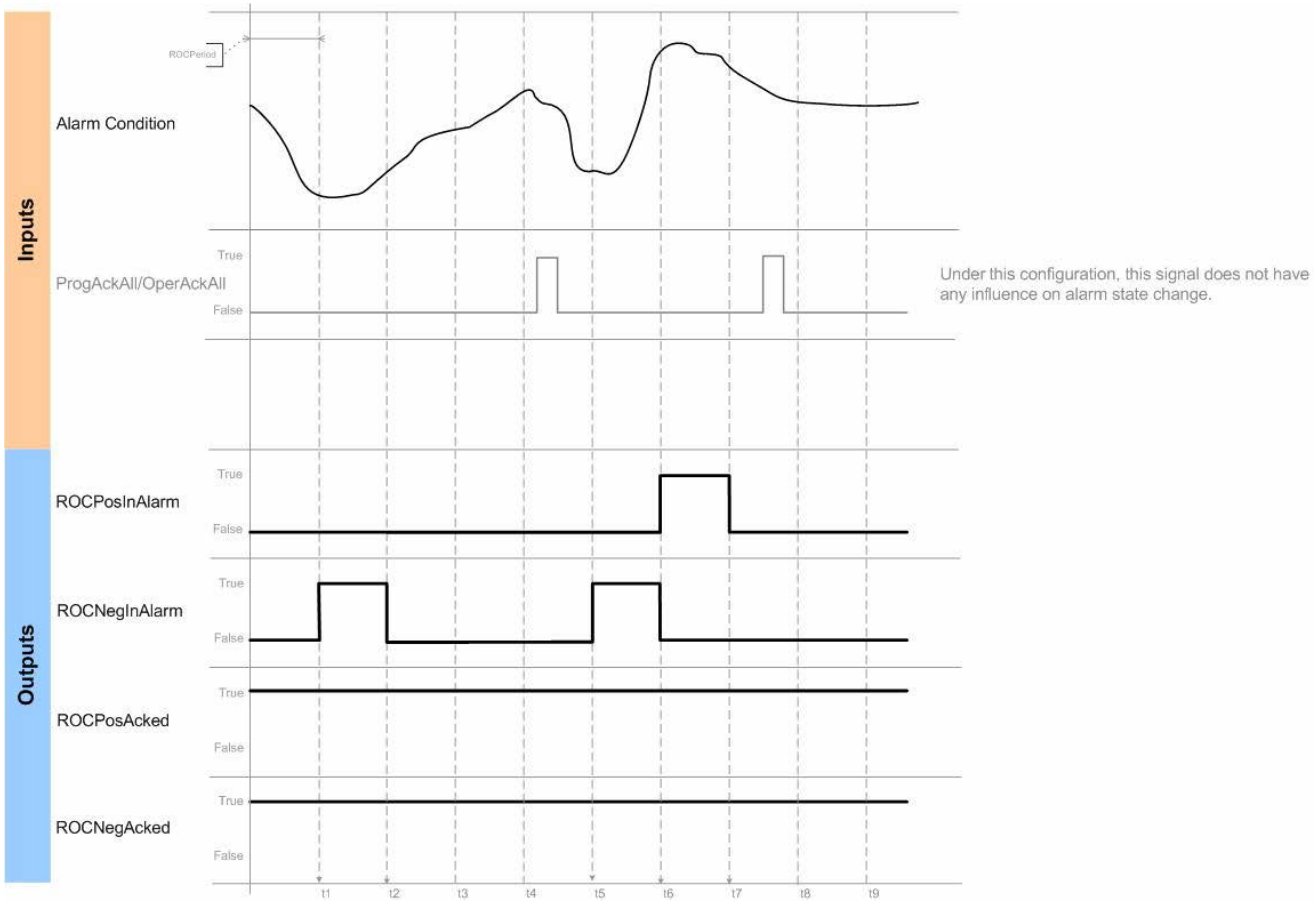
ALMA Level Condition Acknowledge Not Required



ALMA Rate of Change Acknowledge Required

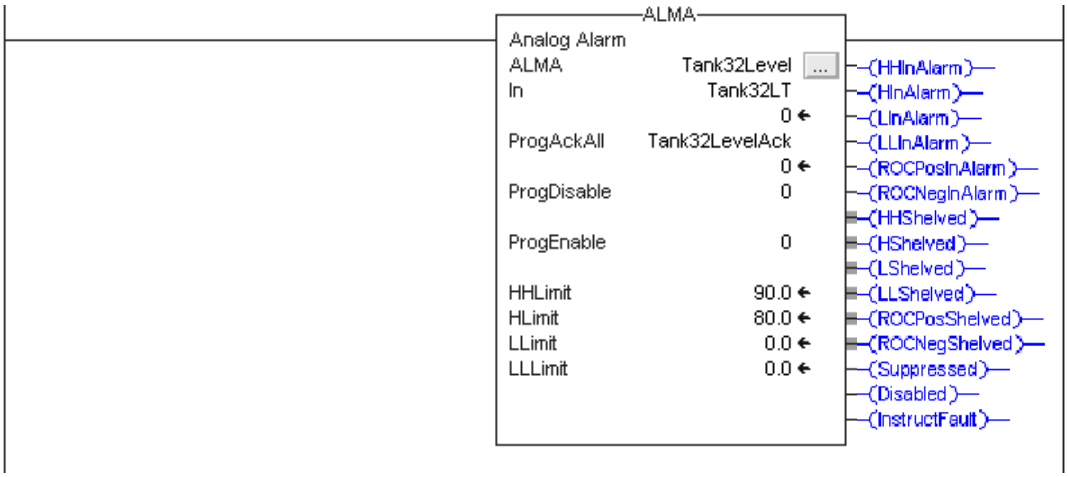


ALMA Rate of Change Acknowledge Not Required



Example: A tank alarm is activated if the tank level surpasses a High or High limit. Programmatically acknowledge all the alarm conditions with a cleared-to-set transition of the Tank32LevelAck tag value. The application logic must clear Tank32LevelAck.

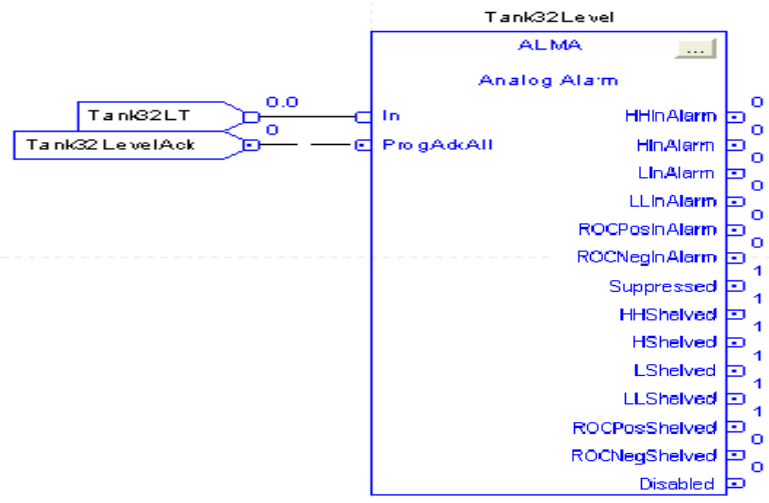
Relay Ladder



Structured Text

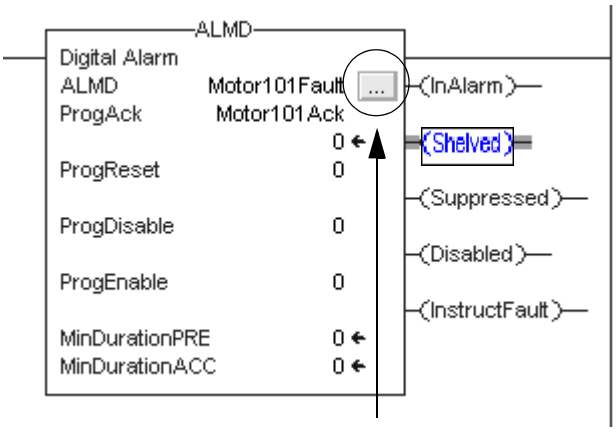
```
ALMA(Tank32Level,Tank32LT,Tank32LevelAck,0, 0);
```

Function Block

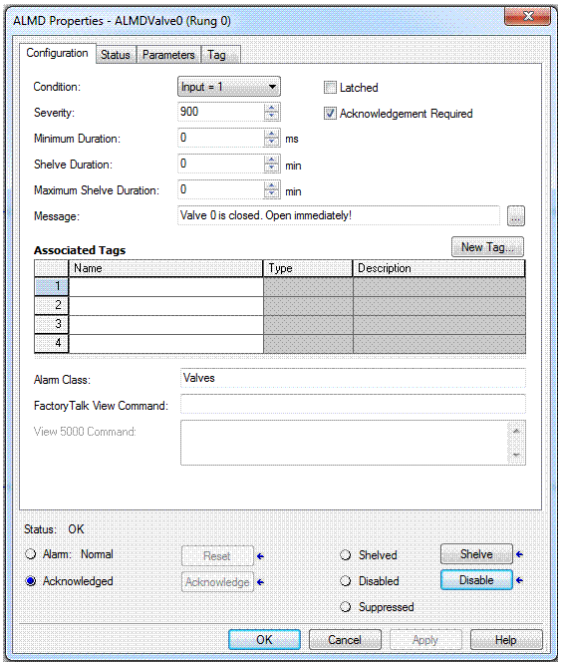


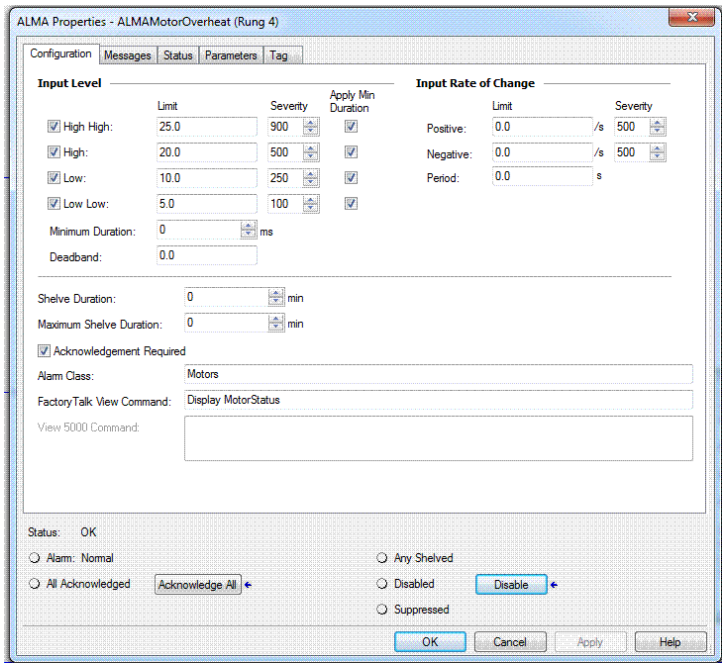
Configure an Alarm Instruction

After you enter an ALMD or ALMA instruction and specify the alarm tag name, use the Alarm Configuration dialog box to specify the details of the message.



The Properties dialog box for the alarm instruction includes a Configuration tab.





For each alarm instruction, configure this information.

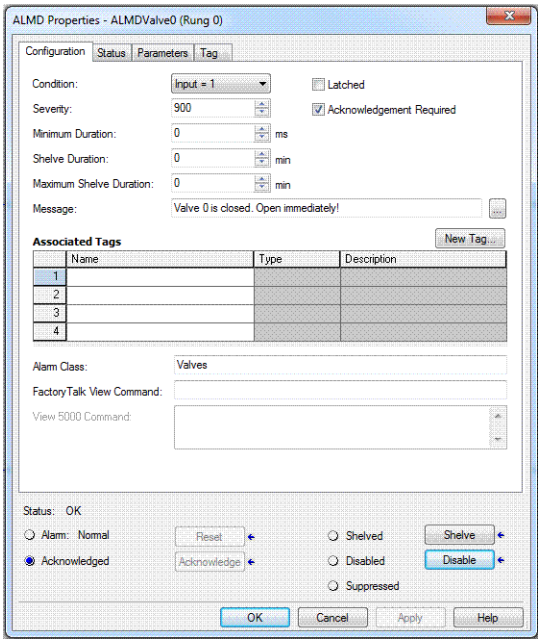
Option	Description
Condition - ALMD instruction	Condition to trigger the alarm. Select Input=1 for an active alarm when In=1. Select Input=0 for an active alarm when In=0.
Input Level - ALMA instruction Input Rate of Change - ALMA instruction	Input Level (High High, High, Low, or Low Low) or Input Rate of Change (Positive or Negative) to trigger an alarm. Select the alarm conditions and enter the limits for those conditions. Disable rate-of-change conditions by entering a 0 for the period or limit.

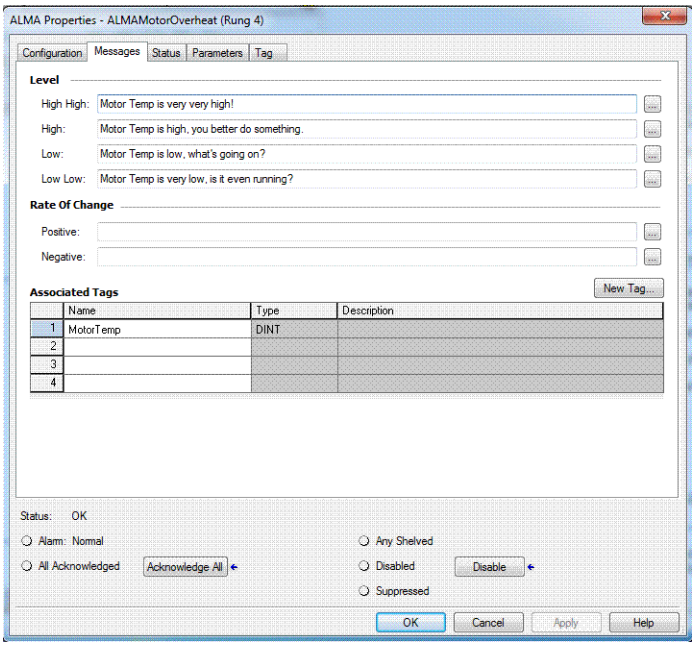
Option	Description
Severity	Select a severity range from 1…1000 to rank the importance of an alarm condition. A severity of 1 is for low priority alarms; a severity of 1000 is for an emergency condition. By default, in the FactoryTalk Alarms and Events system, severity ranges are mapped to priorities as follows: • 1…250 are low priority. • 251…500 are medium priority. • 501…750 are high priority. • 751…1000 are urgent priority. You can configure the severity-to-priority mapping in the FactoryTalk Alarms and Events system. See the FactoryTalk help for details.
Minimum Duration	Enter the amount of time in ms an alarm condition must be active before reporting the alarm.
Latched - ALMD instruction	Select Latched if you want the alarm to stay active (InAlarm) after the alarm condition returns to inactive (normal). Latched alarms require a reset command to transition to normal. The reset command must be received after the condition returns to normal. Acknowledge commands will not reset a latched alarm.
Deadband - ALMA instruction	Specify a Deadband value to reduce alarm condition chattering caused by small fluctuations in the In value. The deadband value does not affect the alarm limit for the transition into the active state, and is also not used during the Minimum Duration interval. Once a level condition becomes active (InAlarm), it will remain active until the In value crosses back over the limit by the specified deadband. For example, if the High limit is 80, the Low limit is 20, and the Deadband is 5, the High condition will be active at ≥ 80 and return to normal at ≤ 75; the Low condition will be active at ≤ 20 and return to normal at ≥ ≤ 25. The Deadband has no effect on Rate of Change alarm conditions.
Acknowledgement Required	Alarms are configured to require acknowledgement by default. Acknowledgement indicates that an operator is aware of the alarm condition, whether or not conditions have returned to normal. Clear the Acknowledgement Required setting when you want the alarm to appear and disappear from the Alarm Summary on the HMI with no operator interaction. Alarms that do not require acknowledgement always have the Acked status set. If a digital alarm is configured as latched, the reset command also acknowledges the alarm.
Alarm class	Use the alarm class to group related alarms. Specify the alarm class exactly the same for each alarm you want in the same class. The alarm class is case sensitive. For example, specify class Tank Farm A to group all the tank alarms for a specific area. Or specify class Control Loop to group all alarms for PID loops. You can then display and filter alarms at the HMI based on the class. For example, an operator can display all tank alarms or all PID loop alarms. The alarm class does not limit the alarms that an Alarm Summary object subscribes to. Use the alarm class to filter the alarms that display to an operator once they have been received by the Alarm Summary object. FactoryTalk View software can filter the alarm class substituting wild cards for characters.
View command	Execute a command on the operator station when requested by an operator for a specific alarm. This lets an operator execute any standard FactoryTalk View command, such as call specific faceplates and displays, execute macros, access help files, and launch external applications. When the alarm condition occurs and is displayed to the operator, a button on the summary and banner displays lets the operator run an associated view command. Be careful to enter the correct command syntax and test the command at runtime as there is no error checking performed when the command is entered.
Shelve Duration	Enter the amount of time in minutes to shelve an alarm condition.
Maximum Shelve Duration	Enter the maximum time duration in minutes for which an alarm condition can be shelved.

You can edit all aspects of the alarm configuration offline and online. Online edits of the configuration impacting run-time behavior on new and existing alarms are immediately sent to FactoryTalk subscribers (legacy HMI terminals that are just polling the tags do not automatically update). FactoryTalk subscribers do not have to re-subscribe to receive updated information. Online changes automatically propagate from the controller alarm structure to the rest of the architecture.

Enter Alarm Message Text

Enter appropriate message text to display when an alarm condition is active (InAlarm). For an ALMD instruction, you enter the message information on the Configuration tab. For an ALMA instruction, you enter the message information on the Message tab.





To define an alarm message, specify this information.

Option	Description
Message string	The message string contains the information to display to the operator regarding the alarm. In addition to entering text, you can also embed variable information. In the alarm message editor, select the variable you want and add it anywhere in the message string. The message string can have a maximum of 255 characters, including the characters that specify any embedded variables (not the number of characters in the actual values of the embedded variables). For example, /*S:0 %Tag1*/ specifies a string tag and adds 13 characters towards the total string length, but the actual value of the string tag could contain 82 characters. You cannot programmatically access the alarm message string from the alarm tag. To change the alarm message based on specific events, configure one of the associated tags as a string data type and embed that associated tag in the message. You can have multiple language versions of messages. You enter the different language via the import/export utility. For more information, see page 76.
Associated tags	You can select as many as four additional tags from the controller project to associate with the alarm. The values of these tags are sent with an alarm message to the alarm server. For example, a digital alarm for a pressure relief valve might also include information such as pump speed and tank temperature. Associated tags may be any atomic data type (BOOL, DINT, INT, SINT, or REAL) or a STRING. They may be elements in a UDT or an Array. Variable array references are not allowed. If the alarm is controller-scoped, the associated tags must also be controller-scoped. Optionally, embed the associated tags into the message text string. Associated tag values are always sent with the alarm, viewable by the operator, and entered in the history log, regardless of whether you embed them in the message string.

Message String Variables

Variable	Embeds in the message string	Default code added to message string
Alarm name	The name of the alarm, which consists of the controller name, program name, and tag name. For example, [Zone1Controller]Program:Main.MyAlarmTagName.	/*S:0 %AlarmName*/
Condition name	The condition that triggers the alarm: • Digital alarm displays the trip. • Analog alarm displays HiHi, Hi, Lo, LoLo, ROC_POS, or ROC_NEG.	/*S:0 %ConditionName*/
Input value	The input value to the alarm: • Digital alarm displays 0 or 1. • Analog alarm displays the value of the input variable being monitored by the alarm.	/*N:5 %InputValue NOFILL DP:0*/
Limit value	The threshold of the alarm: • Digital alarm displays 0 or 1. • Analog alarm displays the actual configured range check for the analog alarm condition.	/*N:5 %LimitValue NOFILL DP:0*/
Severity	The configured severity of the alarm condition.	/*N:5 %Severity NOFILL DP:0*/
Values of associated tags	The value of a tag configured to be included with the alarm event.	/*N:5 %Tag1 NOFILL DP:0*/

The code varies depending on the type of tag you select, how many digits or characters are in a tag value, and whether you want to left fill the empty bits with spaces or zeroes. See the following example.

Tag	Code
BOOL value	/*N:1 %Tag1 NOFILL DP:0*/
DINT value, 9 digits, space left fill	/*N:9 %Tag2 SPACEFILL DP:0*/
REAL input value, 9 digits (includes decimal), 3 digits after decimal, zero left fill	/*N:9 %InputValue NOFILL DP:3*/
REAL value, 8 digits (includes decimal), 4 digits after decimal, zero left fill	/*N:8 %Tag3 ZEROFILL DP:4*/
String value, no fixed width	/*S:0 %Tag4*/
String value, 26 characters, fixed width	/*S:26 %Tag4*/

All of this variable information is included with the alarm data, viewable by the operator, and entered in the history log, regardless of whether you embed the information in the message text.

Multiple Language Versions of Alarm Messages

You can maintain alarm messages in multiple languages. Either enter the different languages in the associated language versions of Logix Designer application or in an import/export (.CSV or .TXT) file.

You can access alarm message text from an import/export (.CSV or .TXT) file and add additional lines for translated versions of the original message string. Messages in different languages use ISO language codes in the TYPE column. Alarm message text, including embedded variable codes, for the operator is in the DESCRIPTION column. The SPECIFIER identifies the alarm condition.

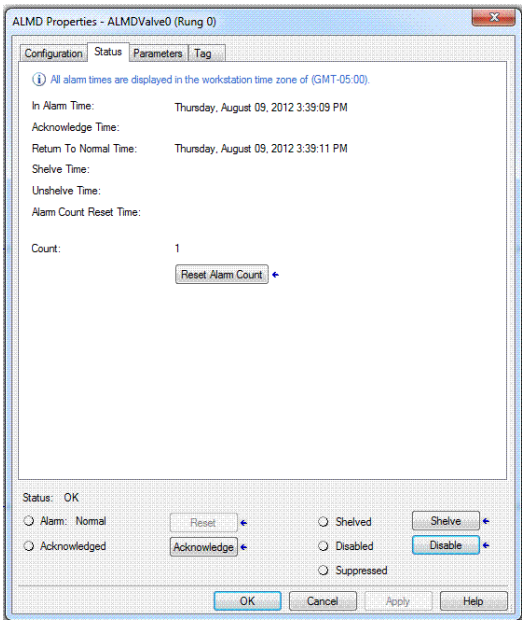
TYPE	SCOPE	NAME	DESCRIPTION	DATATYPE	SPECIFIER
TAG		alma1		ALARM_ANALOG	
ALMMSG:en-us		alma1	HH alarm for operator in English		HH
ALMMSG:en-us		alma1	H alarm for operator in English		H
ALMMSG:en-us		alma1	L alarm for operator in English		L
ALMMSG:en-us		alma1	LL alarm for operator in English		LL
ALMMSG:en-us		alma1	ROC positive alarm for operator in English		POS
ALMMSG:en-us		alma1	ROC positive alarm for operator in English		NEG
ALMMSG:de-ch		alma1	HH Mitteilung für den Operator auf Deutsch		HH
ALMMSG:de-ch		alma1	H Mitteilung für den Operator auf Deutsch		H
TAG		almd1		ALARM_DIGITAL	
ALMMSG:en-us		almd1	digital alarm for operator in English		AM

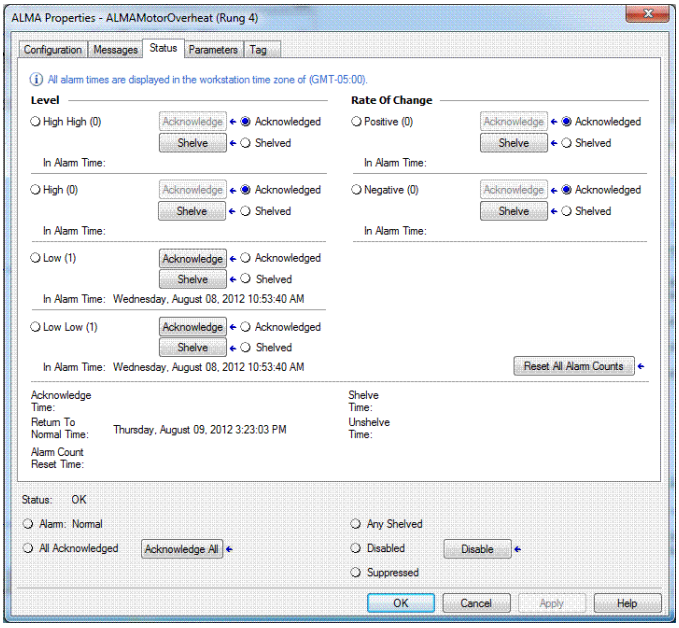
Use the import/export utility to create and translate message strings into multiple languages. The .TXT import/export format supports double-byte characters, so you can use this format for all languages, including Chinese, Japanese, and Korean. The .CSV import/export format does not support double-byte characters.

Importing and exporting messages always performs a merge. Deleting a message in a .CSV or .TXT file does not delete the message from the .ACD file. To delete a message, import the .CSV or .TXT file with the type, name, and specifier fields filled in but the description blank.

Monitor Alarm Status

On the Status tab of the alarm dialog box, monitor the alarm condition, acknowledge an alarm, disable an alarm, suppress an alarm, shelve an alarm or reset an alarm. Use the dialog box selections to see how an alarm behaves, without needing an operational HMI.





Alarms Log

In order to receive controller-based alarm messages, the alarm client (Factory Talk Alarms and Events server) must establish a subscription to the Alarm Log object in the Logix controller.

As alarm state changes occur, the alarm instructions in the controller store the necessary information (such as timestamps and associated tag values) to the Alarm Log. The publisher mechanism delivers the alarm messages to each subscriber as quickly as possible.

When the Alarm Log is full, the oldest alarm messages are rewritten. The size of the Alarm Log is controller-family dependent.

When the subscriber re-establishes a connection, it begins to receive current alarm messages. The historic data is available and the subscriber can obtain the sequence number of the first (and last message) to see what data is missed during the disconnect period.

Programmatically Access Alarm Information

Each alarm instruction has an alarm structure that stores alarm configuration and execution information. The alarm structure includes both Program and Operator control elements and operator elements. The alarm instructions do not use mode settings to determine whether program access or operator access is active, so these elements are always active.

There are three ways to perform actions on an alarm instruction.

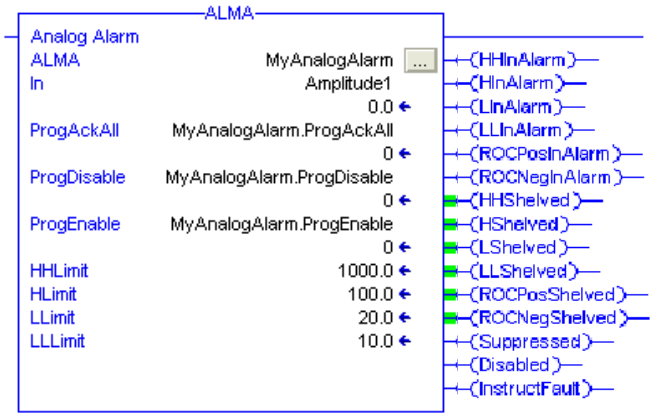
Access	Alarm Structure Elements	Considerations
User program	• ProgAck • ProgReset • ProgUnshelve • ProgSuppress • ProgDisable • ProgEnable	Use controller logic to programmatically access elements of the alarming system. For example, the control program can determine whether to disable a series of alarms that are related to a single root cause. For example, the control program could disable an alarm instruction, MyDigitalAlarm of data type ALARM_DIGITAL, by accessing the tag member MyDigitalAlarm.ProgDisable.
Custom HMI	• OperAck • OperReset • OperShelve • OperUnshelve • OperSuppress • OperDisable • OperEnable	Create a custom HMI faceplate to access elements of the alarming system. For example, if the operator needs to remove a tool, rather than manually disable or suppress alarms individually from the alarming screens, the operator can press a disable key that accesses a tag MyDigitalAlarm.OperDisable. Operator parameters work with any Rockwell Automation or third-party operator interface to allow control of alarm states. When an operator parameter is set, the instruction evaluates whether it can respond to the request, then always resets the parameter.
Standard HMI object	Not accessible	Normal operator interaction is through the alarm summary, alarm banner, and alarm status explorer objects in the FactoryTalk View application. This interaction is similar to the custom HMI option described above, but there is no programmatic visibility or interaction.

When you create an alarm instruction, you must create and assign a tag of the correct alarm data type for that alarm. For example, create MyDigitalAlarm of data type ALARM_DIGITAL. In relay ladder, these instruction parameters must be entered on the instruction:

- ProgAck
- ProgReset
- ProgDisable
- ProgEnable

In relay ladder and structured text, the value or tag you assign to an instruction parameter (such as ProgAck) is automatically written to the alarm tag member (such as MyAnalogAlarm.ProgAck) each time the instruction is scanned.

In relay ladder and structured text, if you want to programmatically access the alarm structure, assign the structure tag to the parameter on the instruction. For example, to use MyAnalogAlarm.ProgAck in logic, assign the tag MyAnalogAlarm.ProgAck to the ProgAck parameter.



Shelve, Suppress, or Disable Alarms

FactoryTalk Alarms and Events system provides three mechanisms to prevent indication of the alarm in the alarm summary: suppress, shelve, and disable.

Shelve and Suppress allow you to clear the alarm from the alarm summary or banner while you are resolving a known alarm, without continuing to view alarm information once the alarm is acknowledged. Shelve has an automatic timeout, after which the alarm is automatically Unshelved and returned to the operator view. Suppress does not have an automatic timeout. If the alarm is unacknowledged at the time it is Shelved or Suppressed, it will continue to appear on the alarm summary and banner until it has been acknowledged, at which point, it will be removed from view. A Shelved or Suppressed alarm is still able to transition alarm status (except becoming unacknowledged), send alarm state changes to subscribers, log state changes in the historical database, and is responsive to other programmatic or operator interactions:

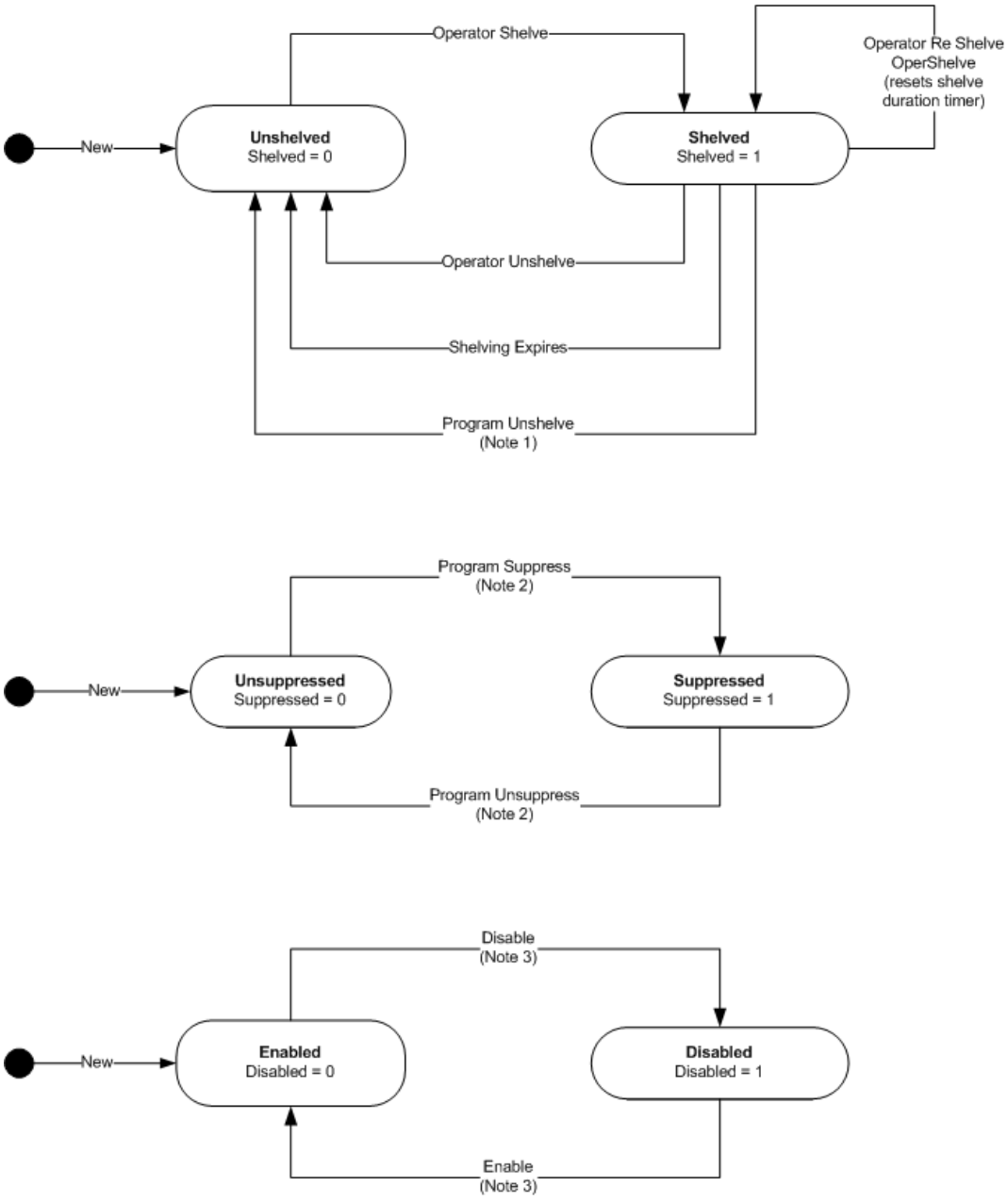
- When an alarm is Suppressed or Shelved, it continues to function normally, monitor the In parameter for alarm conditions, and respond to Acknowledge requests. All subscribers are notified of this event, and any alarm messages generated while the alarm is in the Suppressed or Shelved state include the Suppressed or Shelved status. An alarm is not allowed to become unacknowledged while Shelved or Suppressed.
- When an alarm is Unsuppressed or Unshelved, all subscribers are notified and alarm messages to subscribers no longer include the Suppressed or Shelved status. If the alarm is active when Unsuppressed or Unshelved and acknowledge is required, the alarm becomes unacknowledged (Acked is cleared).

Disable an alarm to take the alarm out-of-service in the control program. A disabled alarm does not transition alarm status or get logged in the historical database. If the alarm is Unacknowledged at the time it is Disabled, it will continue to appear on the alarm summary and banner until it has been Acknowledged, at which point it will be removed from view. A disabled alarm can be re-enabled in the Alarm Status Explorer in FactoryTalk View SE software:

- When an alarm is Disabled, all of its conditions are inactivated (InAlarm is cleared) except the Acknowledged status if Unacknowledged. The In parameter is not monitored for alarm conditions, but will respond to an Acknowledged event. All subscribers are notified of this event.
- When an alarm is Enabled, it begins to monitor the In parameter for alarm conditions. All subscribers are notified of this event. If the alarm is active when Enabled and Acknowledge is required, the alarm becomes Unacknowledged (Acked is cleared).

Shelve, Suppress, and Disable are all methods to suppress indication of alarms, following ANSI/ISA-18.2-2009, Management of Alarm Systems for the Process Industries. You can use Shelve, Suppress, and Disable functionality to track operator-initiated actions from design-initiated actions and maintenance actions. Following this method, use Shelve to initiate this action by the operator (equivalent to the Shelve state in ISA 18.2). The controller must use Suppress to programmatically inhibit operator notification (equivalent to the Suppress-by-Design state in ISA 18.2). Use Disable to inhibit the alarm for maintenance purposes (equivalent to Out-of-Service state in ISA 18.2).

Figure 1 - State Model for Alarms



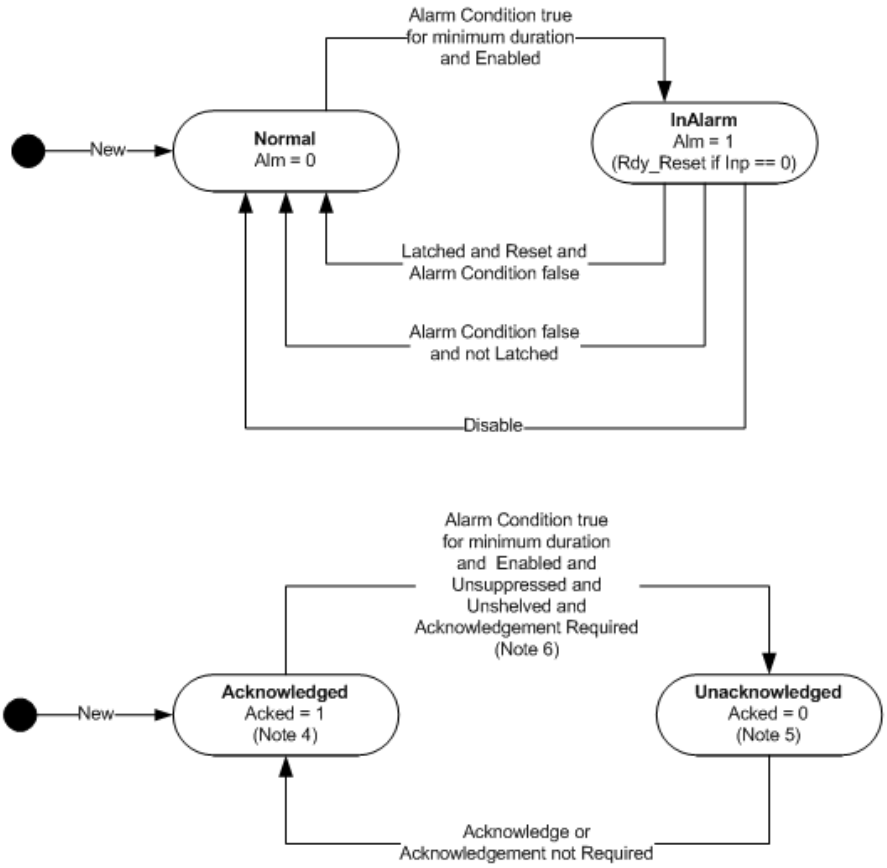


Table 2 - State Model for Alarms

Note	Description
1	The Program Unshelve command is provided so that the user has a means, using a small amount of programming, to unshelve alarms based on an event, for example End of Shift.
2	Following ANSI/ISA-18.2-2009, Suppress could be used to provide Suppress-by-Design functionality. Suppress could also be used to provide Out-of-Service if logging while Out-of-Service is desired and logging while Suppressed-by-Design is not desired. Operator commands for Suppress/Unsuppress are provided, but would not be used if using Suppress to implement Suppress-by-Design as outlined in ISA 18.2.
3	Following ANSI/ISA-18.2-2009, Disable could be used to provide Out-of-Service state described by the standard. Disable could also be used to provide Suppress-by-Design if logging while Suppressed-by-Design is not desired and logging while Out-of-Service is desired. Operator commands for Enable/Disable are provided, but would not be used if using Suppress to implement Suppress-by-Design as outlined in ISA 18.2.
4	While Disabled, Suppressed, or Shelved, an acknowledged alarm will not become unacknowledged.
5	While Disabled, Suppressed, or Shelved, if acknowledge is required, an unacknowledged alarm will remain unacknowledged until it is acknowledged.
6	An alarm will become unacknowledged if the alarm is InAlarm when an alarm changes state to Enabled, Unsuppressed, and Unshelved.

Controller-based Alarm Execution

Controller-based alarms process inputs from two sources.

Source	Description
Alarm tag members	Alarm tag members are, for the most part, processed when the user application scans the alarm instruction. This includes: • processing changes to configuration parameters. • evaluating the alarm condition. • measuring elapsed time for MinDuration. • capturing InAlarmTime and RetToNormalTime timestamps. • capturing associated tag values. • processing Prog and Oper commands.
Client messages	Client messages are processed as they are received, asynchronously to the program scan: • Reset, Acknowledge, Disable/Enable, Shelve/Unshelve and Suppress/Unsuppress commands from a Studio 5000 Environment terminal • Reset, Acknowledge, Disable/Enable, Shelve/Unshelve and Suppress/Unsuppress commands from a FactoryTalk View SE alarm subscriber

Use care when determining where to place alarm instructions in the application. The accuracy of the timestamps are affected by how quickly the instruction is scanned after the alarm condition changes state. MinDuration time accumulation and Rate of Change calculations require repeated scanning, within time intervals determined by the user application. Alarm instructions must continue to be scanned after the alarm condition becomes false, so that the ReturnToNormal transition may be detected. For example, if you desire 10 ms accuracy on timestamps, you could place the alarm instructions that need that resolution in a 10 ms periodic task.

Controller Memory Use

As a guideline, use the following alarm sizes for a rough calculation of controller memory usage:

- Typically 1 KB per digital alarm with no associated tags

Digital Alarm Example	Approximate Size
Digital alarm with no associated tags	1056 bytes
Digital alarm with no associated tags and this configuration: • Alarm message: Contactor Fault • Alarm Class: Tank Farm A	1144 bytes
Digital alarm with two associated tags and this configuration: • Alarm message: Contactor Fault • Alarm Class: Tank Farm A • Associated Tag 1 = DINT data type • Associated Tag 2 = DINT data type	1216 bytes
Digital alarm with two associated tags and this configuration: • Alarm message: Contactor Fault • Alarm Class: Tank Farm A • Associated Tag 1 = DINT data type • Associated Tag 2 = STRING data type	1560 bytes

- Typically 2.2 KB per analog alarm with no associated tags

Analog Alarm Example	Approximate Size
Analog alarm with no associated tags	2496 bytes
Analog alarm with no associated tags and this configuration: • HH Alarm message: Level Alarm • H Alarm Message: Level Alarm • L Alarm Message: Level Alarm • LL Alarm Message: Level Alarm • Rate of Change Positive Message: Fill Too Fast • Rate of Change Negative Message: Empty Too Fast • Alarm Class: Tank Farm A	2624 bytes
Analog alarm with two associated tags and this configuration: • HH Alarm message: Level Alarm • H Alarm Message: Level Alarm • L Alarm Message: Level Alarm • LL Alarm Message: Level Alarm • Rate of Change Positive Message: Fill Too Fast • Rate of Change Negative Message: Empty Too Fast • Alarm Class: Tank Farm A • Associated Tag 1 = DINT data type • Associated Tag 2 = DINT data type	2952 bytes
Analog alarm with two associated tags and this configuration: • HH Alarm message: Level Alarm • H Alarm Message: Level Alarm • L Alarm Message: Level Alarm • LL Alarm Message: Level Alarm • Rate of Change Positive Message: Fill Too Fast • Rate of Change Negative Message: Empty Too Fast • Alarm Class: Tank Farm A • Associated Tag 1 = DINT data type • Associated Tag 2 = STRING data type	4632 bytes

Longer message strings, as well as message strings for multiple languages, consume additional memory from your controller.

Actual memory usage will depend on how the alarm is configured, message length, and any associated tags passed with the alarm.

Scan Time

These execution times show how ALMD instructions and ALMA instructions affect total scan time in a 1756-L7*x* nonredundant controller.

Rung State	Execution Times	
	Digital Alarm (ALMD)	**Analog Alarm (ALMA)**
No Input/State change	6.6 µs	—
Activation	15.7 µs	—
EnableIn true (rung value for RLL) No Input/State change	—	15 µs
HAlarm Condition/ Activation	—	21.1 µs
HHAlarm Condition/ Activation	—	25 µs
EnableIn/False	—	4.7 µs

An alarm state change is any event that changes the condition of the alarm, such as acknowledging or suppressing the alarm. Minimize the potential for a large number of alarms changing state simultaneously (alarm bursts) by creating dependencies on related alarms. Large alarm bursts can have a significant impact on application code scan time.

Notes:

Bit Instructions

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

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

If you want to	Use this instruction	Available in	Page
Enable outputs when a bit is set	XIC	Relay ladder Structured text(1)	90
Enable outputs when a bit is cleared	XIO	Relay ladder Structured text(1)	93
Set a bit	OTE	Relay ladder Structured text(1)	96
Set a bit (retentive)	OTL	Relay ladder Structured text(1)	98
Clear bit (retentive)	OTU	Relay ladder Structured text(1)	100
Enable outputs for one scan each time a rung goes true	ONS	Relay ladder Structured text(1)	102
Set a bit for one scan each time a rung goes true	OSR	Relay ladder	105
Set a bit for one scan each time the rung goes false	OSF	Relay ladder	107
Set a bit for one scan each time the input bit is set in function block	OSRI	Structured text Function block	109
Set a bit for one scan each time the input bit is cleared in function block	OSFI	Structured text Function block	112

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

Bit Addressing

Bits in a Logix controller exist in one of several forms:

- BOOL tag
- BOOL array
- Bits within an integer (SINT, INT, or DINT) tag
- Bits within an integer array

Any of the above constructs may also exist as members of a structure, whether it's a system defined data type (SDT) or user-defined data type (UDT).

Bits defined as BOOLs are referenced simply by name. For example, XIC myBool.

Bits within BOOL arrays are referenced by a subscript appended to the array name. Subscripts can be a literal value (for example, XIC boolArray[31]), a tag (for example, XIC boolArray[index]), or an expression (for example, XIC boolArray[index+1]).

Bits within an integer tag or integer array may be addressed similarly. This is what Logix often refers to as bit addressing mode for integers. IEC 61131-3 might refer to it as 'partial access of ANY_BIT variables'. Bit addressing for integers is programmed by adding a dot (or period) to the integer indicating subelement processing followed by a subscript (for example, myInteger[3].11 addresses bit 11, subelement [3]of the myInteger array). There is a shorthand form for literal subscripts in this case, which is simply a dot followed by the literal value (for example, myInteger.3, meaning bit 3 of myInteger). Similar to subscripts for BOOL arrays above, one can specify a tag or expression for the subscript.

Integer arrays are dealt with similarly except that you must specify a member of the array from which indexing begins. For example, XIC myIntArray[4].[index+1] says to start at the 5th element of myIntArray and access bit 'index+1.' Bounds checking is performed on all subscripts to make sure the reference does not go beyond the boundary of the data object specified. Literals are checked for conformance during verification.

Example: Indirect Serial Bit Reference in a DINT array

Tag

MyBits : DINT[10]

BitRef : DINT

EndTag

MOV(34,BitRef)

XIC(MyBits[(BitRef AND NOT 31)/32].[BitRef AND 31])

Explanation

(BitRef AND NOT 31)/32=Calculates the element in the DINT

.[BitRef AND 31]=Calculates the bit within the element

If the tag MyBits is defined as an INT or SINT, the mask value would be 15 or 7, respectively. Both the Diagnostic Detect (DDT) and File Bit Compare (FBC) instructions provide a bit number as a result of their operation. These instructions are limited to DINT arrays, and so the indirect addressing example can be used to programmatically locate the bit number returned from these instructions.

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 by using an IF...THEN construct.

IF data_bit THEN

<*statement*>;

END_IF;

See Function Block Attributes 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:

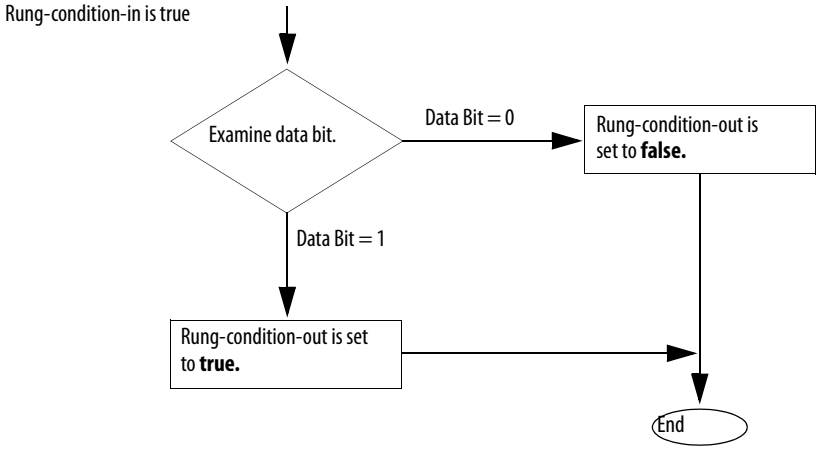
Not affected

Fault Conditions:

None

Execution:

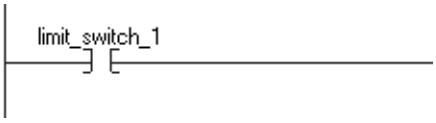
Condition	Relay Ladder Action
Prescan	The rung-condition-out is set to false.
Rung-condition-in is false	The rung-condition-out is set to false.



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

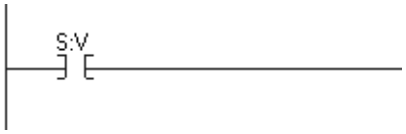
IF limit_switch THEN

<*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 by using an IF...THEN construct.

IF NOT data_bit THEN

<*statement*>;

END_IF;

See Function Block Attributes 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:

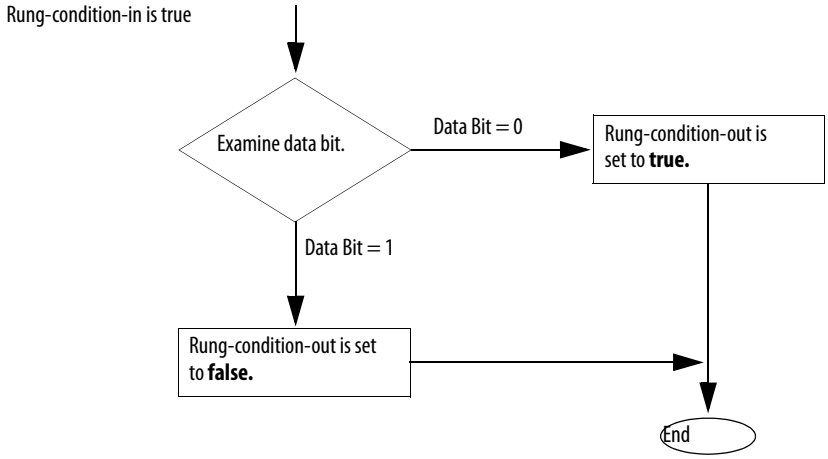
Not affected

Fault Conditions:

None

Execution:

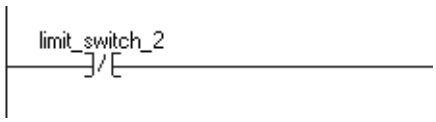
Condition	Relay Ladder Action
Prescan	The rung-condition-out is set to false.
Rung-condition-in is false	The rung-condition-out is set to false.



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

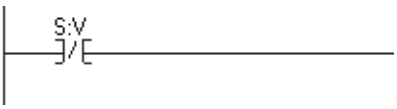
IF NOT limit_switch_2 THEN

<*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 by using a non-retentive assignment.

data_bit [:=] BOOL_expression;

See Function Block Attributes 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 by using an IF...THEN construct and an assignment.

IF BOOL_expression THEN

data_bit := 1;

END_IF;

See Function Block Attributes 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:

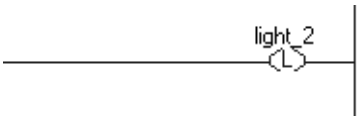
None

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 by using an IF...THEN construct and an assignment.

```
IF BOOL_expression THEN

data_bit := 0;

END_IF;
```

See Function Block Attributes 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;

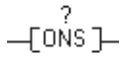
END_IF;

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 by using an IF...THEN construct.

IF BOOL_expression AND NOT storage_bit THEN

<*statement*>;

END_IF;

storage_bit := *BOOL_expression*;

See Function Block Attributes 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:

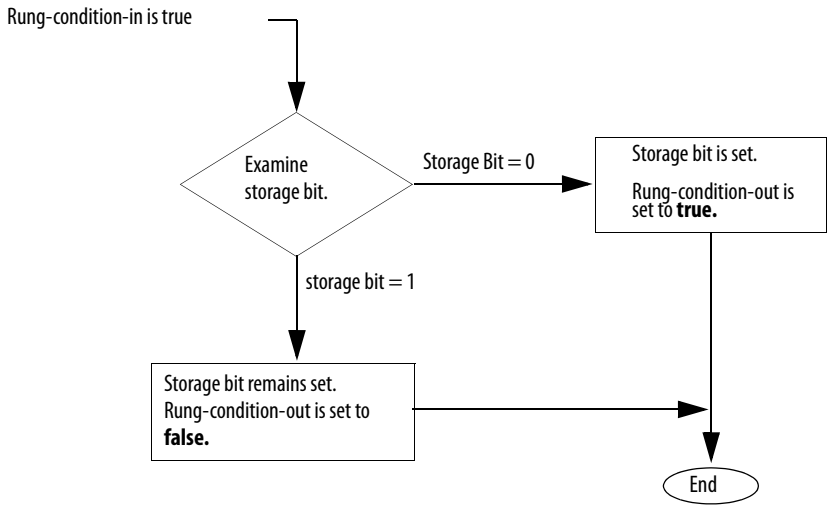
Not affected

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.



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;
```

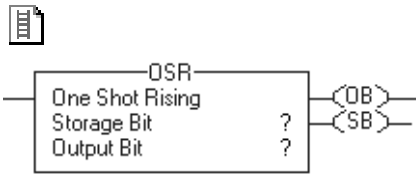
One Shot Rising (OSR)

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 109.

Operands:

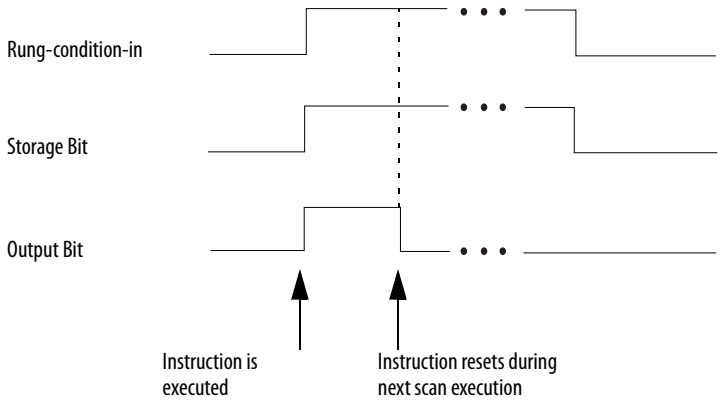
Relay Ladder



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:

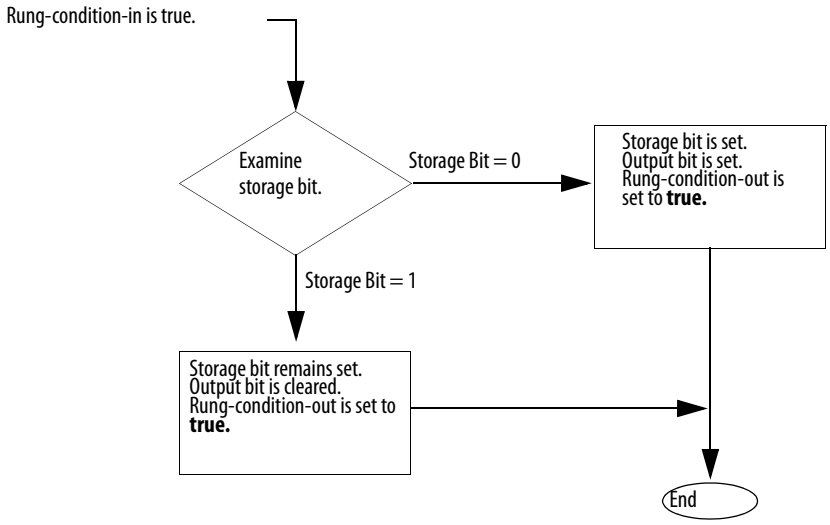
Not affected

Fault Conditions:

None

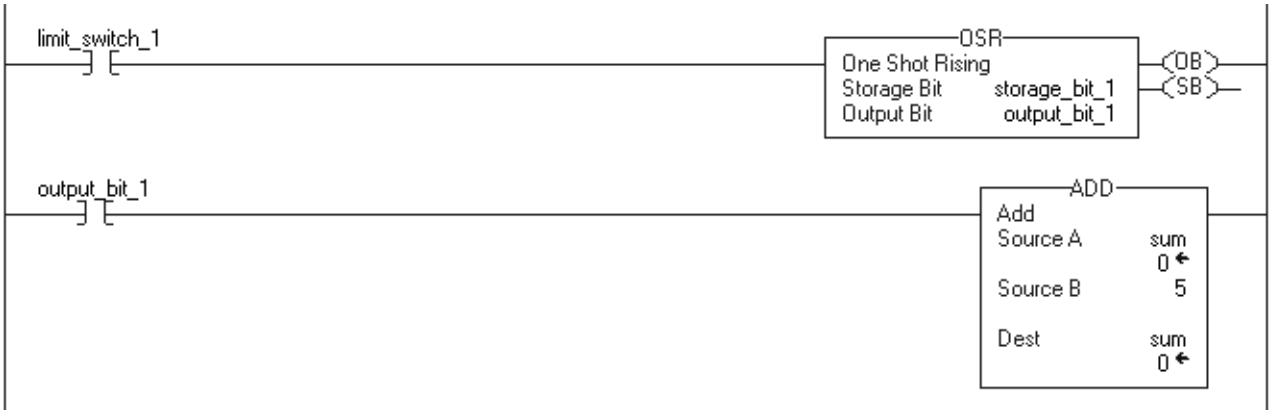
Execution:

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.



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 five. 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

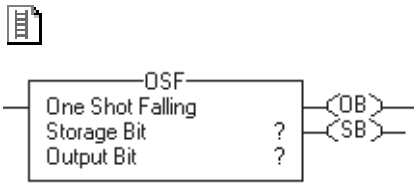


One Shot Falling (OSF)

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 112.

Operands:

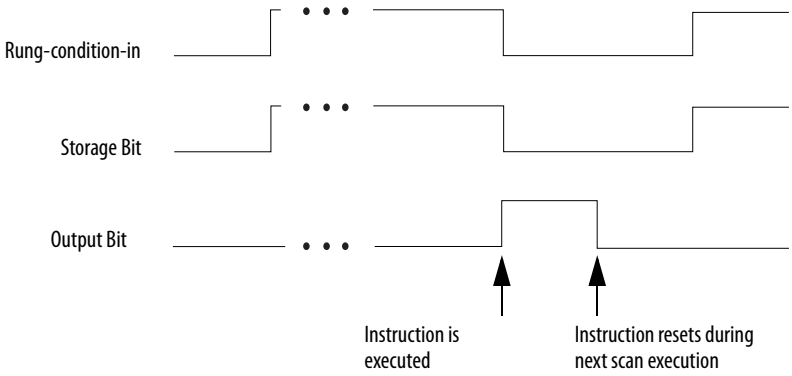


Relay Ladder Operands

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:

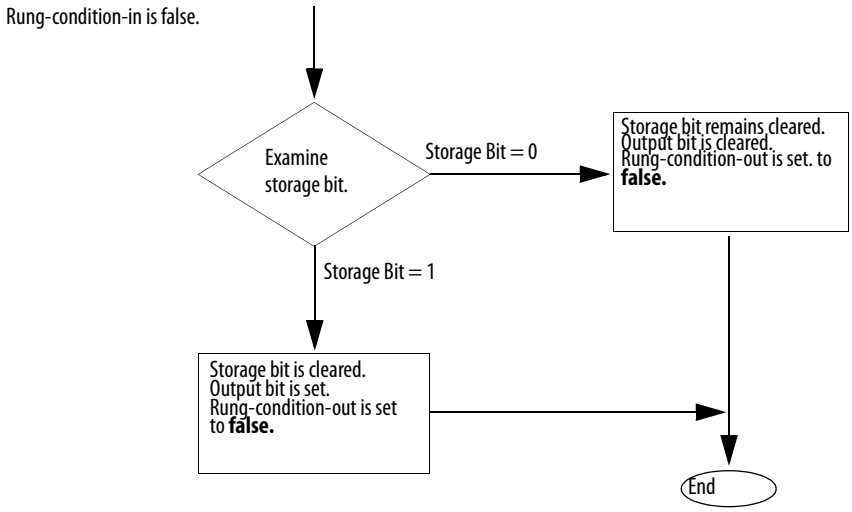
Not affected

Fault Conditions:

None

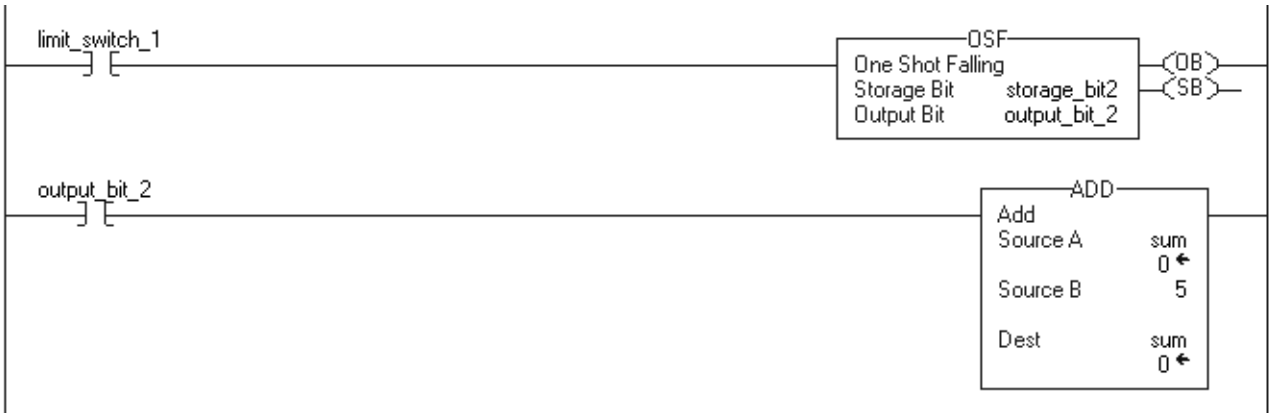
Execution:

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.



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 105.

Operands:

OSRI(OSRI_tag);

Structured Text

Operand	Type	Format	Description
OSRI tag	FBD_ONESHOT	Structure	OSRI structure

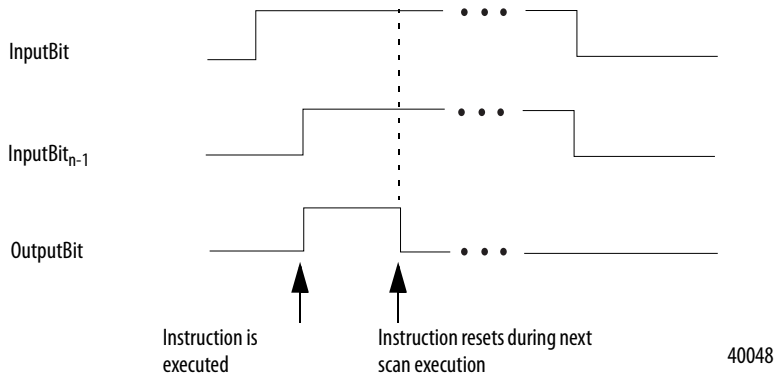
Function Block

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:

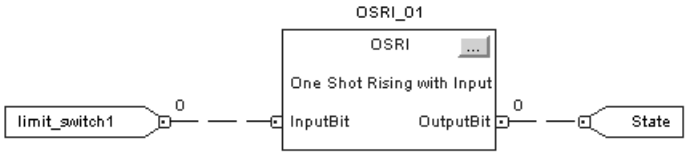
Condition	Function Block Action	Structured Text Action
Prescan	No action taken.	No action taken.
Instruction first scan	$InputBit_{n-1}$ is set.	$InputBit_{n-1}$ is set.
Instruction first run	$InputBit_{n-1}$ is set.	$InputBit_{n-1}$ is set.
EnableIn is cleared	EnableOut is cleared, the instruction does nothing, and the outputs are not updated.	N/A
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.

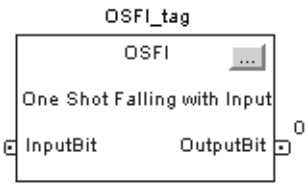
This instruction is available in relay ladder as OSF, see page 107.

Operands:

OSFI(OSFI_tag);

Structured Text

Operand	Type	Format	Description
OSFI tag	FBD_ONESHOT	Structure	OSFI structure



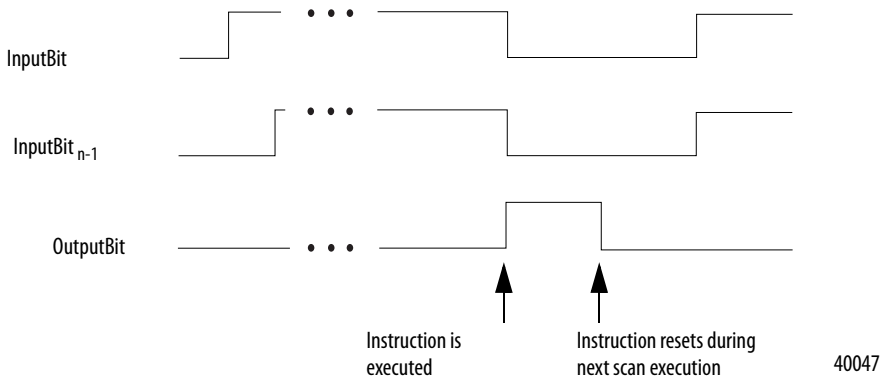
Function Block

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.



Arithmetic Status Flags: Not affected

Fault Conditions: None

Execution:

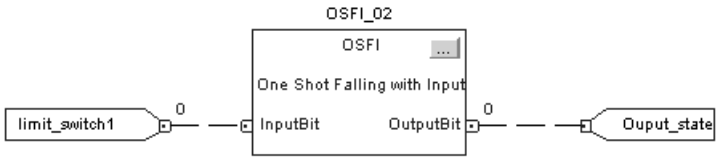
Condition	Function Block Action	Structured Text Action
Prescan	No action taken.	No action taken.
Instruction first scan	InputBit $_{n-1}$ is cleared.	InputBit $_{n-1}$ is cleared.
Instruction first run	InputBit $_{n-1}$ is cleared.	InputBit $_{n-1}$ is cleared.
EnableIn is cleared	EnableOut is cleared, the instruction does nothing, and the outputs are not updated.	N/A
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)

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

If you want to	Use this instruction	Available in these languages	Page
Time how long a timer is enabled	TON	Relay ladder	116
Time how long a timer is disabled	TOF	Relay ladder	120
Accumulate time	RTO	Relay ladder	124
Time how long a timer is enabled with built-in reset in function block	TONR	Structured text Function block	128
Time how long a timer is disabled with built-in reset in function block	TOFR	Structure text Function block	132
Accumulate time with built-in reset in function block	RTOR	Structured text Function block	136
Count up	CTU	Relay ladder	140
Count down	CTD	Relay ladder	144
Count up and count down in function block	CTUD	Structured text Function block	148
Reset a timer or counter	RES	Relay ladder	152

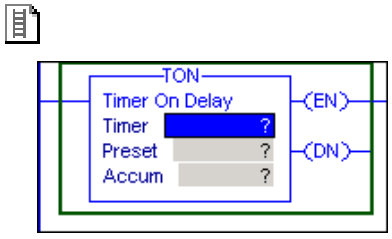
The time base for all timers is 1 ms.

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.

Operands:



Relay Ladder

Operand	Type	Format	Description
Timer	TIMER	Tag	Timer structure
Preset	DINT	Immediate Tag	How long to delay (accumulate time)
Accum	DINT	Immediate Tag	Total milliseconds 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 ≥ .PRE.
.PRE	DINT	The preset value specifies the value (1 ms units) that 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.

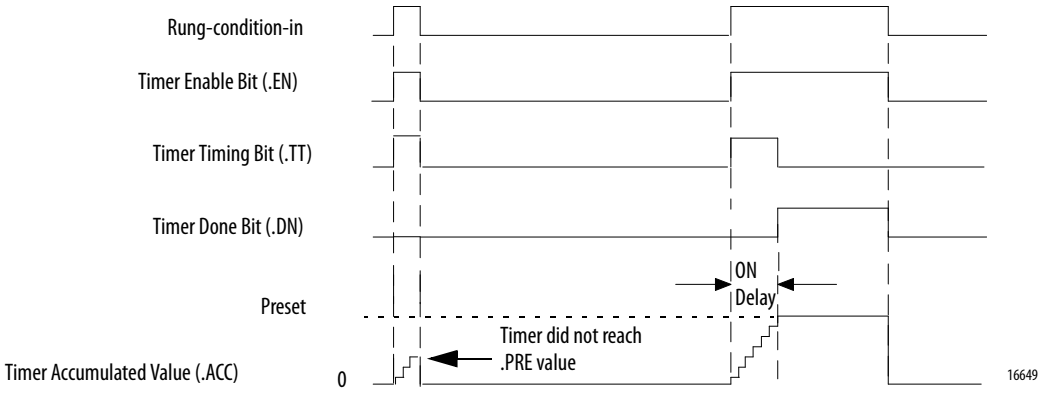
Description:

The TON instruction accumulates time until the following occurs:

- The TON instruction is disabled
- The .ACC ≥ .PRE

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

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



A timer runs by subtracting the time of its last scan from the time now:

$$ACC = ACC + (current\_time - last\_time\_scanned)$$

After it updates the ACC, the timer sets last_time_scanned = current_time. This gets the timer ready for the next scan.

IMPORTANT

Make sure to scan the timer at least every 69 minutes while it runs. Otherwise, the ACC value won't be correct.

The *last_time_scanned* value has a range of up to 69 minutes. The timer's calculation rolls over if you don't scan the timer within 69 minutes. The ACC value won't be correct if this happens.

While a timer runs, scan it within 69 minutes if you put it in the following:

- A subroutine
- A section of code that is between JMP and LBL instructions
- A sequential function chart (SFC)
- An event or periodic task
- A state routine of a phase

Arithmetic Status Flags:

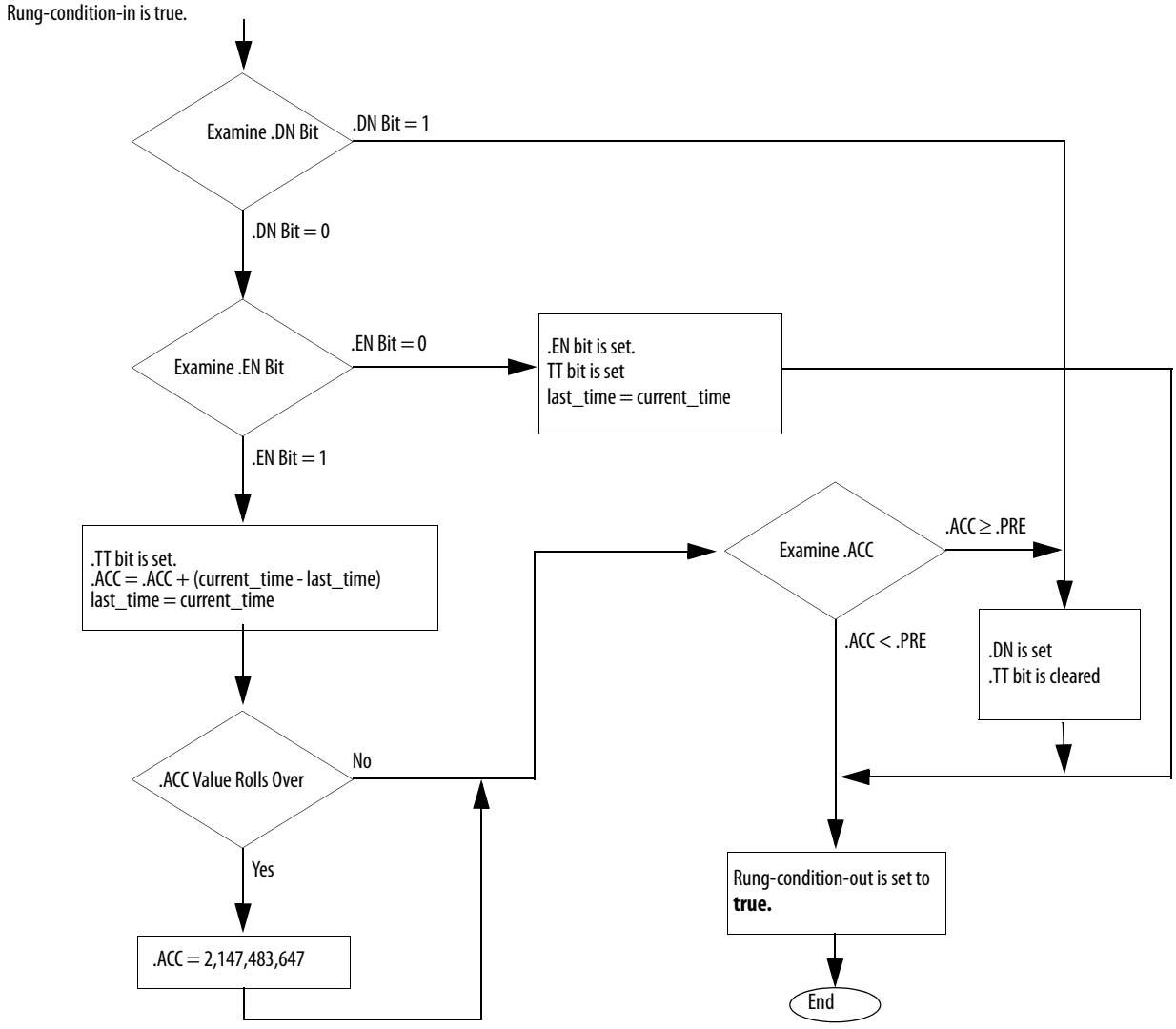
Not affected

Fault Conditions:

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

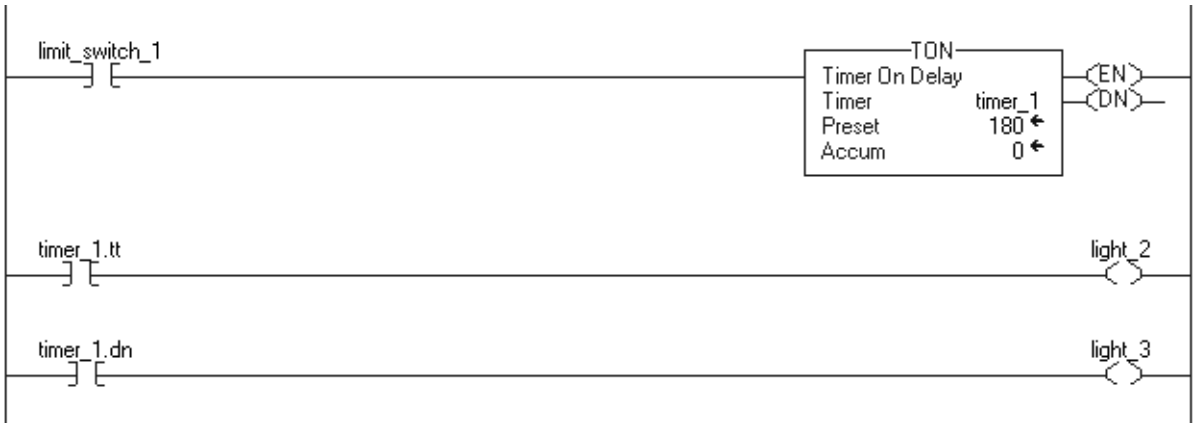
Execution:

Condition	Relay Ladder Action
Prescan	The .EN, .TT, and .DN bits are cleared. The .ACC value is cleared. The rung-condition-out is set to false.
Rung-condition-in is false	The .EN, .TT, and .DN bits are cleared. The .ACC value is cleared. The rung-condition-out is set to false.



Postscan	The rung-condition-out is set to false.

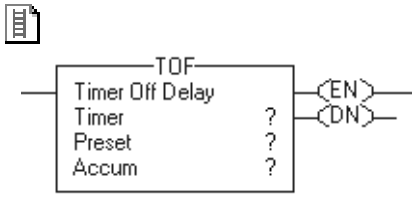
Example: When *limit_switch_1* is set, *light_2* is on for 180 ms (timer_1 is timing). When *timer_1.acc* reaches 180, *light_2* goes off and *light_3* goes on. *Light_3* remains on until the TOF instruction is disabled. If *limit_switch_1* is cleared while *timer_1* is timing, *light_2* goes off.



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.

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 milliseconds 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 ≥ .PRE.
.PRE	DINT	The preset value specifies the value (1 ms units) that 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.

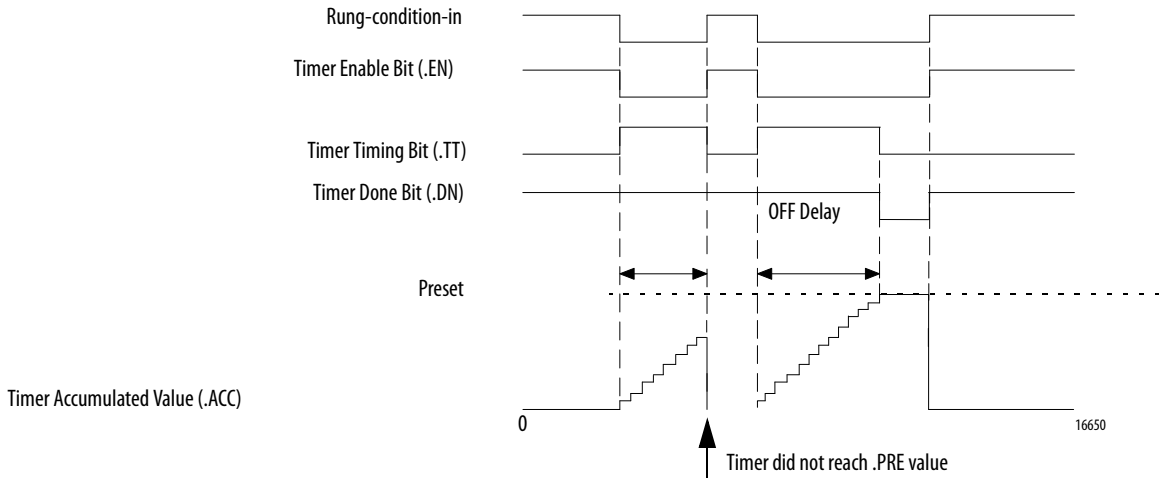
Description:

The TOF instruction accumulates time until the following occurs:

- The TOF instruction is disabled
- The .ACC ≥ .PRE

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

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



A timer runs by subtracting the time of its last scan from the time now:

$$ACC = ACC + (current\_time - last\_time\_scanned)$$

After it updates the ACC, the timer sets last_time_scanned = current_time. This gets the timer ready for the next scan.

IMPORTANT

Make sure to scan the timer at least every 69 minutes while it runs. Otherwise, the ACC value won't be correct.

The *last_time_scanned* value has a range of up to 69 minutes. The timer's calculation rolls over if you don't scan the timer within 69 minutes. The ACC value won't be correct if this happens.

While a timer runs, scan it within 69 minutes if you put it in the following:

• A subroutine

- A section of code that is between JMP and LBL instructions
- A sequential function chart (SFC)
- An event or periodic task
- A state routine of a phase

Arithmetic Status Flags:

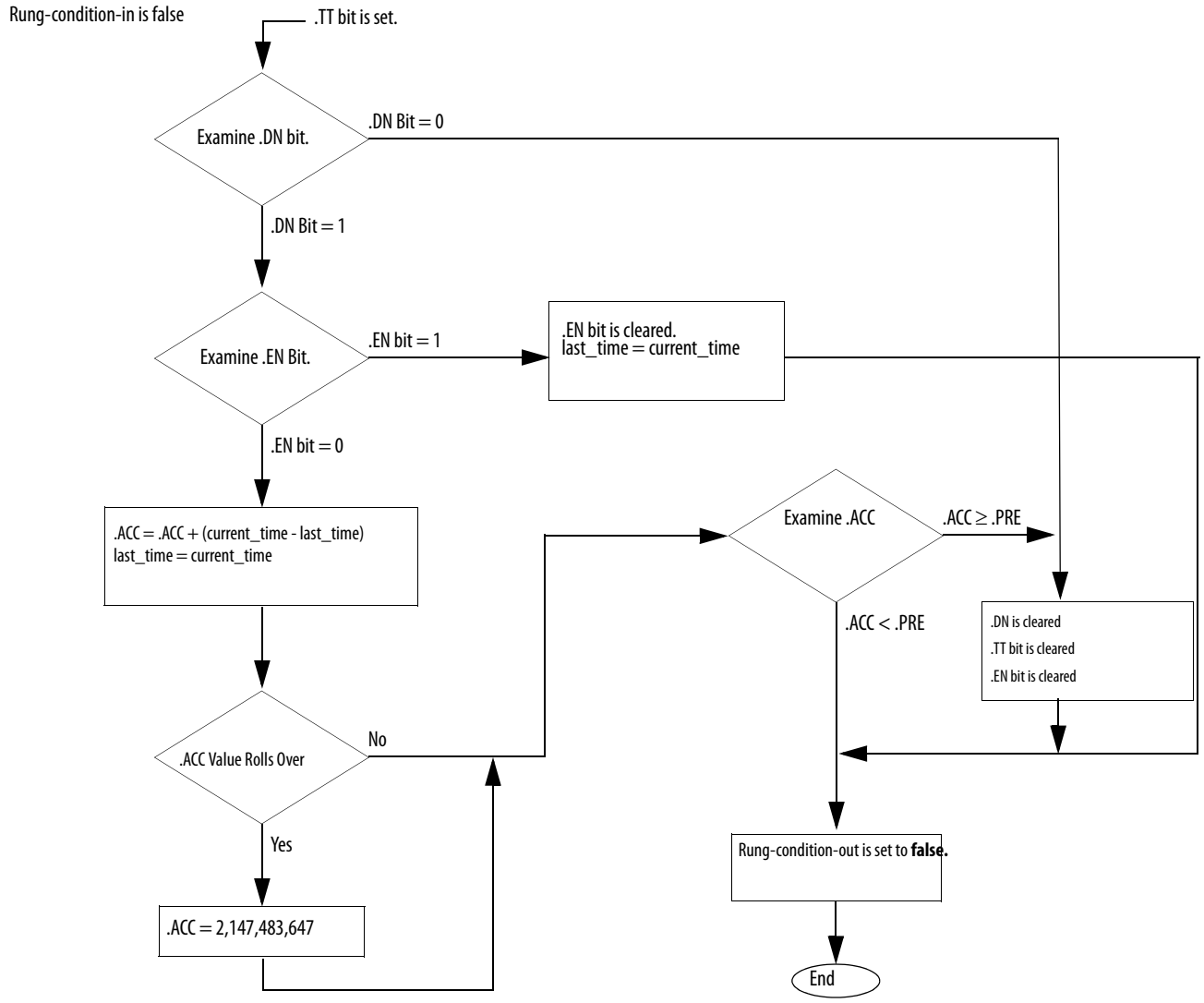
Not affected

Fault Conditions:

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

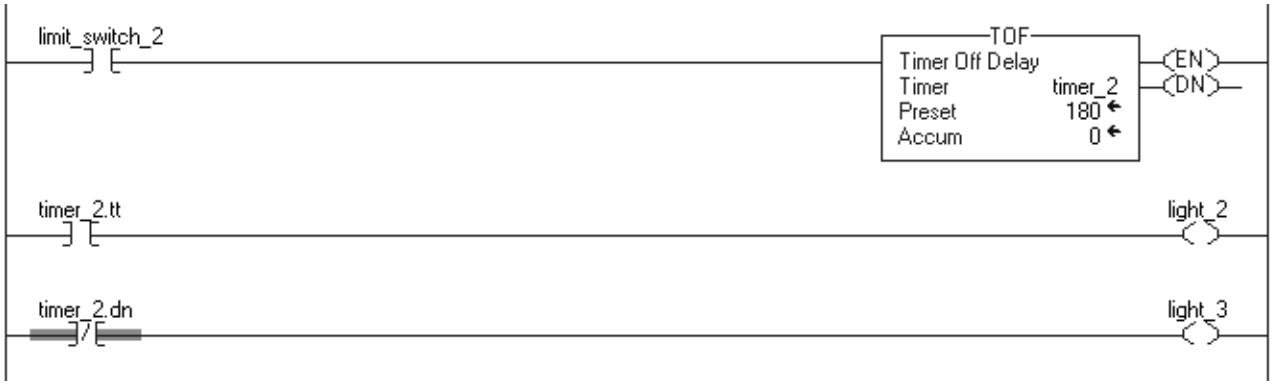
Execution:

Condition	Relay Ladder Action
Prescan	The .EN, .TT, and .DN bits are cleared. The .ACC value is set to equal the .PRE value. The rung-condition-out is set to false.



Rung-condition-in is true	The .EN, .TT, and .DN bits are set. The .ACC value is cleared. The rung-condition-out is set to true.
Postscan	The rung-condition-out is set to false.

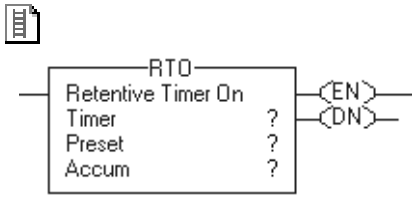
Example: When *limit_switch_2* is cleared, l*ight_2* is on for 180 ms (timer_2 is timing). When *timer_2.acc* reaches 180, *light_2* goes off and *light_3* goes on. *Light_3* remains on until the TOF instruction is enabled. If *limit_switch_2* is set while *timer_2* is timing, *light_2* goes off.



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.

Operands:



Relay Ladder

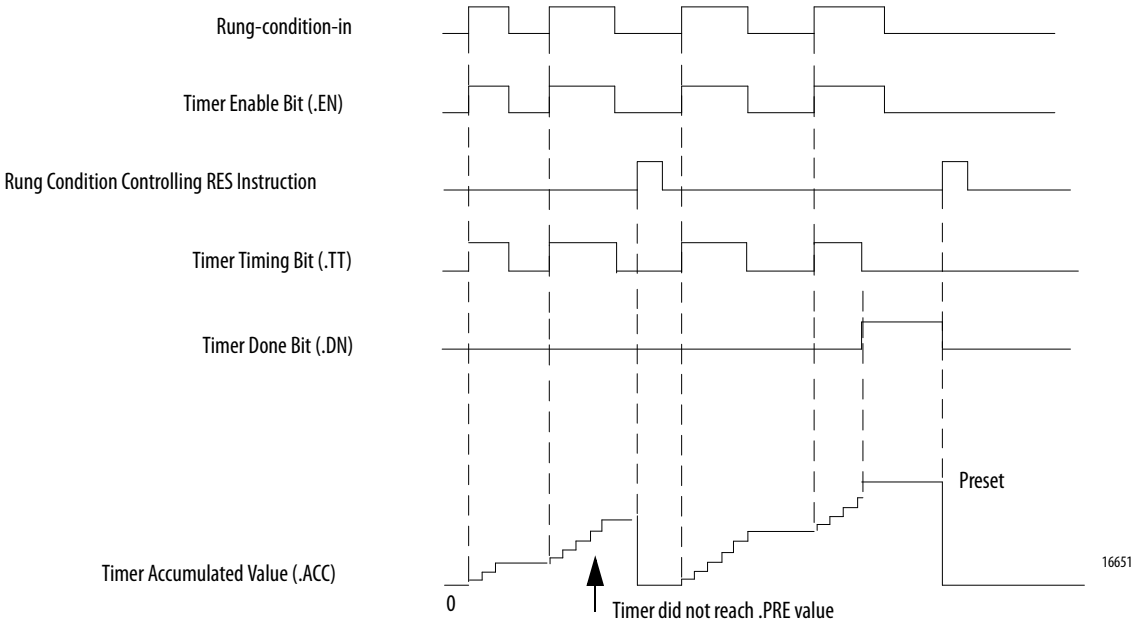
Operand	Type	Format	Description
Timer	TIMER	Tag	Timer structure
Preset	DINT	Immediate	How long to delay (accumulate time)
Accum	DINT	Immediate	Number of milliseconds the timer has counted Initial value is typically 0

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 ≥ .PRE.
.PRE	DINT	The preset value specifies the value (1 ms units) that 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 ms. For example, for a 2-second timer, enter 2000 for the .PRE value.



A timer runs by subtracting the time of its last scan from the time now:

$$ACC = ACC + (current\_time - last\_time\_scanned)$$

After it updates the ACC, the timer sets last_time_scanned = current_time. This gets the timer ready for the next scan.

IMPORTANT

Make sure to scan the timer at least every 69 minutes while it runs. Otherwise, the ACC value won't be correct.

The *last_time_scanned* value has a range of up to 69 minutes. The timer's calculation rolls over if you don't scan the timer within 69 minutes. The ACC value won't be correct if this happens.

While a timer runs, scan it within 69 minutes if you put it in the following:

- A subroutine
- A section of code that is between JMP and LBL instructions
- A sequential function chart (SFC)
- An event or periodic task
- A state routine of a phase

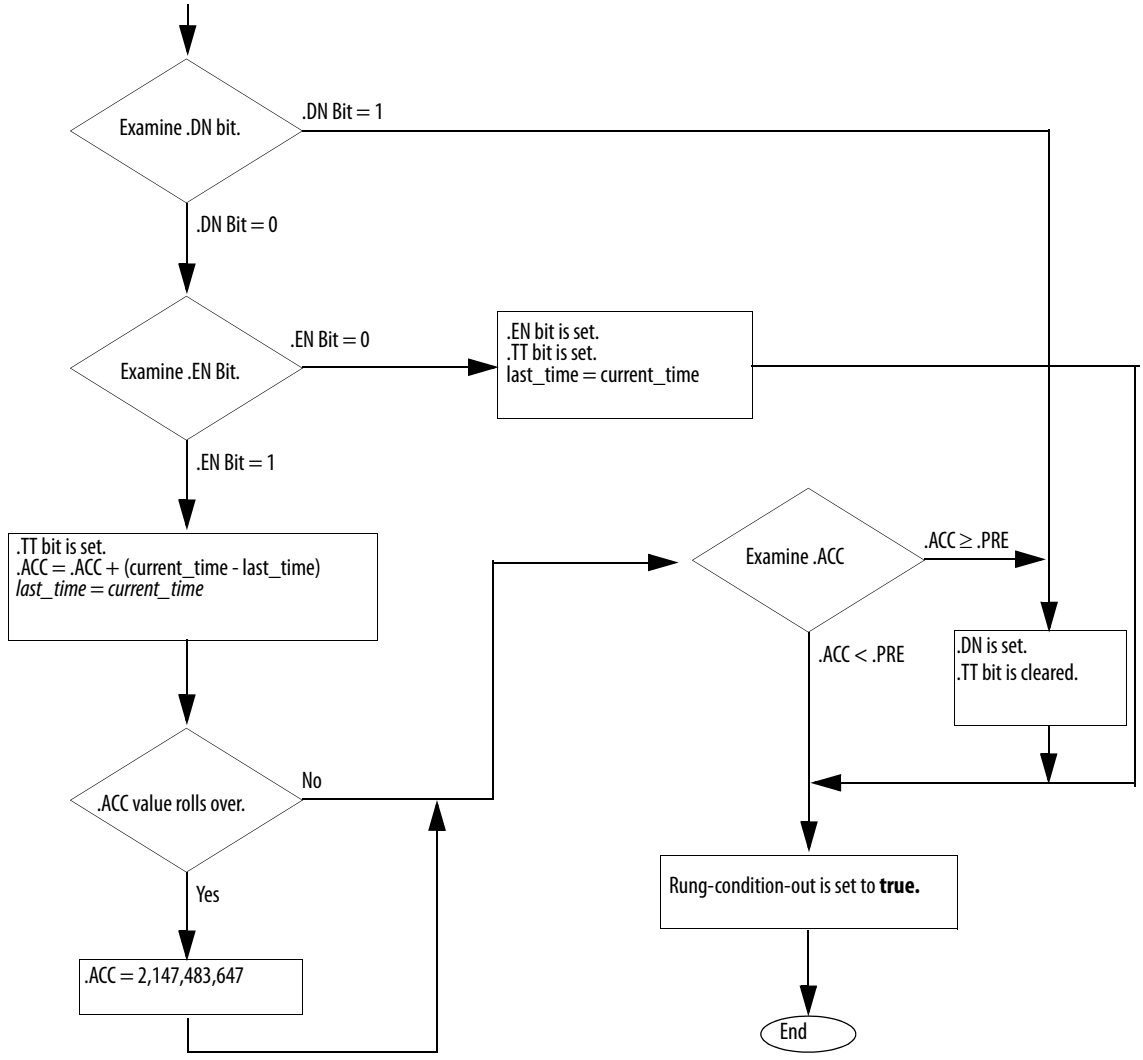
Arithmetic Status Flags:

Not affected

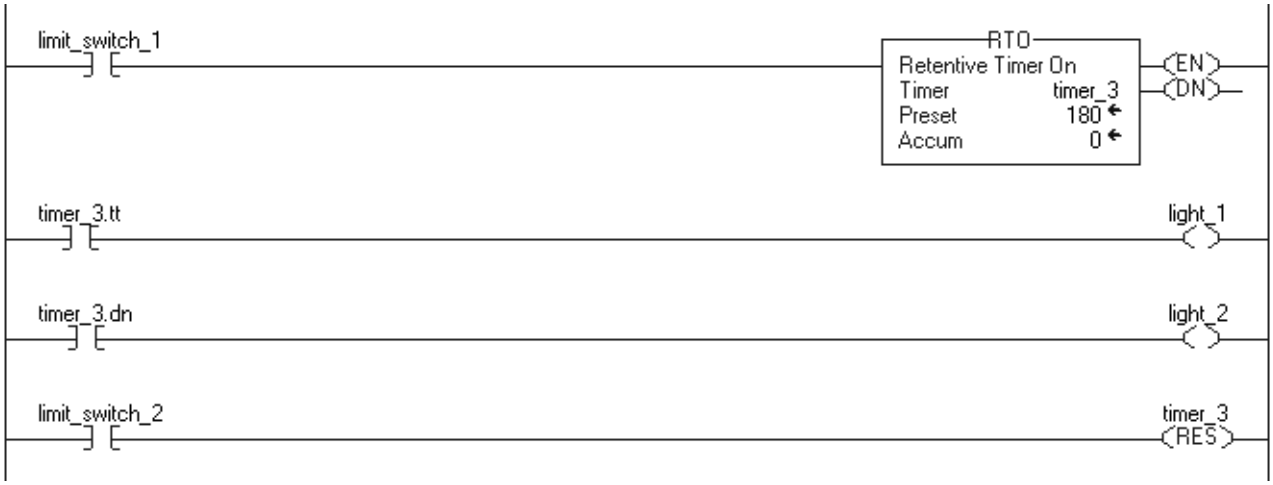
Fault Conditions:

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

Execution:

Condition	Relay Ladder Action
Prescan	The .EN, .TT, and .DN bits are cleared. The .ACC value is not modified. The rung-condition-out is set to false.
Rung-condition-in is false	The .EN and .TT bits are cleared. The .DN bit is not modified. The .ACC value is not modified. The rung-condition-out is set to false.
Rung-condition-in is true	
Postscan	The rung-condition-out is set to false.

Example: When *limit_switch_1* is set, *light_1* is on for 180 ms (t*imer_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 l*imit_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:

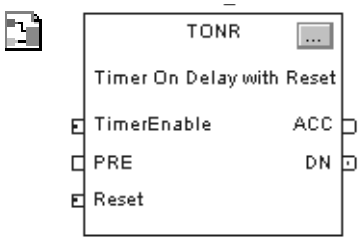
- TON (See page 116).
- RES (See page 152).

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

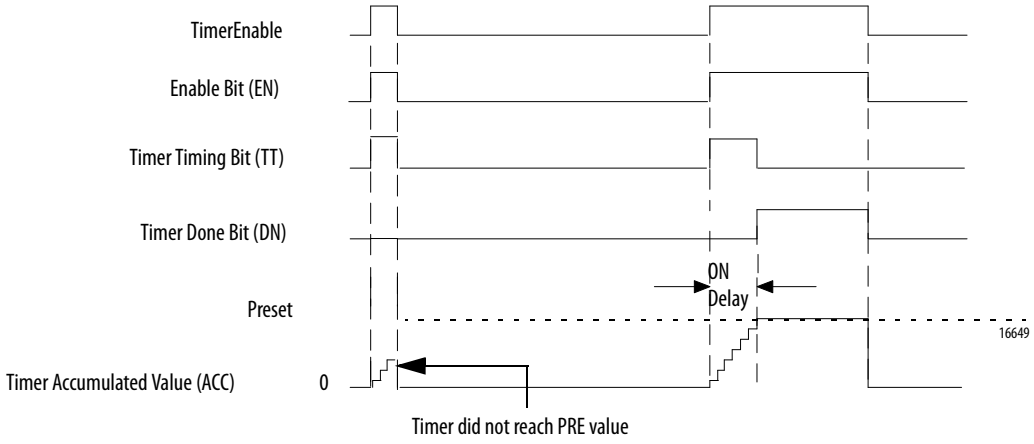
FBD_TIMER Structure

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 1ms 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 following occurs:

- TONR instruction is disabled
- ACC ≥ PRE

The time base is always 1 ms. For example, for a two-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.

A timer runs by subtracting the time of its last scan from the time now:

$$ACC = ACC + (current\_time - last\_time\_scanned)$$

After it updates the ACC, the timer sets last_time_scanned = current_time. This gets the timer ready for the next scan.

IMPORTANT

Make sure to scan the timer at least every 69 minutes while it runs. Otherwise, the ACC value won't be correct.

The *last_time_scanned* value has a range of up to 69 minutes. The timer's calculation rolls over if you don't scan the timer within 69 minutes. The ACC value won't be correct if this happens.

While a timer runs, scan it within 69 minutes if you put it in the following:

- A subroutine
- A section of code that is between JMP and LBL instructions
- A sequential function chart (SFC)
- An event or periodic task
- A state routine of a phase

Arithmetic Status Flags:

Not affected

Fault Conditions:

None

Execution:

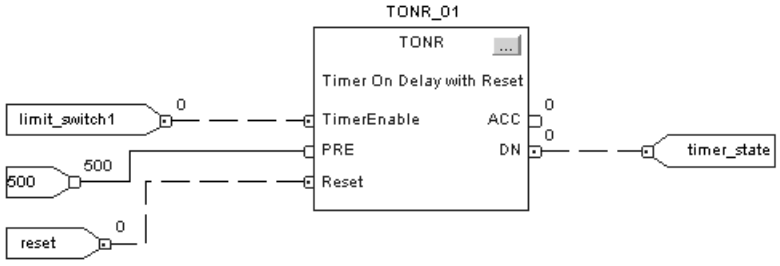
Condition	Function Block Action	Structured Text Action
Prescan	No action taken.	No action taken.
Instruction first scan	EN, TT and DN are cleared. ACC value is set to 0.	EN, TT and DN are cleared. ACC value is set to 0.
Instruction first run	EN, TT and DN are cleared. ACC value is set to 0.	EN, TT and 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.	N/A
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 ≥ PRE, the DN parameter is set, and timer_state is set.

Structured Text

```
TONR_01.Preset := 500;

TONR_01.Reset : = reset;

TONR_O1.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:

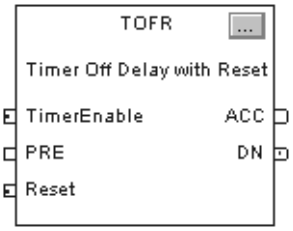
- TOF (See page 116).
- RES (See page 152).

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

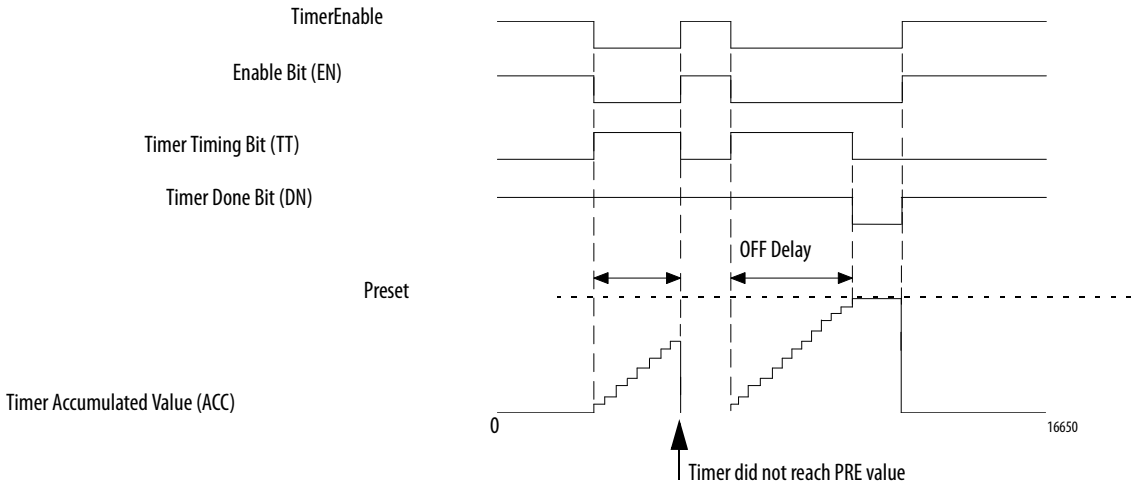
FBD_TIMER Structure

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 1ms 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 following occurs:

- TOFR instruction is disabled
- ACC ≥ PRE

The time base is always 1 ms. For example, for a two-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.

A timer runs by subtracting the time of its last scan from the time now:

$$ACC = ACC + (current\_time - last\_time\_scanned)$$

After it updates the ACC, the timer sets ***last_time_scanned = current_time***. This gets the timer ready for the next scan.

IMPORTANT

Make sure to scan the timer at least every 69 minutes while it runs. Otherwise, the ACC value won't be correct.

The *last_time_scanned* value has a range of up to 69 minutes. The timer's calculation rolls over if you don't scan the timer within 69 minutes. The ACC value won't be correct if this happens.

While a timer runs, scan it within 69 minutes if you put it in the following:

- A subroutine
- A section of code that is between JMP and LBL instructions
- A sequential function chart (SFC)
- An event or periodic task
- A state routine of a phase

Arithmetic Status Flags:

Not affected

Fault Conditions:

None

Execution:

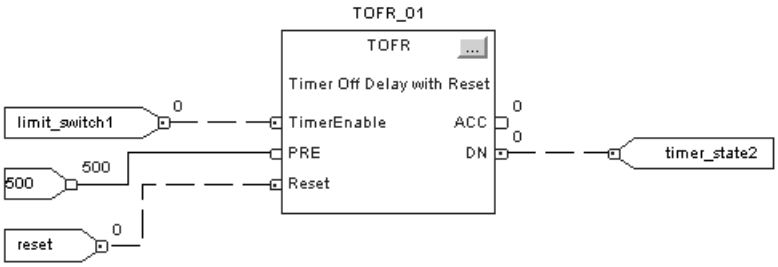
Condition	Function Block Action	Structured Text Action
Prescan	No action taken.	No action taken.
Instruction first scan	EN, TT and DN are cleared. ACC value is set to PRE.	EN, TT and DN are cleared. ACC value is set to PRE.
Instruction first run	EN, TT and DN are cleared. ACC value is set to PRE.	EN, 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.	N/A
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 ≥ PRE, the DN parameter is cleared, and *timer_state2* is set.

Structured Text

```
TOFR_01.Preset := 500

TOFR_01.Reset := reset;

TOFR_O1.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.

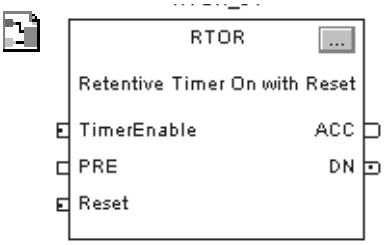
This instruction is available in relay ladder as two separate instructions:
- RTO (See page 124).
- RES (See page 152).

Operands:

RTOR(RTOR_tag);

Structured Text

Variable	Type	Format	Description
RTOR tag	FBD_TIMER	Structure	RTOR structure



Function Block Operands

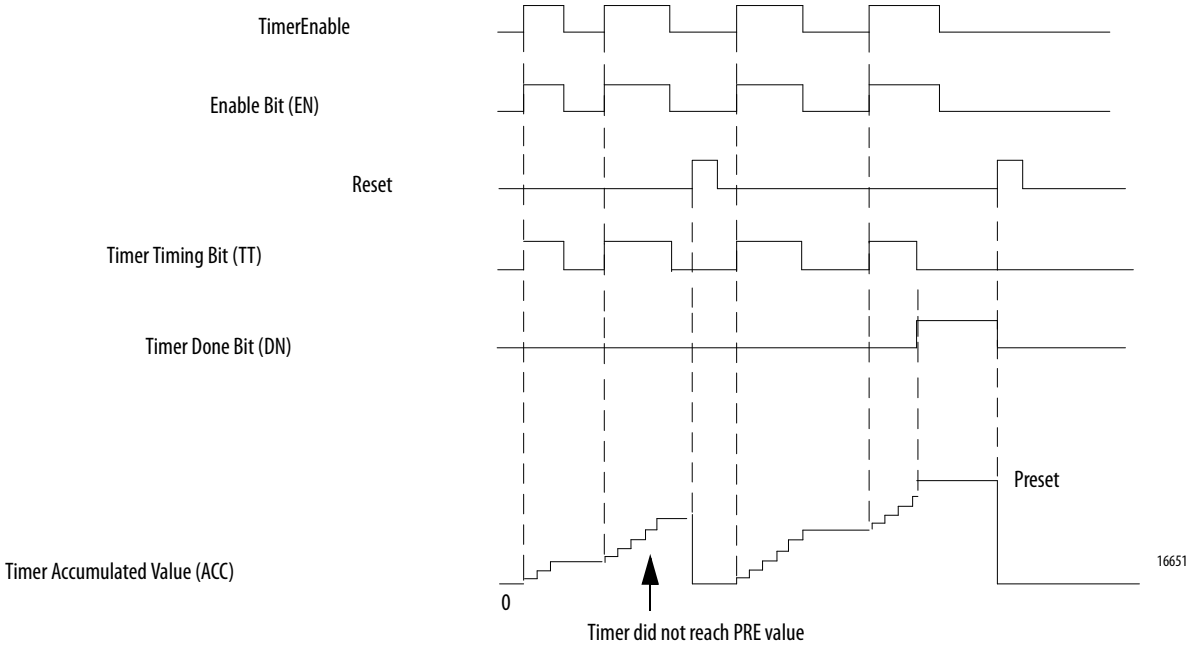
Operand	Type	Format	Description
RTOR tag	FBD_TIMER	Structure	RTOR structure

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 1ms 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 by using the Reset input.

The time base is always 1 ms. For example, for a two-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.

A timer runs by subtracting the time of its last scan from the time now:

$$ACC = ACC + (current\_time - last\_time\_scanned)$$

After it updates the ACC, the timer sets last_time_scanned = current_time. This gets the timer ready for the next scan.

IMPORTANT

Make sure to scan the timer at least every 69 minutes while it runs. Otherwise, the ACC value won't be correct.

The *last_time_scanned* value has a range of up to 69 minutes. The timer's calculation rolls over if you don't scan the timer within 69 minutes. The ACC value won't be correct if this happens.

While a timer runs, scan it within 69 minutes if you put it in a:

- Subroutine
- Section of code that is between JMP and LBL instructions
- Sequential function chart (SFC)
- Event or periodic task
- State routine of a phase

Arithmetic Status Flags:

Not affected

Fault Conditions:

None

Execution:

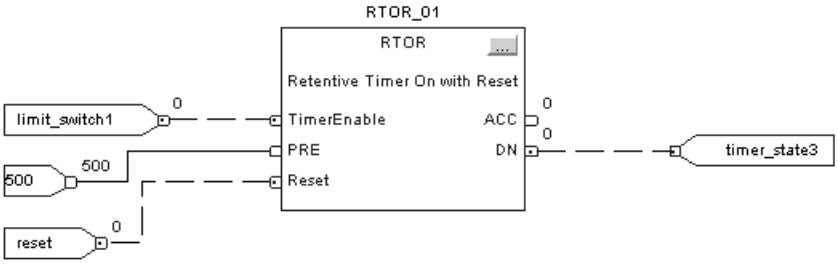
Condition	Function Block Action	Structured Text Action
Prescan	No action taken.	No action taken.
Instruction first scan	EN, TT and DN are cleared. ACC value is not modified.	EN, TT and DN are cleared. ACC value is not modified.
Instruction first run	EN, TT and DN are cleared. ACC value is not modified.	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.	N/A
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 ≥ PRE, the DN parameter is set, and *timer_state3* is set.

Structured Text

```
RTOR_01.Preset := 500

RTOR_01.Reset := reset;

RTOR_O1.TimerEnable := limit_switch1;


RTOR(RTOR_01);


timer_state3 := RTOR_01.DN;
```

Function Block



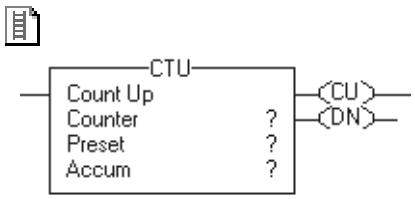
Count Up (CTU)

The CTU instruction counts upward.

This instruction is available in structured text and function block as CTUD.

Operands:

Relay Ladder



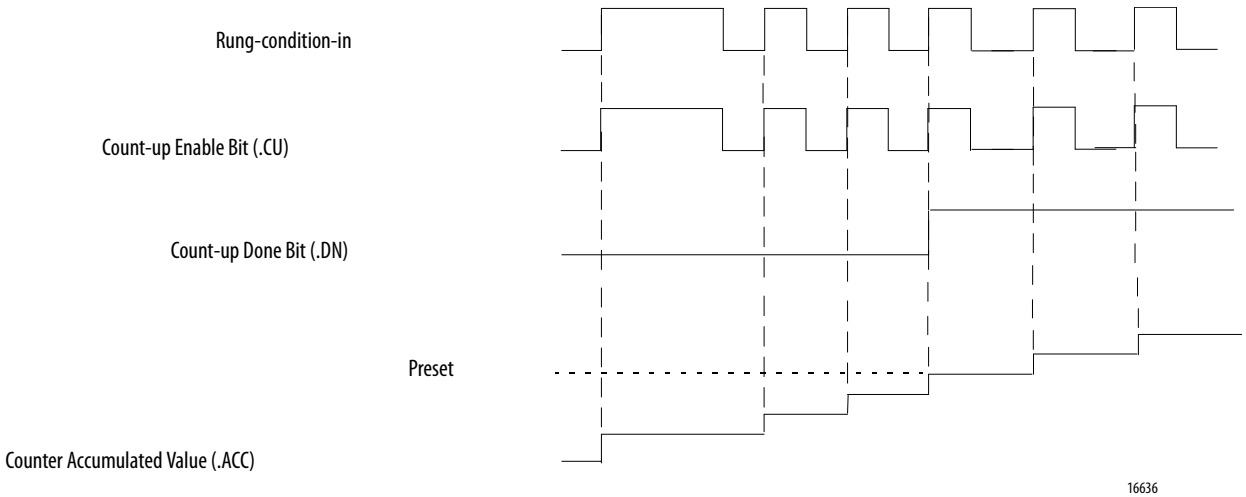
Operand	Type	Format	Description
Counter	COUNTER	Tag	Counter structure
Preset	DINT	Immediate	How high to count
Accum	DINT	Immediate	Number of times the counter has counted Initial value is typically 0

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 ≥ .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 that 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.



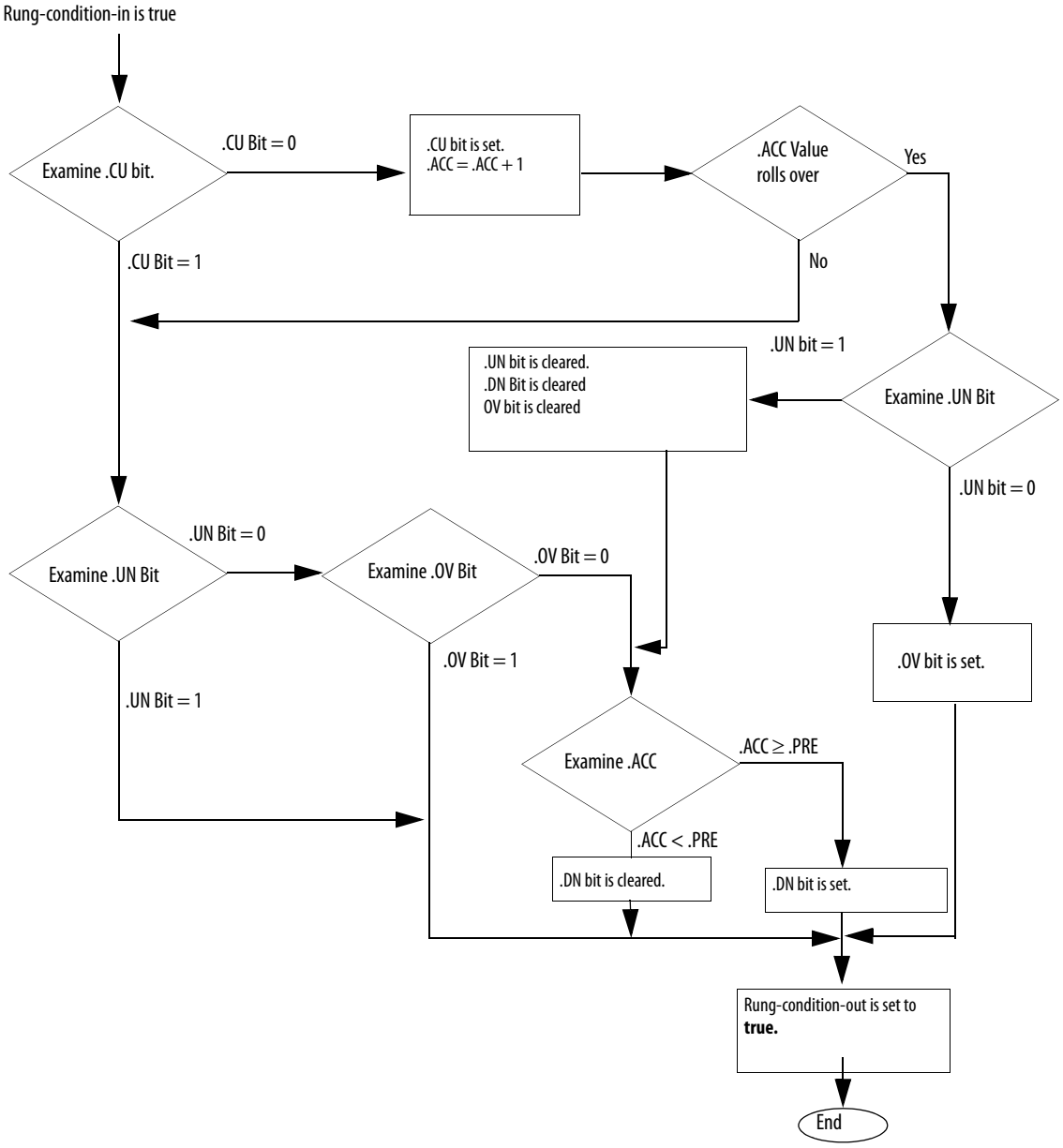
The accumulated value continues incrementing, even after the .DN bit is set. To clear the accumulated value, use a RES instruction that references the counter structure or write 0 to the accumulated value.

Arithmetic Status Flags: Not affected

Fault Conditions: None

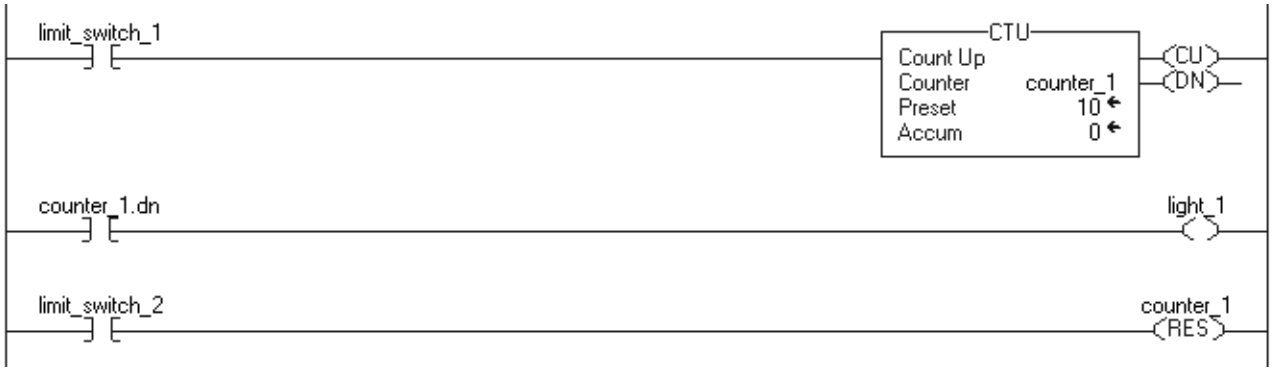
Execution:

Condition	Relay Ladder Action
Prescan	The .CU bit is set to prevent invalid increments during the first program scan. The rung-condition-out is set to false.
Rung-condition-in is false	The .CU bit is cleared. The rung-condition-out is set to false.



Postscan	The rung-condition-out is set to false.

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.

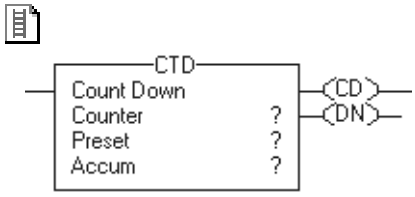


Count Down (CTD)

The CTD instruction counts downward.

This instruction is available in structured text and function block as CTUD.

Operands:



Relay Ladder

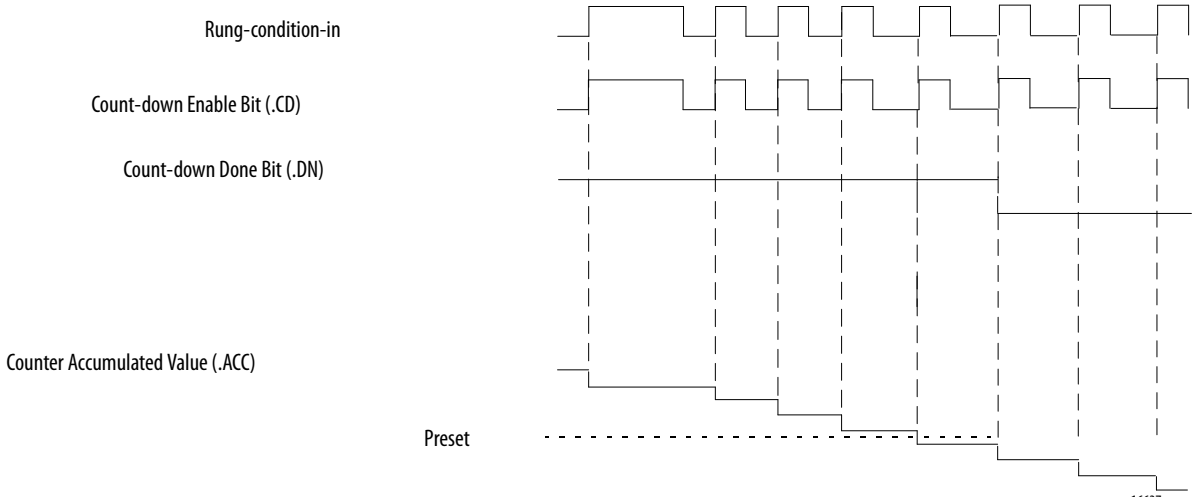
Operand	Type	Format	Description
Counter	COUNTER	Tag	Counter structure
Preset	DINT	Immediate	How low to count
Accum	DINT	Immediate	Number of times the counter has counted Initial value is typically 0

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 ≥ .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 that 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.



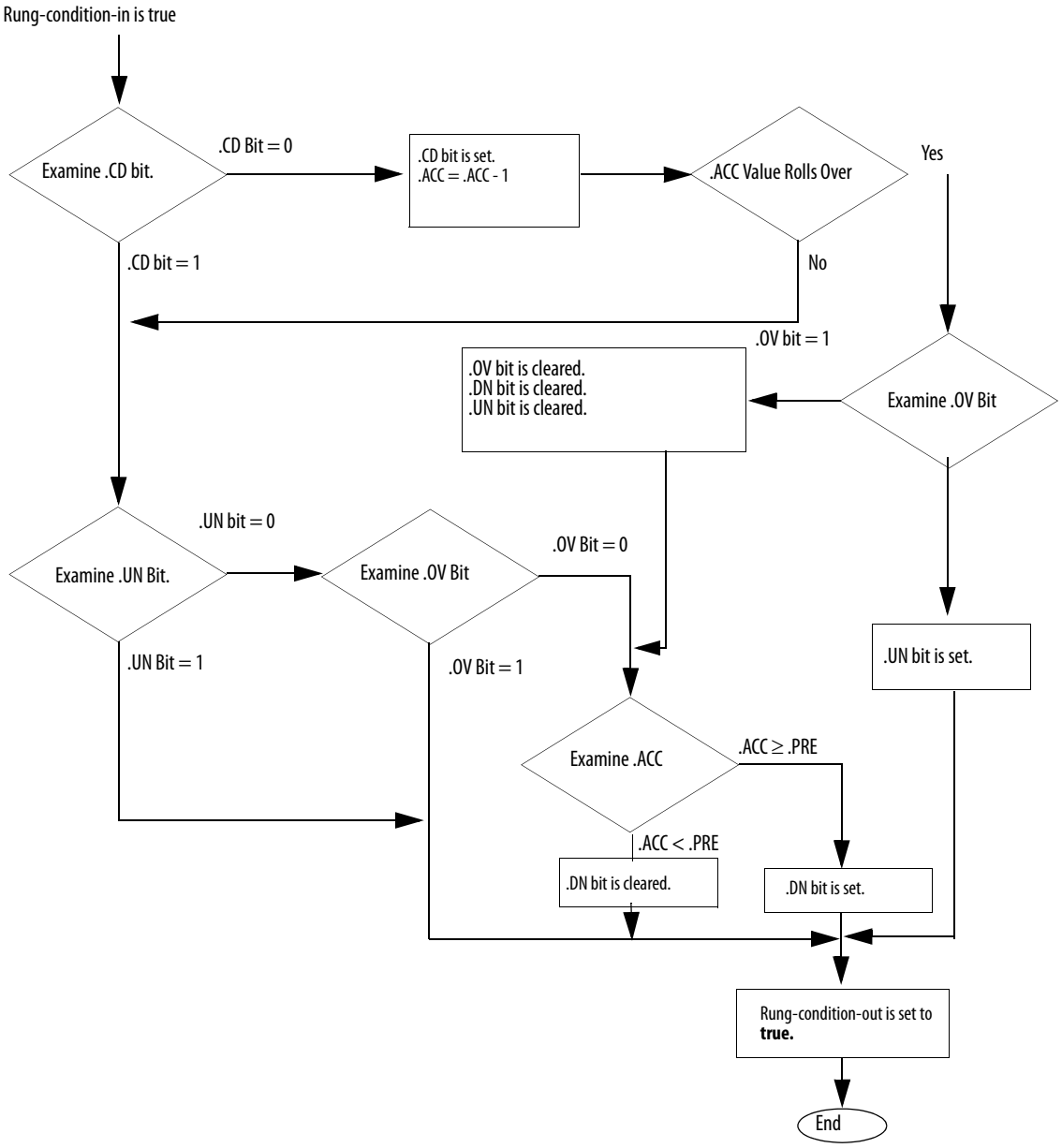
The accumulated value continues decrementing, even after the .DN bit is set. To clear the accumulated value, use a RES instruction that references the counter structure or write 0 to the accumulated value.

Arithmetic Status Flags: Not affected

Fault Conditions: None

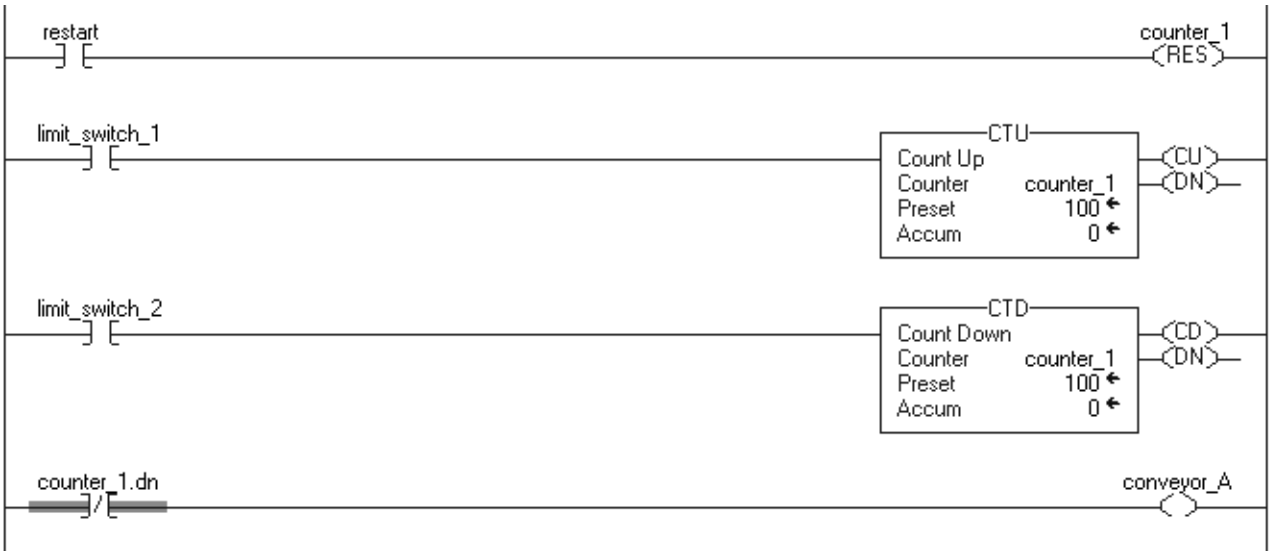
Execution:

Condition	Relay Ladder Action
Prescan	The .CD bit is set to prevent invalid decrements during the first program scan. The rung-condition-out is set to false.
Rung-condition-in is false	The .CD bit is cleared. The rung-condition-out is set to false.



Postscan	The rung-condition-out is set to false.

Example: A conveyor brings parts into a buffer zone. Each time a part enters, *limit_switch_1* is enabled and *counter_1* increments by one. Each time a part leaves, *limit_switch_2* is enabled and *counter_1* decrements by one. 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 140)
- CTD (See page 144)
- RES (See page 152)

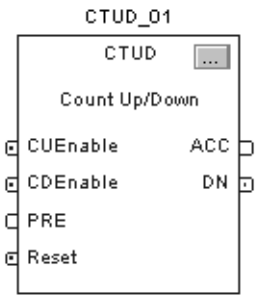
Operands:

Structured Text

CTUD(CTUD_tag);

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.

Figure 2 - Counting Up

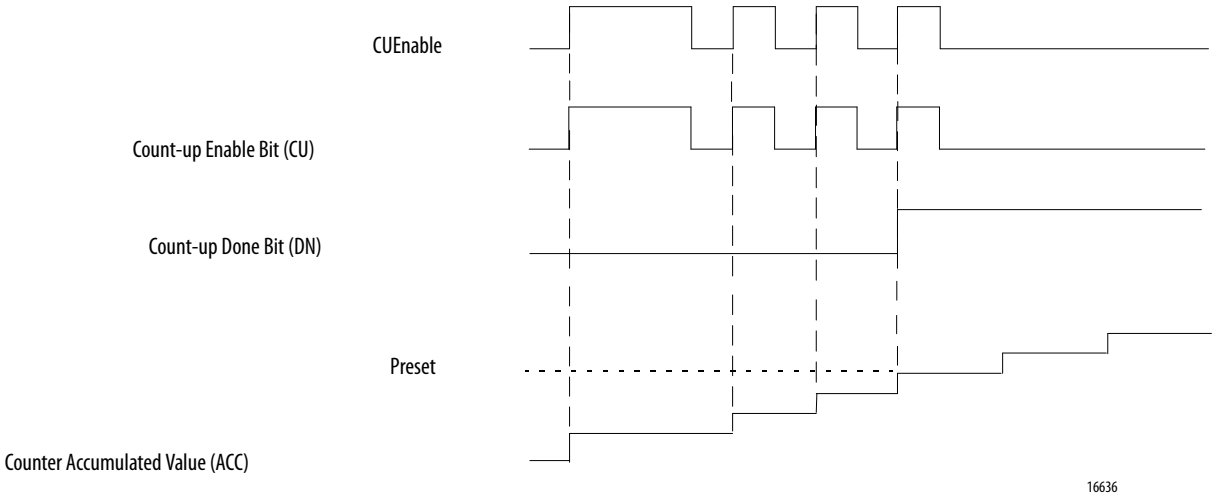
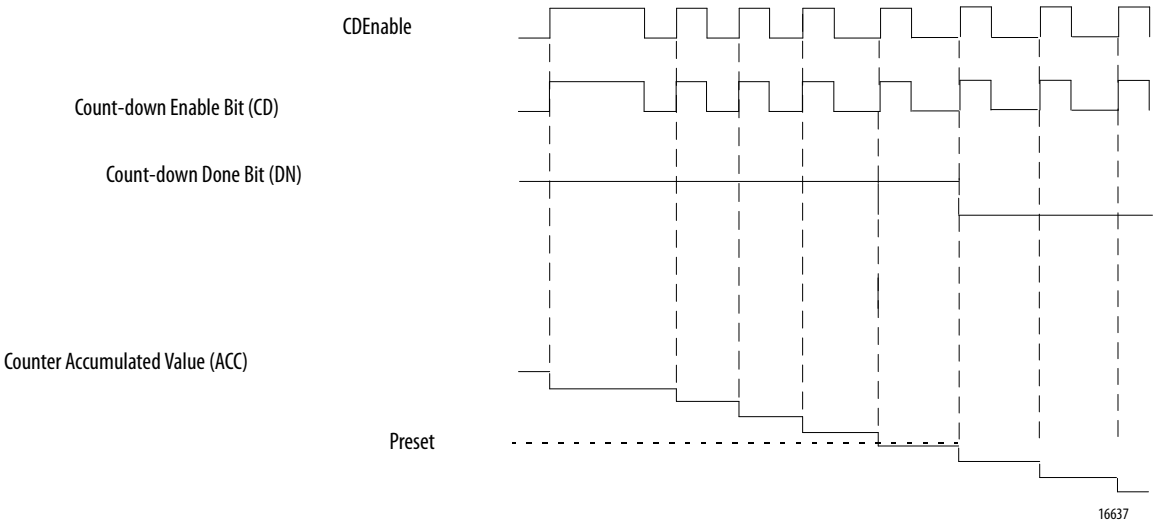


Figure 3 - Counting Down



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:

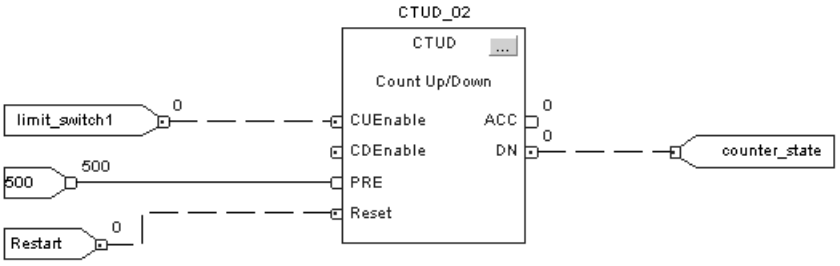
Condition	Function Block Action	Structured Text Action
Prescan	No initialization required.	No initialization required.
Instruction first scan	$CUEnable_{n-1}$ and $CDEnable_{n-1}$ are set.	$CUEnable_{n-1}$ and $CDEnable_{n-1}$ are set.
Instruction first run	$CUEnable_{n-1}$ and $CDEnable_{n-1}$ are set.	$CUEnable_{n-1}$ and $CDEnable_{n-1}$ are set.
EnableIn is cleared	EnableOut is cleared, the instruction does nothing, and the outputs are not updated.	N/A
EnableIn is set	The instruction sets $CUEnable_{n-1}$ and $CDEnable_{n-1}$. On a cleared to set transition of EnableIn: • 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 ≥ 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_O1.CUEnable := limit_switch1;


CTUD(CTUD_01);


counter_state := CTUD_01.DN;
```

Function Block

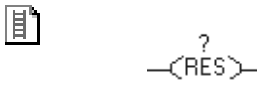


Reset (RES)

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.

Input/Output Instructions

(MSG, GSV, SSV, IOT)

The input/output instructions read or write data to, or from, a controller or module.

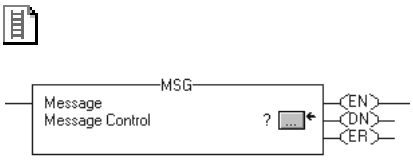
If you want to	Use this instruction	Available in these languages	Page
Send data to or from another module	MSG	Relay ladder Structured text	154
Get controller status information	GSV	Relay ladder Structured text	188
Set controller status information	SSV	Relay ladder Structured text	188
Send output values to an I/O module or consuming controller at a specific point in your logic. Trigger an event task in another controller.	IOT	Relay ladder Structured text	209

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

Structured Text

```
MSG(MessageControl);
```

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

MESSAGE Structure

TIP We recommend that you always include an XIO of the MSG.EN bit as an in-series MSG rung precondition. This guarantees that other ring preconditions cannot trigger the rung until the MSG completes or errors.

IMPORTANT If modifying the default values for the UnconnectedTimeout or ConnectionRate, do not enter values lower than 1 second. If values lower than 1 second are used, it may cause the controller to do the following:

- Drop off the backplane and require the controller to be power cycled to recover communications
- Experience a unrecoverable major fault

ATTENTION: The controller changes the DN, ER, EW, and ST bits asynchronous to the scan of your logic. If you check these status bits more than once in your program, use a copy of these bits and examine the state of the copied values, otherwise your logic may not work as expected.

One way to make a copy is to use the FLAGS word. Copy the FLAGS word to another tag and check the bits in the copy.

IMPORTANT Do not change or manipulate the following status bits of a MSG instruction, as this may cause an unrecoverable fault to occur:

- DN
- EN
- ER
- EW
- ST

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: Do not change or manipulate the EN, EW, ER, DN, or ST bits of the FLAGS member. For example, do not clear the entire FLAGS word. The controller ignores the change and uses the internally-stored values of the bits.	
.ERR	INT	If the .ER bit is set, the error code word identifies error codes for the MSG instruction. Manipulating these bits can cause an unrecoverable major fault in the controller. See MSG Error Codes on page 161.	
.EXERR	INT	The extended error code word specifies additional error code information for some error codes. See Extended Error Codes on page 162.	
.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. **Important:** Do not change or manipulate the EW bit. The controller ignores the change and uses the internally-stored value of the bit. Manipulating these bits can cause an unrecoverable major fault in the controller.	
.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. **Important:** Do not change or manipulate the ER bit. Manipulating these bits can cause an unrecoverable major fault in the controller.	
.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. **Important:** Do not change or manipulate the DN bit. Manipulating these bits can cause an unrecoverable major fault in the controller.	
.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. **Important:** Do not change or manipulate the ST bit. The controller ignores the change and uses the internally-stored value of the bit. Manipulating these bits can cause an unrecoverable major fault in the controller.	
.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. **Important:** Do not change or manipulate the EN bit. Manipulating these bits can cause an unrecoverable major fault in the controller.	
.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 186 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 Logix Designer application 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.	

Mnemonic	Data Type	Description	
.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**
		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**
		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. • 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 '*Go to Message Path Editor*'.	
.RemoteIndex	DINT	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.	
		If the message	**Then the remote array is**
		Reads data	Source element
		Writes data	Destination element
.RemoteElement	STRING	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.	
		If the message	**Then the remote array is**
		Reads data	Source element
		Writes data	Destination element
.UnconnnectedTimeout	DINT	Time out for an unconnected message or for making a connection. The default value is 30 seconds.	
		If the message is	**Then**
		Unconnected	The ER bit turns on if the controller doesn't get a response within the UnconnectedTimeout time.
		Connected	The ER bit turns on if the controller doesn't get a response for making the connection within the UnconnectedTimeout time.
.ConnectionRate	DINT	Time out for a connected message once it has a connection. This time out is for the response from the other device about the sending of the data. • This time out applies only after the connection is made. • The time out = ConnectionRate x TimeoutMultiplier. • The default ConnectionRate is 7.5 seconds. • The default TimeoutMultiplier is 0 (which is 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		

Description

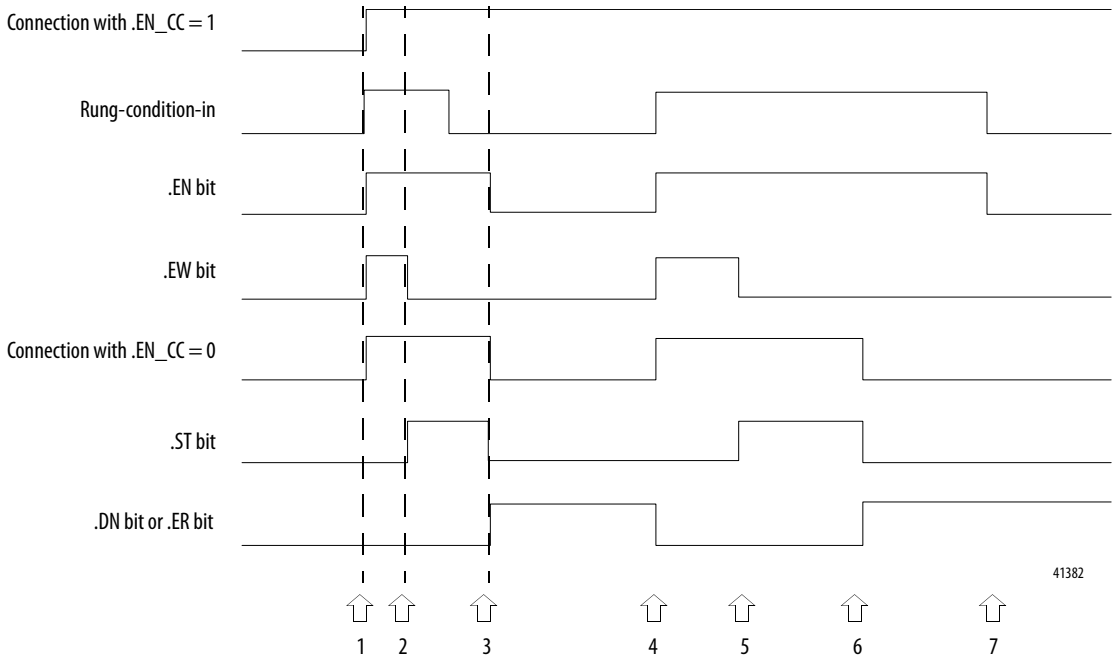
The MSG instruction transfers elements of data.

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 executes only on a transition.

See Appendix B.

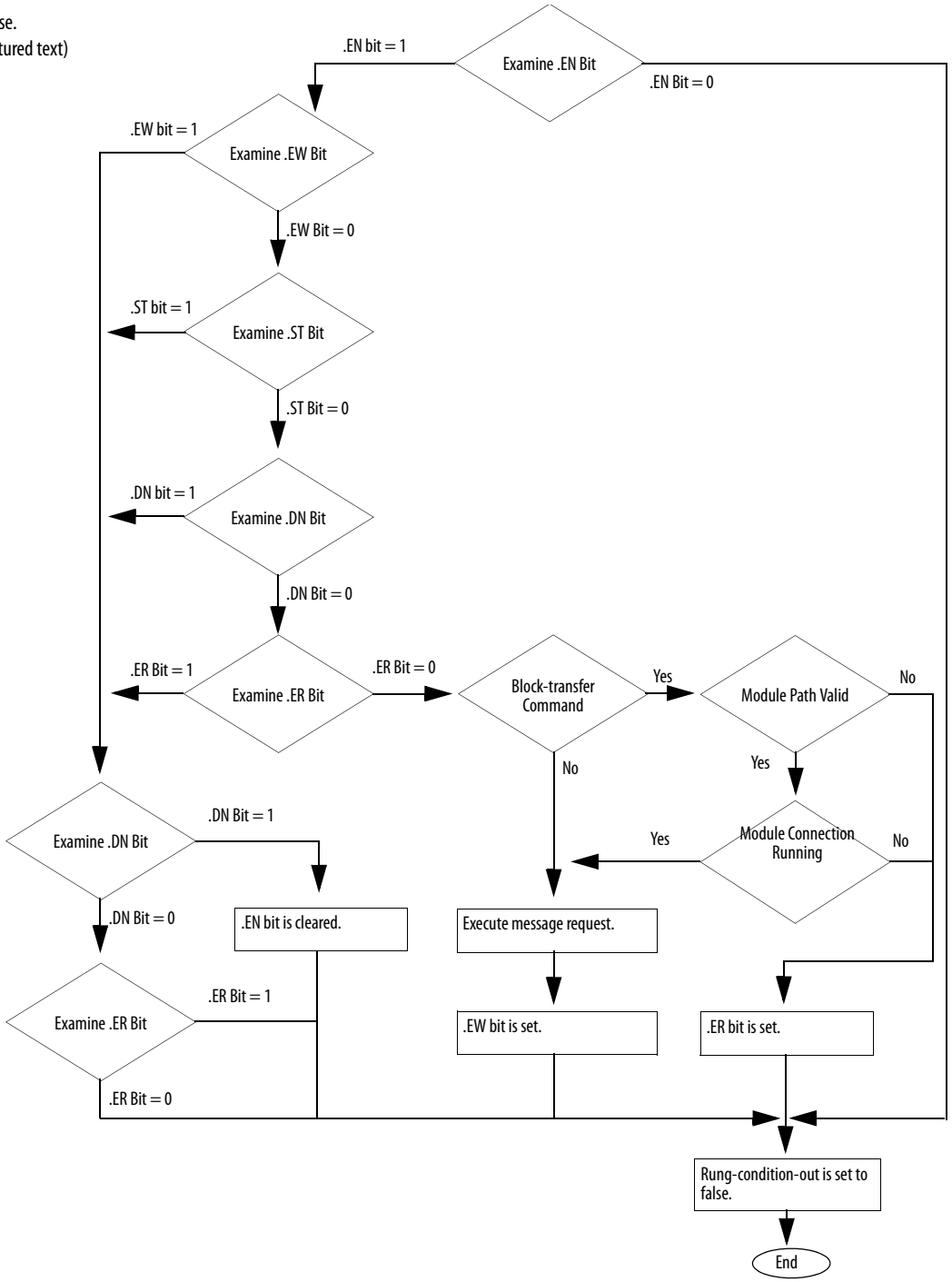
The size of each element depends on the data types you specify and the type of message command you use.



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 setCconnection is opened*. .DN or .ER is cleared.	N/A	N/A

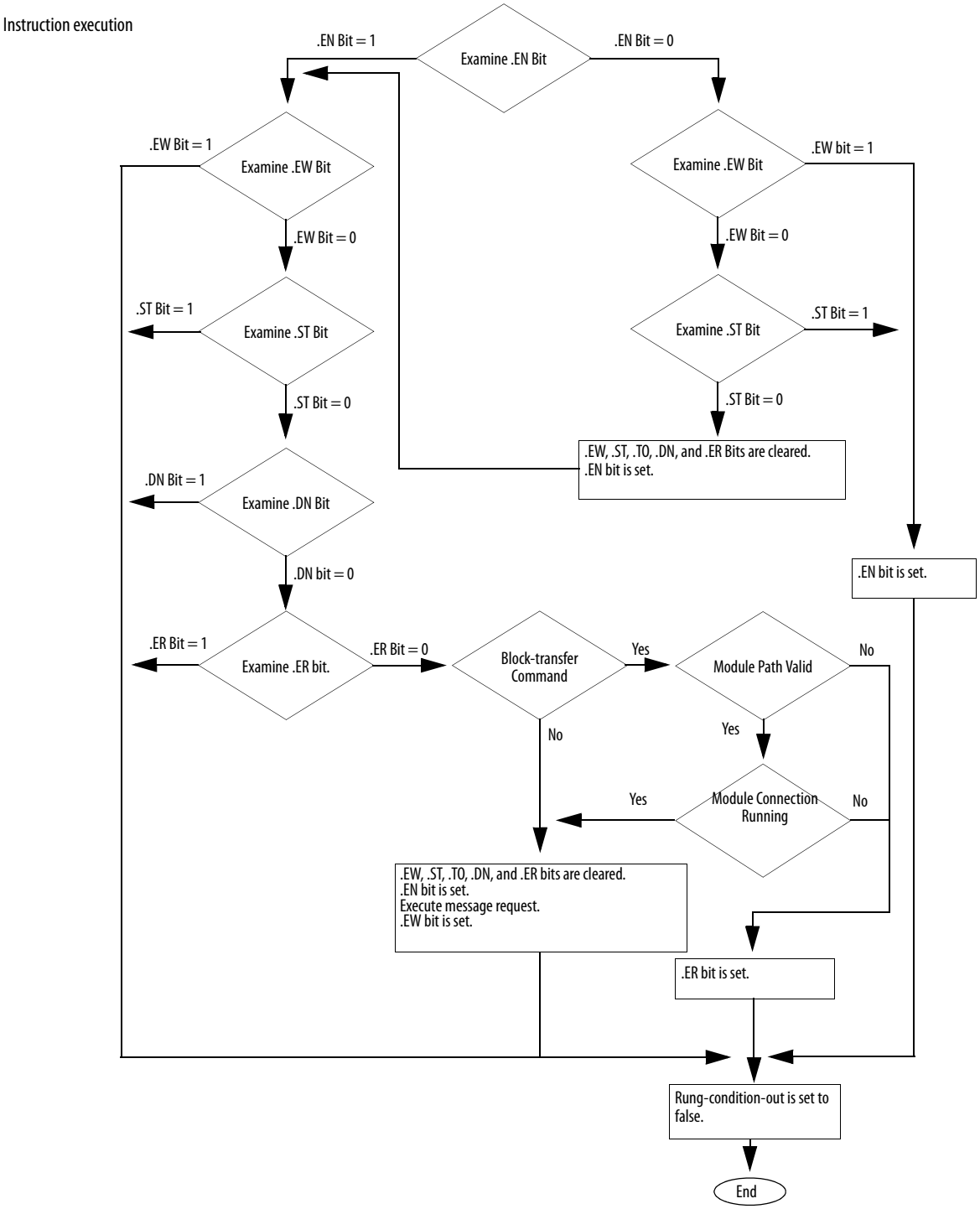
Execution:

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

Condition	Relay Ladder Action	Structured Text Action
Rung-condition-in is false. (does not apply to structured text)		
Rung-condition-in is true.	The instruction executes. The rung-condition-out is set to true.	N/A

Condition	Relay Ladder Action	Structured Text Action
EnableIn is set.	N/A	EnableIn is always set. The instruction executes.

Instruction execution



Condition	Relay Ladder Action	Structured Text Action
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

The Logix Designer application 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	
0003	Invalid value	
0004	IOI syntax error (see extended error codes)	
0005	Destination unknown, class unsupported, instance undefined or structure element undefined (see extended error codes)	
0006	Insufficient packet space	
0007	Connection lost	
0008	Service unsupported	
0009	Error in data segment or invalid attribute value	
000A	Attribute list error	
000B	State already exists	
000C	Object model conflict	
000D	Object already exists	
000E	Attribute cannot be set	
000F	Permission denied	
0010	Device state conflict	
0011	Reply will not fit	
0012	Fragment primitive	
0013	Insufficient command data	
0014	Attribute not supported	
0015	Too much data	
001A	Bridge request too large	
001B	Bridge response too large	
001C	Attribute list shortage	

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

Extended Error Codes

The Logix Designer application 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
0100	Connection in use
0103	Transport not supported
0106	Ownership conflict
0107	Connection not found
0108	Invalid connection type
0109	Invalid connection size
0110	Module not configured
0111	EPR not supported
0114	Wrong module
0115	Wrong device type
0116	Wrong revision
0118	Invalid configuration format
011A	Application out of connections
0203	Connection timeout
0204	Unconnected message timeout

Extended Error Code (Hex)	Description
0205	Unconnected send parameter error
0206	Message too large
0301	No buffer memory
0302	Bandwidth not available
0303	No screeners available
0305	Signature match
0311	Port not available
0312	Link address not available
0315	Invalid segment type
0317	Connection not scheduled

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
2001	Excessive IOI
2002	Bad parameter value
2018	Semaphore reject
201B	Size too small
201C	Invalid size
2100	Privilege failure
2101	Invalid keyswitch position
2102	Password invalid
2103	No password issued
2104	Address out of range
2105	Address and how many out of range
2106	Data in use
2107	Type is invalid or not supported
2108	Controller in upload or download mode

Extended Error Code (Hex)	Description
2109	Attempt to change number of array dimensions
210A	Invalid symbol name
210B	Symbol does not exist
210E	Search failed
210F	Task cannot start
2110	Unable to write
2111	Unable to read
2112	Shared routine not editable
2113	Controller in faulted mode
2114	Run mode inhibited

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 - 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
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

Table 3 - PLC and SLC Error Codes (hex) (Continued)

R9.x And Earlier		R10.x And Later		Description
.ERR	.EXERR	.ERR	.EXERR	
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)

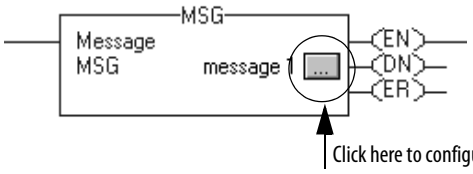
Block-transfer Error Codes

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

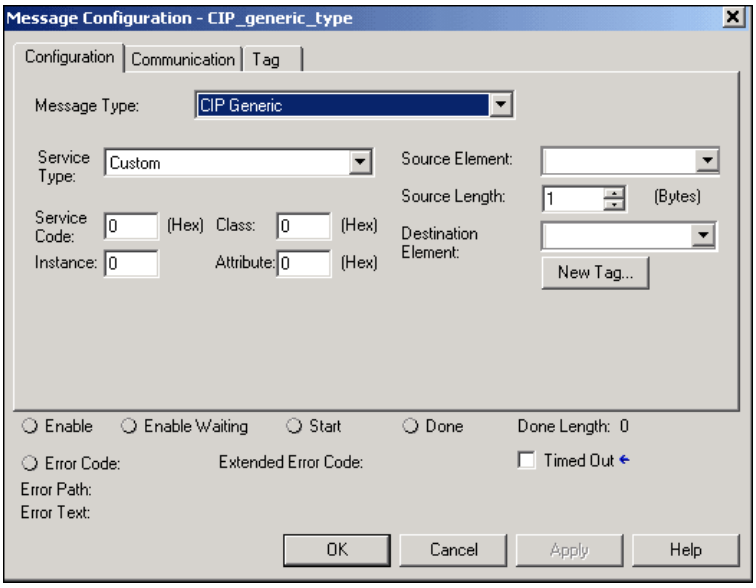
Specify the Configuration Details

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.

Figure 4 - Message Configuration Tab



45634

If the target device is	Select one of these message types	Page
Logix5000 controller	CIP Data Table Read	168
	CIP Data Table Write	
I/O module that you configure by using Logix Designer application	Module Reconfigure	169
	CIP Generic	171
PLC-5 controller	PLC5 Typed Read	174
	PLC5 Typed Write	
	PLC5 Word Range Read	
	PLC5 Word Range Write	
SLC controller MicroLogix™ controller	SLC Typed Read	176
	SLC Typed Write	
Block-transfer module	Block-Transfer Read	176
	Block-Transfer Write	

PLC-3 processor	PLC3 typed read	177
	PLC3 typed write	
	PLC3 word range read	
	PLC3 word range write	
PLC-2® processor	PLC2 unprotected read	178
	PLC2 unprotected write	

Specify This Data for a Logix5000 Controller as a Target Device

Use this configuration information if a Logix5000 controller is the target device.

For this property	Specify
Source Element	• 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. 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. • The only source data type that is not supported is Boolean. All other data types—SINT, INT, DINT, LINT, REAL—can be used. Any structure type predefined, module-defined, or user-defined also can be used to send messages.
Source Length	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 timer1 is one element that consists of one timer control structure.
Destination Element	• 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.

Specify 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, the following occurs:

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

A Module Reconfigure message requires this configuration properties.

In this property	Select
Message Type	Module Reconfigure

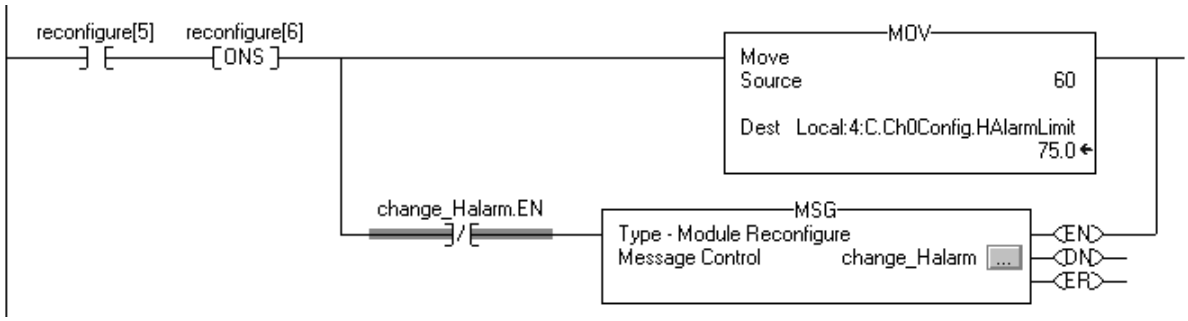
Example: Follow these steps to reconfigure an I/O module.

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.

TIP We recommend that you always include an XIO of the MSG.EN bit as an in-series MSG rung precondition.

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

IMPORTANT ControlLogix modules have services that can be invoked by using a MSG instruction and choosing the CIP Generic message type.

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	tag_name of type INT [5]	
		This array contains	**Description**
		tag_name[0]	Bit mask of points to test (test only one point at a time)
		tag_name[1]	Reserved, leave 0
		tag_name[2]	Pulse width (hundreds of µs, usually 20)
		tag_name[3]	Zero cross delay for ControlLogix I/O (hundreds of µs, usually 40)
		tag_name[4]	Verify delay
	Destination	Blank	
Get audit value	Message Type	CIP Generic	
	Service Type	Audit Value Get	
	Source Element	Cannot change this field, blank	
	Source Length	Cannot change this field, set to 0 bytes	
	Destination Element	**This array contains**	**Description**
		tag_name of type DINT[2] or LINT	This tag contains the Audit Value for the controller. **IMPORTANT:** We recommend using the DINT[2] data type to avoid limitations when working with LINT data types in Allen-Bradley® controllers.
Get controller events monitored for changes See Table 4	Message Type	CIP Generic	
	Service Type	Changes to Detect Get	
	Source Element	Cannot change this field, blank	
	Source Length	Cannot change this field, set to 0 bytes	
	Destination Element	**This array contains**	**Description**
		tag_name of type DINT[2] or LINT	This tag represents a bit mask of the changes monitored for the controller. **IMPORTANT:** We recommend using the DINT[2] data type to avoid limitations when working with LINT data types in Allen-Bradley controllers.

If you want to	In this property	Type or select	
Set controller events monitored for changes See Table 4	Message Type	CIP Generic	
	Service Type	Changes to Detect Set	
	Source Element	**This array contains**	**Description**
		tag_name of type DINT[2] or LINT	This tag represents a bit mask of the changes monitored for the controller. **IMPORTANT:** We recommend using the DINT[2] data type to avoid limitations when working with LINT data types in Allen-Bradley controllers.
	Source Length	Cannot change this field, set to 8 bytes	
	Destination Element	Cannot change this field, blank	
Reset electronic fuses on a digital output module	Message Type	CIP Generic	
	Service Type	Reset Electronic Fuse	
	Source	tag name 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	tag_name 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	tag_name 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. • 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	
Unlatch the alarm of an analog output module	Message Type	CIP Generic	
	Service Type	Select which alarm that you want to unlatch. • Unlatch All Alarms (O) • Unlatch High Alarm (O) • Unlatch Low Alarm (O) • Unlatch Ramp Alarm (O)	
	Instance	Channel of the alarm that you want to unlatch	

Table 4 - Get/Set Controller Events Monitored for Changes Bit Definitions

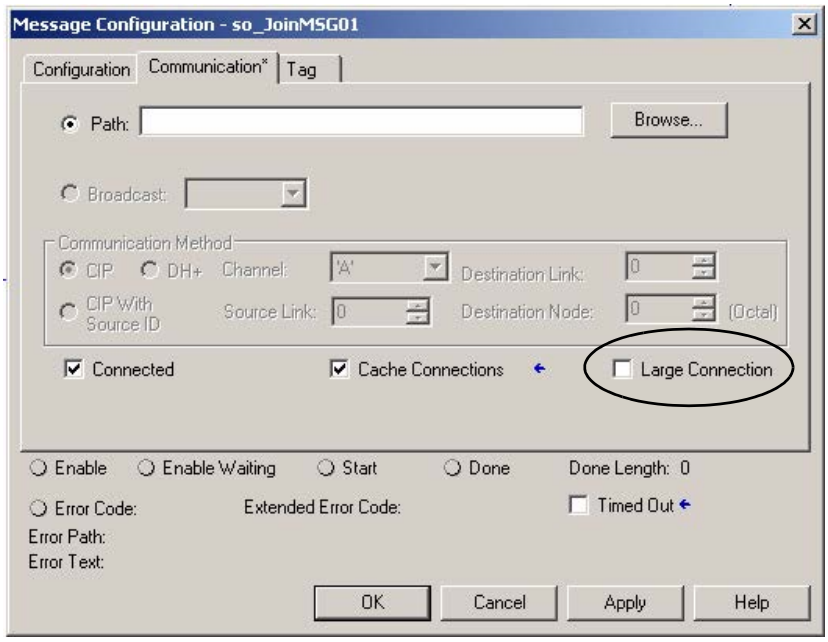
Tag Names	Data Type	Bit Definition	
Get Controller Events Monitored for Changes Set Controller Events Monitored for Changes	DINT[0]	**Bit**	**Meaning**
		0	Store to removable media through Logix Designer application
		1	Online edits were accepted, tested, or assembled
		2	Partial import online transaction completed
		3	SFC Forces were enabled
		4	SFC Forces were disabled
		5	SFC Forces were removed
		6	SFC Forces were modified
		7	I/O Forces were enabled
		8	I/O Forces were disabled
		9	I/O Forces were removed
		10	I/O Forces were changed
		11	Firmware update from unconnected source
		12	Firmware update via removable media
		13	Mode change via workstation
		14	Mode change via mode switch
		15	A major fault occurred
		16	Major faults were cleared
		17	Major faults were cleared via mode switch
		18	Task properties were modified
		19	Program properties were modified
		20	Controller timeslice options were modified
		21	Removable media was removed
		22	Removable media was inserted
		23	Safety signature created
		24	Safety signature deleted
		25	Safety lock
		26	Safety unlock
		27	Constant tag value changed
		28	Constant tag multiple values changed
		29	Constant tag attribute cleared
		30	Tag set as constant
		31	Custom log entry added
	DINT[1]	32	Change that affects correlation
		33	Protect signature in Run mode attribute set
		34	Protect signature in Run mode attribute cleared
		35…63	Unused

IMPORTANT

Selecting the CIP Generic message type enables the Large Connection option on the Communication tab. Use large CIP Generic connections when a message is greater than 480 bytes. (500 bytes is typical, but note that there are headers at the front of the message.) Large CIP connections can be used for messages up to 3980 bytes.

The Large Connection checkbox is enabled only when the Connected box is checked and CIP Generic is selected as the message type on the Configuration tab (see Figure 4 on page 167).

The Large Connection option is available only in Logix Designer application, version 21.00.00 or later and RSLogix 5000 software, version 20.00.00 or later.



Specify 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 Data types for PLC5 Typed Read and Typed Write messages on page 175.
PLC5 Typed Write	Write 16-bit integer, floating-point, or string type data and maintain data integrity. See Data types for PLC5 Typed Read and Typed Write messages on page 175
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.

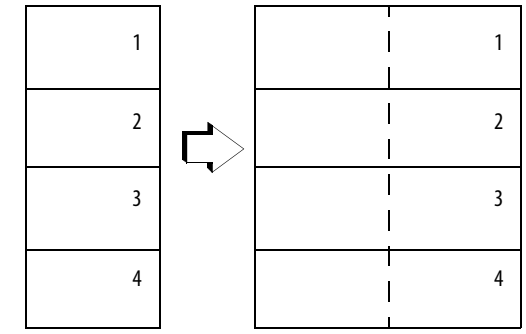
Table 5 - Data types for PLC5 Typed Read and Typed Write messages

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 ≥ −32,768 and ≤ 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.

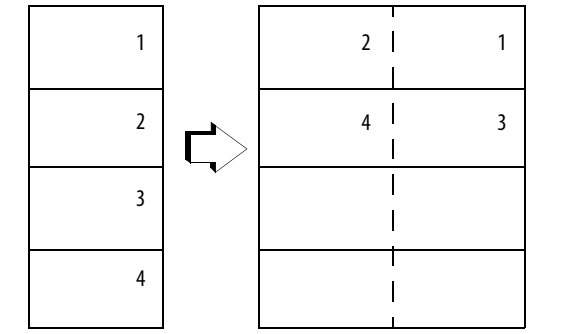
The following diagrams show how the typed and word-range commands differ. The example uses read commands from a PLC-5 processor to a Logix5000 controller.

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.

Specify 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...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)	

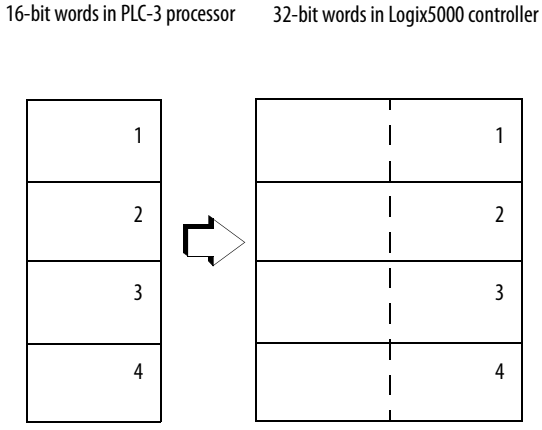
Specify PLC-3 Messages

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

Select this command	If you want to
PLC3 Typed Read	Read integer or REAL type data. 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. This command also reads floating-point data from the PLC-3 and stores it in a REAL data type tag in the Logix5000 controller.
PLC3 Typed Write	Write integer or REAL type data. 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 ($-32{,}768 \geq$ data $\leq 32{,}767$). This command also writes REAL type data from the Logix5000 controller to a PLC-3 floating-point file.
PLC3 Word Range Read	Read a contiguous range of 16-bit words in PLC-3 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.
PLC3 Word Range Write	Write a contiguous range of 16-bit words from Logix5000 memory regardless of data type to PLC-3 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-3 processor.

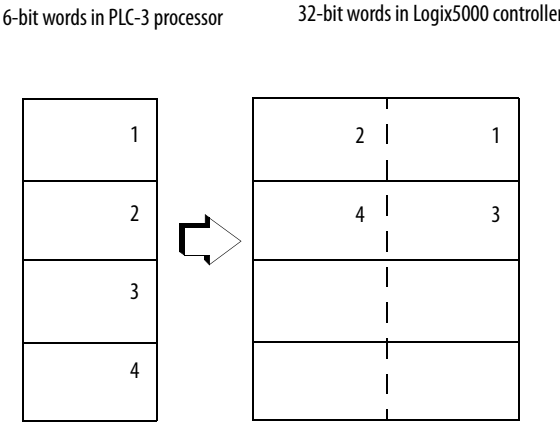
The following diagrams show how the typed and word-range commands differ. The example uses read commands from a PLC-3 processor to a Logix5000 controller.

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.

Specify 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.

The table explains the path for MSG instructions originating from a Logix5000 controller and being writing to another controller.

Message Path	Example Source and Destination	
Logix5000 → Logix5000	Source tag	*array_1[0]*
	Destination tag	*array_2[0]*
	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	*array_1[0]*
	Destination element	*N7:10*
	You can use an alias tag for the source tag (in originating Logix5000 controller).	
Logix5000 → PLC-2	Source tag	*array_1[0]*
	Destination element	*010*

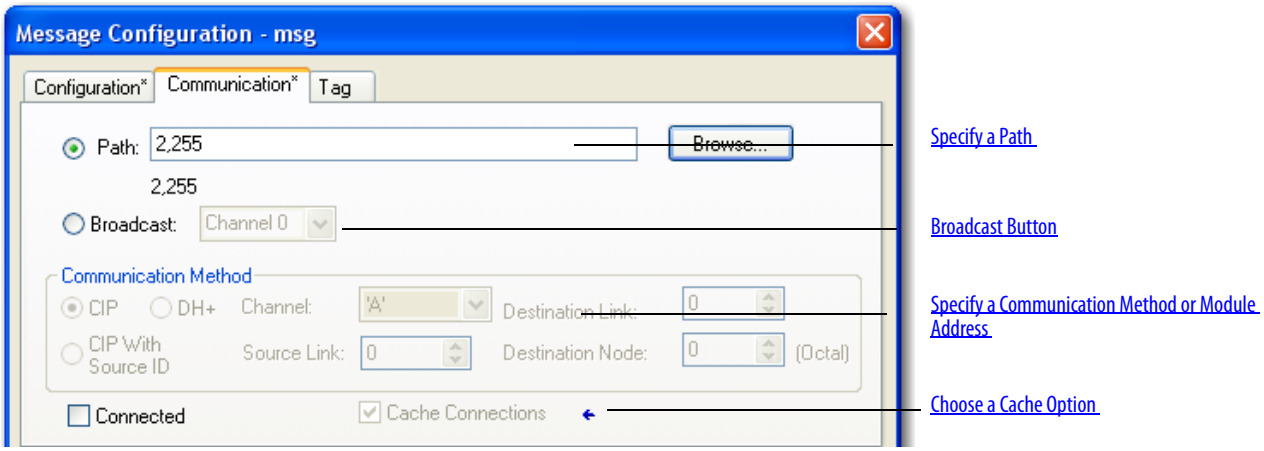
The table explains the path for MSG instructions originating from a Logix5000 controller and reading from another controller.

Message Path	Example Source and Destination	
Logix5000 → Logix5000	Source tag	*array_1[0]*
	Destination tag	*array_2[0]*
	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	*N7:10*
	Destination tag	*array_1[0]*
	You can use an alias tag for the destination tag (in originating Logix5000 controller).	
Logix5000 → PLC-2	Source element	*010*
	Destination tag	*array_1[0]*

Specify the Communication Details

You typically set up a broadcast in ladder logic or structured text programs. In ladder logic, you can add a rung and click on the MSG property to access the Message Configuration dialog box to set up a new message. In structured text, you type MSG (aMsg) and then right-click on aMsg to get the Message Configuration dialog box to configure a message.

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

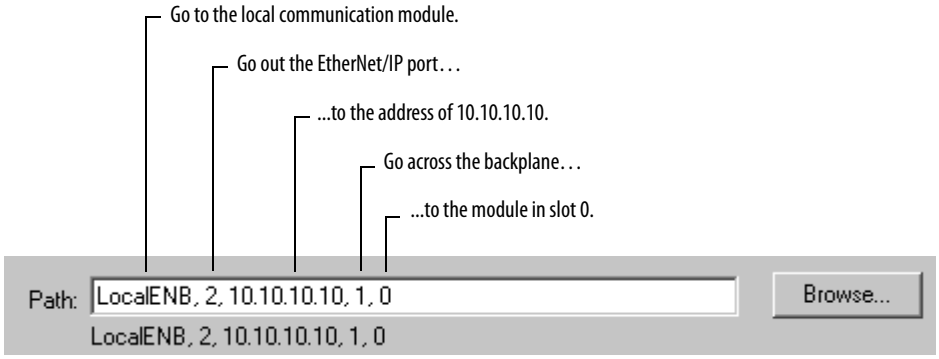


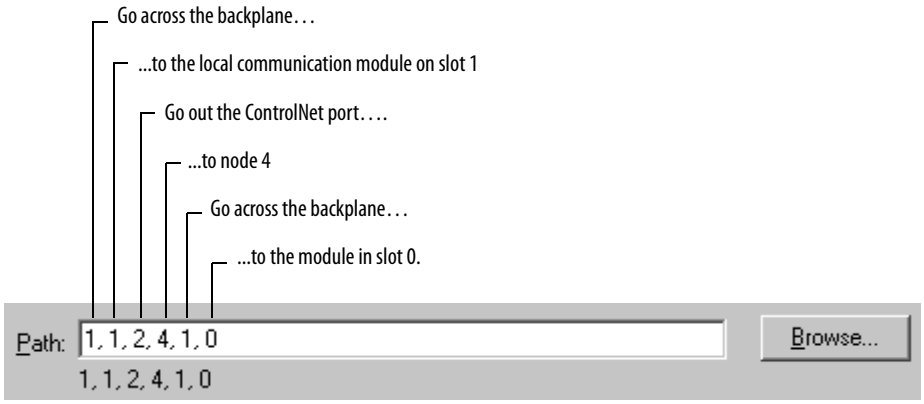
Specify a Path

The path shows the route that the message takes to get to the destination. It uses either names from the I/O configuration of the controller, numbers that you type, or both. You can default the path by using the broadcast button, which must be enabled with the system protocol and message type.

If	Then
The I/O configuration of the controller has the module that gets the message.	Use *Browse* to select the module.
The I/O configuration of the controller has only the local communication module.	1. Use Browse to select the local communication module. 2. Type the rest of the path.
The I/O configuration of the controller doesn't have any of the modules that you need for the message.	Type the path.

Example
The I/O configuration of the controller has the module that gets the message.
The I/O configuration of the controller has only the local communication module.
The I/O configuration of the controller doesn't have any of the modules that you need for the message.





To type a path, use this format:

port, next_address, port, next_address, ...

Where	Is	
	For this network	**Type**
Port	Backplane	1
	DF1 (serial, serial channel 0)	2
	ControlNet	
	EtherNet/IP	
	DH+ channel A	
	DH+ channel B	3
	DF1 channel 1 (serial channel 1)	
Next_address	Backplane	Slot number of the module
	DF1 (serial)	Station address (0-254)
	ControlNet	Node number (1-99 decimal)
	DH+	8# followed by the node number (1-77 octal) For example, to specify the octal node address of 37, type 8#37.
	EtherNet/IP	You can specify a module on an EtherNet/IP network by using any of these formats: IP address (for example, 10.10.10.10) IP address:Port (for example, 10.10.10.10:24) DNS name (for example, tanks) DNS name:Port (for example, tanks:24)

Broadcast Button

This functionality for RSLogix 5000 software, beginning with version 18, enhances the ability to define the route and message type that are required to send a message to its destination. You still can type in a path, such as '2,255'. However, manually entering a path is problematic for two reasons:

- The number entered into the path may be confusing.
- It's error-prone, as you may forget to set up other conditions, such as system protocol and message type.

The Broadcast button, when enabled, allows you to default the path by selecting an available channel(s) in a combo box. The number of channels listed in the combo box depends on the current controller.

By default, the Path radio button on the Communication tab is active (dot in radio button). Do these steps to enable the Broadcast button and select a channel to default a path for the message.

1. On the Controller Organizer, right-click Controller, and choose Properties.

   The Controller Properties dialog box appears.
2. Click the System Protocol tab.
3. Choose DF1 Master in the Protocol box.

   The Polling mode defaults 'Message Based' (slave can initiate messages).
4. Click OK.
5. In ladder logic, click the box inside the MSG tag.

   The Message Configuration dialog box appears with the Configuration tab open.
6. In the Message Type box, choose CIP Data Table Write.(1)

   See page 167 for an example.
7. Click OK.

   You have enabled the Broadcast button on the Communication tab.
8. Click the Communication tab.
9. Next to the Broadcast button, choose a channel in the combo box. The number of channels in the combo box depends on the controller.

   When you select channel 0 or 1, the corresponding message path on the Message Configuration dialog box defaults to 2,255 (channel 0) or 3,255 (channel 1). The Path grays out to not allow you to manually enter a path value.
10. Click OK.

(1) See system protocol instructions on page 183 to enable the Broadcast button.

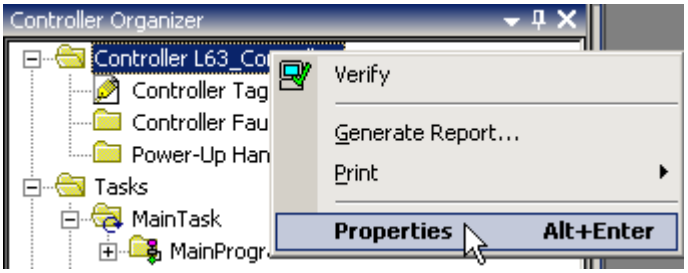
System Protocol Tab Configuration

To run broadcast in ControlLogix controllers in the Logix Designer application, you must configure the System Protocol tab in the Controller Properties dialog box. The protocol must be compatible with the message type of 'write' on the Message Configuration dialog box.

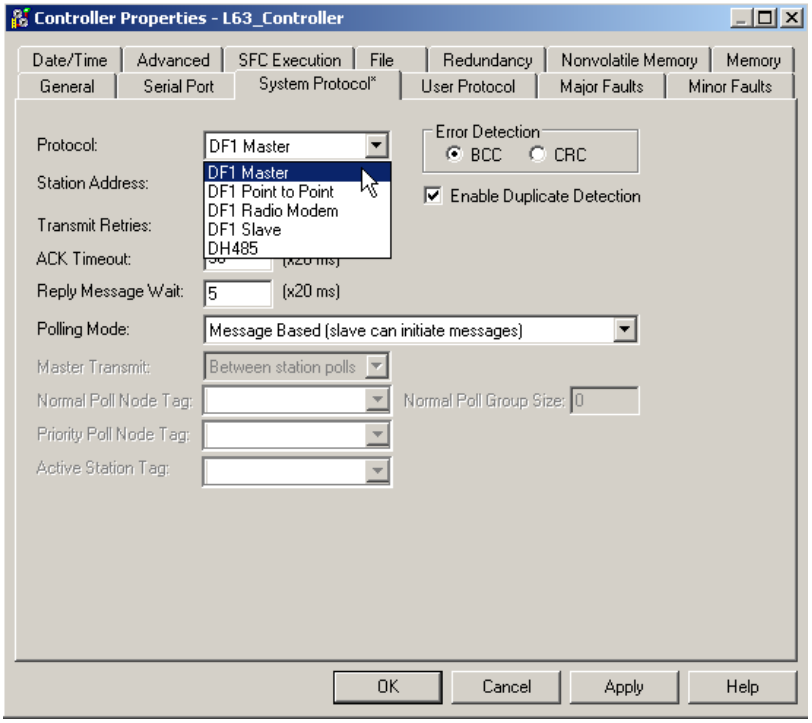
See page 167 for an example in the Message Type box.

Follow these steps to set up the system protocol to be compatible with the broadcast feature.

1. Create or open an existing controller in the application.
2. On the Controller Organizer, right-click the controller name, and choose Properties.



The Controller Properties dialog box appears.



3. Click System Protocol tab.

4. Choose a protocol from the Protocol box.

IMPORTANT	The Message Type box on the Message Configuration Tab dialog box must be write-typed to be compatible with the system protocol. Otherwise, the Broadcast button is disabled.

5. Enter the information on the System Protocol tab for each protocol outlined below.

Topic	Description
Protocol	DF-1 Master
Station Address	Type controller station address number
Transmit Retries	3
ACK Timeout	50
Reply Message Wait	5
Polling Mode	Choose from the following modes: • 'Message Based' to poll the slave using message instruction • 'Slave can initiate message' for slave to slave broadcast • 'Standard' to have the schedule poll for the slave
Error Detection	BCC
Duplicate Detection	Enabled (checked)

Topic	Description
Protocol	DF-1 Slave
Station Address	Type controller station address number
Transmit Retries	3
Slave Poll Timeout	3000
EOT Suppression	Disable (unchecked)
Error Detection	BCC
Duplicate Detection	Enabled (checked)

Topic	Description
Protocol	DF-1 Slave
Station Address	Type controller station address number
Enable Store and Forward	Enable box (checkmark) to use store and forward tag
Error Detection	BCC

6. Click OK.

For Block Transfers

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	• Local communication module (for example, 1756-CNB module) • Remote adapter module (for example, 1771-ACN module)
Universal remote I/O	• Local communication module (for example, 1756-DHRIO module) • One emote adapter module (for example, 1771-ASB module) for each rack, or portion of a rack, in the chassis • Block-transfer module (optional)

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	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 software	**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 software (0…65535).
		Destination Node	Destination ID (0…77 octal) provided by the application to RSLinx software. For a DDE topic in RSLinx software, 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	→	Your option(1)
PLC-2, PLC-3, PLC-5, or SLC (all types)	CIP CIP with Source ID	
	DH+	X
CIP generic	→	Your option(2)
Block-transfer read or write	→	X

(1) CIP data table read or write messages can be connected or unconnected. But, for most applications, we recommend you leave CIP data table read or write messages connected.

(2) CIP generic messages can be connected or unconnected. 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	• 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 instruction uses 1 connection and 1 cached buffer. They share the connection and the buffer.

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 one connection. Caching both messages counts as one on the cache list.

Guidelines

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

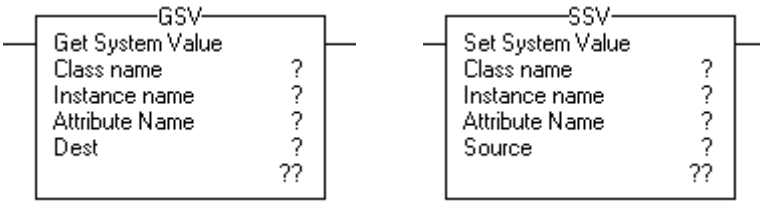
Guideline	Details
1. For each MSG instruction, create a control tag.	Each MSG instruction requires its own control tag.
	• Data type = MESSAGE
	• Scope = controller
	• The tag **cannot** be part of an array or a user-defined data type.
2. Keep the source and/or destination data at the controller scope.	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.	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.
	This increases the efficiency of your project because Logix controllers execute more efficiently and use less memory when working with 32-bit integers (DINTs).
	To convert between INTs and DINTs, see the Logix5000 Controllers Common Procedures Programming Manual, publication 1756-PM001.
4. Cache the connected MSGs that execute most frequently.	Cache the connection for those MSG instructions that execute most frequently, up to the maximum number permissible for your controller revision.
	This optimizes execution time because the controller does not have to open a connection each time the message executes.
5. If you want to enable more than 16 MSGs at one time, use some type of management strategy.	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:
	• Enable each message in sequence.
	• Enable the messages in groups.
	• Program a message to communicate with multiple devices. For more information, see the Logix5000 Controllers Common Procedures Programming Manual, publication 1756-PM001.
	• Program logic to coordinate the execution of messages. For more information, see the Logix5000 Controllers Common Procedures Programming Manual, publication 1756-PM001.
6. Keep the number of unconnected and uncached MSGs less than the number of unconnected buffers.	The controller can have 10…40 unconnected buffers. The default number is 10.
	• 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 the Logix5000 Controllers Common Procedures Programming Manual, 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 structure	Tag	Destination for attribute data.
Source (SSV)	SINT INT DINT REAL structure	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.

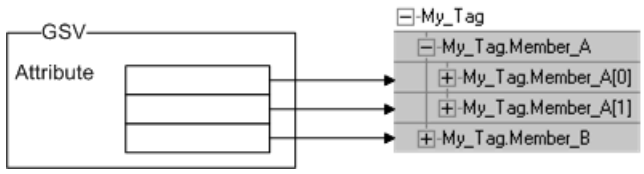
When you enter a GSV/SSV instruction, the programming software displays the valid object classes, object names, and attribute names for each instruction. For the GSV instruction, you can get values for all the available attributes. For the SSV instruction, the software displays only those attributes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

ATTENTION: Use the GSV and SSV instructions carefully. Making changes to objects may cause unexpected controller operation or injury to personnel.

You **must** test and confirm that the instructions don't change data that you don't want them to change.

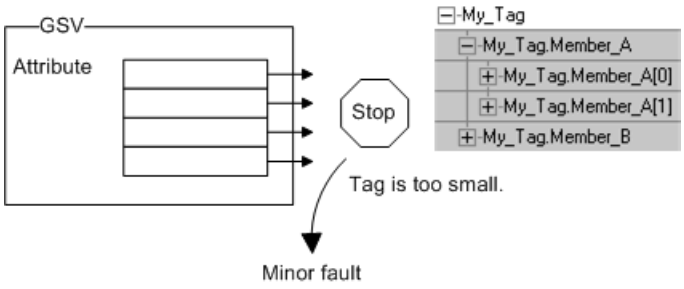
The GSV and SSV instructions write or read past a member into other members of a tag. If the tag is too small, the instructions don't write or read the data. They log a minor fault instead.

Example 1



Member_A is too small for the attribute. So the GSV instruction writes the last value to Member_B.

Example 2



My_Tag is too small for the attribute. So the GSV instruction stops and logs a minor fault.

The GSV/SSV Objects section shows each object's attributes and their associated data types. For example, the MajorFaultRecord attribute of the Program object needs 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.	N/A
Rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	N/A
EnableIn is set	N/A	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.

Table 6 - GSV/SSV Objects

For information about this object	See this page or publication
AddOnInstructionDefinition	192
Axis	See the SERCOS Motion Configuration and Startup User Manual, publication MOTION-UM001.
Axis_Consumed	
Axis_Generic	
Axis_Generic_Drive	
Servo	
Servo_Drive	
Virtual	
Axis_CIP_Drive	See the CIP Motion Configuration and Startup User Manual, publication MOTION-UM003.
Controller	192
ControllerDevice	194
CoordinateSystem	See the Motion Coordinate System User Manual, publication MOTION-UM002.
CST	194
DF1	196
FaultLog	198
Message	198
Module	199
MotionGroup	See the Motion Reference Manual, publication MOTION-RM001.
Program	200
Redundancy	See the ControlLogix Enhanced Redundancy System User Manual, publication 1756-UM535.
Routine	201
SafetyAddonInstructionDefinition, SafetyController, SafetyProgram, SafetyRoutine, SafetyTask.	See the GuardLogix Controllers User Manual, publication 1756-UM020.
SerialPort	202

Table 6 - GSV/SSV Objects

Task	203
TimeSynchronization	See the Integrated Architecture and CIP Sync Configuration Application Technique, publication IA-AT003.
WallClockTime	205

AddOnInstructionDefinition Attributes

The AddOnInstructionDefinition object lets you customize instructions for sets of commonly-used logic, provides a common interface to this logic, and provides documentation for the instruction.

For details, see the Logix5000 Controllers Add-On Instructions Programming Manual, publication 1756-PM010.

Attribute	Data Type	Instruction within Standard Task	Instruction within Safety Task	Description
LastEditDate	LINT	GSV	None	Date and time stamp of the last edit to an Add-On Instruction definition.
MajorRevision	DINT	GSV	None	Major revision number of the Add-On Instruction.
MinorRevision	DINT	GSV	None	Minor revision number of the Add-On Instruction.
Name	String	GSV	GSV	Name of the Add-On Instruction.
RevisionExtendedText	String	GSV	None	Text describing the revision of the Add-On Instruction.
SafetySignature ID	DINT	GSV	None	In a safety project, the ID number, date, and timestamp of an Add-On Instruction definition.
SignatureID	DINT	GSV	None	32-bit identification number of an Add-On Instruction definition.
Vendor	String	GSV	None	Vendor that created the Add-On Instruction.

Controller Attributes

The Controller object provides status information about controller execution.

Attribute	Data Type	Instruction	Description
Audit Value	DINT[2], LINT	GSV	The audit value is a unique value that is generated when a project is downloaded to the controller or loaded from removable storage. When a change is detected, this value is updated. To specify which changes are monitored, use the ChangesToDetect attribute. **Note:** We recommend using the DINT[2] data type to avoid limitations when working with LINT data types in Rockwell Automation controllers.
ChangesToDetect See Table 4	DINT[2], LINT	GSV, SSV	Used to specify which changes are monitored. When a monitored change occurs, the Audit Value is updated. **Note:** We recommend using the DINT[2] data type to avoid limitations when working with LINT data types in Rockwell Automation controllers.
CanUseRPIFrom Producer	DINT	GSV	Identifies whether to use the RPI specified by the producer. **ValueMeaning** 0 Do not use the RPI specified by the producer 1 Use the RPI specified by the producer
ControllerLog Execution Modification Count	DINT	GSV SSV	Number of controller log entries that originated from a program/task properties change, an online edit, or a controller timeslice change. It can also be configured to include log entries originating from forces. The number will be reset if RAM enters a bad state. The number is not capped at the largest DINT, and a rollover can occur.

Attribute	Data Type	Instruction	Description
ControllerLog TotalEntryCount	DINT	GSV SSV	Number of controller log entries since the last firmware upgrade. The number will be reset if RAM enters a bad state. The number is capped at the largest DINT.
DataTablePad Percentage	INT	GSV	Percentage (0...100) of free data table memory set aside.
IgnoreArrayFaultsDuringPostScan	SINT	GSV SSV	Used to configure the suppression of selected faults encountered when an SFC action is postscanned. Only valid when SFCs are configured for automatic reset. **ValueMeaning** 0 Do not suppress faults during postscan execution (default behavior - recommended). 1 Major faults 4/20 (Array subscript too large) and 4/83 (Value out of range) are automatically suppressed while postscanning SFC actions. When a fault is suppressed, the controller uses an internal fault handler to automatically clear the fault. This causes the faulted instruction to be skipped, with execution resuming at the following instruction. Because the fault handler is internal, you do not have to configure a fault handler to get this behavior. In fact, even if a fault handler is configured, a suppressed fault will not trigger it.
InhibitAutomaticFirmwareUpdate	BOOL	GSV SSV	Identifies whether to enable the firmware supervisor. **ValueMeaning** 0 Do not execute the firmware supervisor 1 Execute the firmware supervisor
KeepTestEditsOnSwitchover	SINT	GSV	Identifies whether to maintain test edits on controller switchover. **ValueMeaning** 0 Automatically untest edits at switchover 1 Continue test edits at switchover
Name	String	GSV	Name of the controller.
Redundancy Enabled	SINT	GSV	Identifies whether the controller is configured for redundancy. **ValueMeaning** 0 Not configured for redundancy 1 Configured for redundancy
ShareUnusedTimeSlice	INT	GSV SSV	Identifies how the continuous task and the background tasks shared any unused timeslice. **ValueMeaning** 0 The operating system will not give control to the continuous task even if background is done. 1 Continuous task still runs if the background tasks are done (default). 2 or greater will cause a minor fault to be logged and the setting remains unchanged.
TimeSlice	INT	GSV SSV	Percentage of available CPU (10...90) that is assigned to communications. This value cannot be changed when the keyswitch is in the Run position.

ControllerDevice Attributes

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: **ValueMeaning** 15SoftLogix5800 49PowerFlex® with DriveLogix5725 52PowerFlex with DriveLogix5730 53Emulator 541756-L61 ControlLogix 551756-L62 ControlLogix 561756-L63 ControlLogix 571756-L64 ControlLogix 641769-L31 CompactLogix 651769-L35E CompactLogix 671756-L61S GuardLogix 681756-L62S GuardLogix 691756-LSP GuardLogix 721768-L43 CompactLogix 741768-L45 CompactLogix 761769-L32C CompactLogix 771769-L32E CompactLogix 801769-L35CR CompactLogix 851756-L65 ControlLogix 861756-L63S GuardLogix 871769-L23E-QB1 CompactLogix 881769-L23-QBFC1 CompactLogix 891769-L23E-QBFC1 CompactLogix
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.
SerialNumber	DINT	GSV	Serial number of the device. The serial number is assigned when the device is built.
Status	INT	GSV	**Device Status BitsController Status Bits** **7...4Meaning13...12Meaning** 0000Reserved01Keyswitch in run 0001Flash update in progress10Keyswitch in program 0010Reserved11Keyswitch in remote 0011Reserved 0100Flash is bad**15...14Meaning** 0101Faulted modes01Controller is changing modes 0110Run10Debug mode if controller in run mode 0111Program **Fault Status Bits** **11...8Meaning** 0001Recoverable minor fault 0010Unrecoverable minor fault 0100Recoverable major fault 1000Unrecoverable major fault
Type	INT	GSV	Identifies the device as a controller. Controller = 14.
Vendor	INT	GSV	Identifies the vendor of the device. Allen-Bradley = 0001.

CST Attributes

The coordinated system time (CST) 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. **BitMeaning** 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. 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 master CST object via a system time update 4 Local network master: the CST object is the local network master time source 5 Relay mode: the CST object is acting in a time relay mode 6 Duplicate master detected: a duplicate local network time master was detected. This bit is always 0 for time-dependent nodes. 7 Unused 8 -900 = time dependent node 01 = time master node 10 = time relay node 11 = Unused 10-15Unused
CurrentValue	DINT[2]	GSV	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.

DF1 Attributes

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).
Diagnostic Counters	INT[19]	GSV	Array of diagnostic counters for the DF1 communication driver.

Word offsetDF1 point-to-pointDF1 slaveMaster
0Signature (0x0043)Signature (0x0042)Signature (0x0044)
1Modem bitsModem bitsModem bits
2Packets sentPackets sentPackets sent
3Packets receivedPackets receivedPackets received
4Undelivered packetsUndelivered packetsUndelivered packets
5UnusedMessages retriedMessages retried
6NAKs receivedNAKs receivedUnused
7ENQs receivedPoll packets receivedUnused
8Bad packets NAKedBad packets not ACKedBad packets not ACKed
9No memory sent NAKNo memory not ACKedUnused
10Duplicate packets receivedDuplicate packets receivedDuplicate packets received
11Bad characters receivedUnusedUnused
12DCD recoveries countDCD recoveries countDCD recoveries count
13Lost modem countLost modem countLost modem count
14UnusedUnusedPriority scan time maximum
15UnusedUnusedPriority scan time last
16UnusedUnusedNormal scan time maximum
17UnusedUnusedNormal scant time last
18ENQs sentUnusedUnused

Duplicate Detection	SINT	GSV	Enables duplicate message detection. **Value Meaning** 0 Duplicate message detection disabled Non zeroDuplicate message detection enabled
Embedded ResponseEnable	SINT	GSV	Enables embedded response functionality (point-to-point only). **Value Meaning** 0 Initiated only after one is received (default) 1 Enabled unconditionally
EnableStoreFwd	SINT	GSV	Enables the store and forward behavior when receiving a message. **Value Meaning** 0 Do not forward message Non zero. See the store and forward table when receiving a message (default)
ENQTransmit Limit	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). **ValueMeaning** 0 EOT suppression disabled (disabled) Non zeroEOT suppression enabled
ErrorDetection	SINT	GSV	Specifies the error-detection scheme. **ValueMeaning** 0 BCC (default) 1 CRC
MasterMessageTransmit	SINT	GSV	Current value of the master message transmission (master only). **ValueMeaning** 0 Between station polls (default) 1 In poll sequence (in place of master's station number)
MaxStation Address	SINT	GSV	Current value (0...31) of the maximum node address on a DH-485 network. Default is 31.
NAKReceiveLimit	SINT	GSV	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.

Attribute	Data Type	Instruction	Description
NormalPollGroupSize	INT	GSV	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.
PollingMode	SINT	GSV	Current polling mode (master only). Default setting is 1. **ValueMeaning** 0Message-based, but don't allow slaves to initiate messages 1Message-based, but allow slaves to initiate messages (default) 2Standard, single-message transfer per node scan 3Standard, multiple-message transfer per node scan
ReplyMessage Wait	DINT	GSV	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).
SlavePollTimeout	DINT	GSV	The amount of time in msecs 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).
StationAddress	INT	GSV	Current station address of the serial port. Valid value 0...254. Default is 0.
TokenHoldFactor	SINT	GSV	Current value (1...4) of the maximum number of messages sent by this node before passing the token on a DH-485 network. Default is 1.
TransmitRetries	SINT	GSV	Number of times to resend a message without getting an acknowledgment (master and slave only). Valid value 0...127. Default is 3.
PendingACK Timeout	DINT	SSV	Pending value for the ACKTimeout attribute.
Pending Duplicate Detection	SINT	SSV	Pending value for the DuplicateDetection attribute.
Pending Embedded ResponseEnable	SINT	SSV	Pending value for the EmbeddedResponse attribute.
PendingEnable StoreFwd	SINT	SSV	Pending value for the EnableStoreFwd attribute.
PendingENQ TransmitLimit	SINT	SSV	Pending value for the ENQTransmitLimit attribute.
PendingEOT Suppression	SINT	SSV	Pending value for the EOTSuppression attribute.
PendingError Detection	SINT	SSV	Pending value for the ErrorDetection attribute.
PendingMaster Message Transmit	SINT	SSV	Pending value for the MasterMessageTransmit attribute.
PendingMax StationAddress	SINT	SSV	Pending value for the MaxStationAddress attribute.
PendingNAK ReceiveLimit	SINT	SSV	Pending value for the NAKReceiveLimit attribute.
PendingNormal PollGroupSize	INT	SSV	Pending value for the NormalPollGroupSize attribute.
PendingPolling Mode	SINT	SSV	Pending value for the PollingMode attribute.
PendingReply MessageWait	DINT	SSV	Pending value for the ReplyMessageWait attribute.
PendingSlavePollTimeout	DINT	SSV	Pending value for the SlavePollTimeout attribute.

Attribute	Data Type	Instruction	Description
PendingStation Address	INT	SSV	Pending value for the StationAddress attribute.
PendingToken HoldFactory	SINT	SSV	Pending value for the TokenHoldFactor attribute.
PendingTransmitRetries	SINT	SSV	Pending value for the TransmitRetries attribute.

FaultLog Attributes

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.
MajorFaultBits	DINT	GSV SSV	Individual bits indicate the reason for the current major fault. **Bit** **Meaning** 1 Power loss 3 I/O 4 Instruction execution (program) 5 Fault Handler 6 Watchdog 7 Stack 8 Mode change 11 Motion
MinorEvents	INT	GSV SSV	How many minor faults have occurred since the last time this counter was reset.
MinorFaultBits	DINT	GSV SSV	Individual bits indicate the reason for the current minor fault. **Bit** **Meaning** 4 Instruction execution (program) 6 Watchdog 9 Serial port 10 Battery/Energy Storage Module (ESM)(1)

(1) Battery for 1756-L6*x*, 1769-L2*x*, and 1769-L3*x* controllers. ESM for 1756-L7*x* controllers and CompactLogix 5370 series controllers.

Message Attributes

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.

Attribute	Data Type	Instruction	Description
MessageType	SINT	GSV SSV	Specifies the type of message. **ValueMeaning** 0Not initialized
Port	SINT	GSV SSV	Indicates which port the message should be sent on. **ValueMeaning** 1Backplane 2Serial port
Timeout Multiplier	SINT	GSV SSV	Determines when a connection should be considered timed out and closed. **ValueMeaning** 0Connection will timeout in 4 times the update rate default) 1Connection will timeout in 8 times the update rate 2Connection will timeout in 16 times the update rate
Unconnected Timeout	DINT	GSV SSV	Timeout in microseconds for all unconnected messages. The default is 30,000,000 microseconds (30 s).

Module Attributes

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. **ValueMeaning** 16#0000Standby: the controller is powering up. 16#1000Faulted: 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#2000Validating: the Module object is verifying Module object integrity prior to establishing connections to the module. 16#3000Connecting: the Module object is initiating connections to the module. 16#4000Running: all connections to the module are established and data is transferring. 16#5000Shutting down: the Module object is in the process of shutting down all connections to the module. 16#6000Inhibited: the Module object is inhibited (the inhibit bit in the Mode attribute is set). 16#7000Waiting: the parent object upon which this Module object depends is not running.
FaultCode	INT	GSV	A number that identifies a module fault, if one occurs.
FaultInfo	DINT	GSV	Provides specific information about the Module object fault code.
Firmware SupervisorStatus	INT	GSV	Identifies current operating state of the firmware supervisor feature. **ValueMeaning** 0Module updates are not being executed 1Module updates are being executed

Attribute	Data Type	Instruction	Description
ForceStatus	INT	GSV	Specifies the status of forces. **BitMeaning** 0Forces installed (1=yes, 0-no) 1Forces enabled (1=yes, 0=no)
Instance	DINT	GSV	Provides the instance number of this module object.
LEDStatus	INT	GSV	Specifies the current state of the I/O status indicator on the front of the controller.(1) **ValueMeaning** 0Status indicator off: No Module objects are configured for the controller. (There are no modules in the I/O Configuration section of the controller organizer.) 1Flashing red: None of the Module objects are Running. 2Flashing green: At least one Module object is not Running. 3Solid green: All the Module objects are Running. You do not enter an object name with this attribute because this attribute applies to the entire collection of modules.
Mode	INT	GSV SSV	Specifies the current mode of the Module object. **BitMeaning** 0If set, causes a major fault to be generated if any of the Module object connections fault while the controller is in Run mode. 2If set, causes the Module object to enter Inhibited state after shutting down all the connections to the module.

(1) The 1756-L7*x* controllers do not have a status indicator display on the front of the controller, but do use this functionality.

Program Attributes

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

Attribute	Data Type	Instruction within Standard Task	Instruction within Safety Task	Description
DisableFlag	SINT	GSV SSV	None	Controls this program's execution. **Value Meaning** 0 Execution enabled non-zero) Execution disabled
	DINT	GSV	GSV	A non-zero value will disable.
LastScanTime	DINT	GSV SSV	None	Time it took to execute this program the last time it was executed. Time is in microseconds.
MajorFault Record	DINT[11]	GSV SSV	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	Data Type	Style	Description
TimeLow	DINT	Decimal	Lower 32 bits of fault timestamp value
TimeHigh	DINT	Decimal	Upper 32 bits of fault timestamp value
Type	INT	Decimal	Fault type (program, I/O, and so forth)
Code	INT	Decimal	Unique code for the fault (depends on fault type)
Info	DINT[8]	Hexadecimal	Fault specific information (depends on fault type and code)

Attribute	Data Type	Instruction within Standard Task	Instruction within Safety Task	Description
MinorFault Record	DINT[11]	GSV SSV	GSV SSV	Records minor faults for this program We recommend that you create a user-defined structure to simplify access to the MinorFaultRecord attribute:
NameData TypeStyleDescription TimeLowDINTDecimalLower 32 bits of fault timestamp value TimeHighDINTDecimalUpper 32 bits of fault timestamp value TypeINTDecimalFault type (program, I/O, and so forth) CodeINTDecimalUnique code for the fault (depends on fault type) InfoDINT[8]HexadecimalFault specific information (depends on fault type and code)				
MaxScanTime	DINT	GSV SSV	None	Maximum recorded execution time for this program. Time is in microseconds.
Name	String	GSV	GSV	Name of the program.

Routine Attributes

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

Attribute	Data Type	Instruction within Standard Task	Instruction within Safety Task	Description
Instance	DINT	GSV	GSV	Provides the instance number for this routine object. Valid values are 0…65,535.
Name	String	GSV	GSV	Name of the routine.
SFCPaused	INT	GSV	None	In an SFC routine, indicates whether the SFC is paused. **Value Meaning** 0 SFC is not paused 1 SFC is paused
SFCResuming	INT	GSV SSV	None	In an SFC routine, indicates whether the SFC is resuming execution. **Value Meaning** 0 SFC is not executing. This attribute is automatically set to 0 at the end of a scan in which the chart was executed 1 SFC is executing. Step and action timers will retain their previous value if configured to do so. This attribute is automatically set to 1 on the first scan after a chart is unpaused.

Safety Attributes

The Safety Controller object provides safety status and safety signature information. The SafetyTask and SafetyFaultRecord attributes can capture information about non-recoverable faults.

See the GuardLogix Controllers User Manual, publication 1756-UM020.

Attribute	Data Type	Instruction within Standard Task	Instruction within Safety Task	Description
SafetyLocked	SINT	GSV	None	Indicates whether the controller is safety locked or unlocked.
SafetyStatus	INT	GSV	None	Specifies the safety status as: **Value** **Meaning** 1000000000000000 Safety task OK. 1000000000000001 Safety task inoperable. 0000000000000000 Partner missing. 0000000000000001 Partner unavailable. 0000000000000010 Hardware incompatible. 0000000000000011 Firmware incompatible.
SafetySignature Exists	SINT	GSV	GSV	Indicates whether the safety task signature is present.
SafetySignature ID	DINT	GSV	None	32-bit identification number.
SafetySignature	String	GSV	None	32-bit identification number.
SafetyTaskFault Record	DINT[11]	GSV	None	Records safety task faults.

SerialPort Attributes

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).
ComDriverID	SINT	GSV	Specifies the specific driver. **Value** **Meaning** 0xA2 DF1 (default) 0xA3 ASCII
DataBits	SINT	GSV	Specifies the number of bits of data per character. **Value** **Meaning** 7 7 data bits (ASCII only) 8 8 data bits (default)
DCDDelay	INT	GSV	Specifies the amount of time to wait for the data carrier detect (DCD) to become low before erroring the packet. The delay is in counts of 1 s packets. Default is 0 counter.
Parity	SINT	GSV	Specifies the parity. **Value** **Meaning** 0 No parity (no default) 1 Odd parity (ASCII only) 2 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.

Attribute	Data Type	Instruction	Description
StopBits	SINT	GSV	Specifies the number of stop bits. **Value** **Meaning** 1 1 stop bit (default) 2 2 stop bits (ASCII only)
PendingBaudRate	DINT	SSV	Pending value for the BaudRate attribute.
PendingCOM DriverID	SINT	SSV	Pending value for the COMDriverID attribute.
PendingDataBits	SINT	SSV	Pending value for the DataBits attribute.
PendingDCD Delay	INT	SSV	Pending value for the DCDDelay attribute.
PendingParity	SINT	SSV	Pending value for the Parity attribute.
PendingRTSOff Delay	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.

Task Attributes

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

Attribute	Data Type	Instruction within Standard Task	Instruction within Safety Task	Description
DisableUpdate Outputs	DINT	GSV SSV	None	Enables or disables the processing of outputs at the end of a task. **Value** **Meaning** 0 Enable the processing of outputs at the end of the task Non zero Disable the processing of outputs at the end of the task
EnableTimeOut	DINT	GSV SSV	None	Enables or disables the timeout function of an event task. **Value** **Meaning** 0 Disable the timeout function Non zero Enable the timeout function
InhibitTask	DINT	GSV SSV	None	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. **Value** **Meaning** 0 Enable the task 0 (default) Non zero Inhibit (disable) the task
Instance	DINT	GSV	GSV	Provides the instance number of this task object. Valid values are 0...31.
LastScanTime	DINT	GSV SSV	None	Time it took to execute this task the last time it was executed. Time is in microseconds.
MaximumInterval	DINT[2]	GSV SSV	None	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.
MaximumScan Time	DINT	GSV SSV	None	Maximum recorded execution time for this program. Time is in microseconds.
MinimumInterval	DINT[2]	GSV SSV	None	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.
Name	String	GSV	GSV	Name of the task.

Attribute	Data Type	Instruction within Standard Task	Instruction within Safety Task	Description
OverlapCount	DINT	GSV SSV	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	GSV	Relative priority of this task as compared to the other tasks. Valid values 0...15.
Rate	DINT	GSV SSV	GSV	The time interval between executions of the task. Time is in microseconds.
StartTime	DINT[2]	GSV SSV	None	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	None	Status information about the task. Once the controller sets one of these bits, you must manually clear the bit. **Bit** **Meaning** 0 An EVENT instruction triggered the task (event task only) 1 A timeout triggered the task (event task only) 2 An overlap occurred for this task
Watchdog	DINT	GSV SSV	GSV	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 5.0 sec Continuous

WallClockTime Attributes

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

Attribute	Data Type	Instruction	Description
ApplyDST	SINT	GSV SSV	Identifies whether to enable daylight savings time. **Value** **Meaning** 0 Do not adjust for daylight savings time non zero Adjust for daylight savings time
CSTOffset	DINT[2]	GSV SSV	Positive offset from the CurrentValue of the CST object (coordinated system time, see page 194). DINT[0] contains the lower 32 bits of the value; DINT[1] contains the upper 32 bits of the value. Value in μs. 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 hours 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 CTSOffset, the result is the WALLCLOCKTIME CurrentValue.
DateTime	DINT[7]	GSV SSV	The date and time. **Value** **Meaning** DINT[0] Year DINT[1] Month (1...12) DINT[2] Day (1...31) DINT[3] Hour (0...23) DINT[4] Minute (0...59) DINT[5] Seconds (0...59) DINT[6] Microseconds (0...999,999)
DSTAdjustment	INT	GSV SSV	The number of minutes to adjust for daylight saving time.
LocalDateTime	DINT[7]	GSV SSV	Current adjusted local time. **Value** **Meaning** DINT[0] Year DINT[1] Month (1...12) DINT[2] Day (1...31) DINT[3] Hour (0...23) DINT[4] Minute (0...59) DINT[5] Seconds (0...59) DINT[6] Microseconds (0...999,999)
TimeZoneString	INT	GSV SSV	Time zone for the time value.

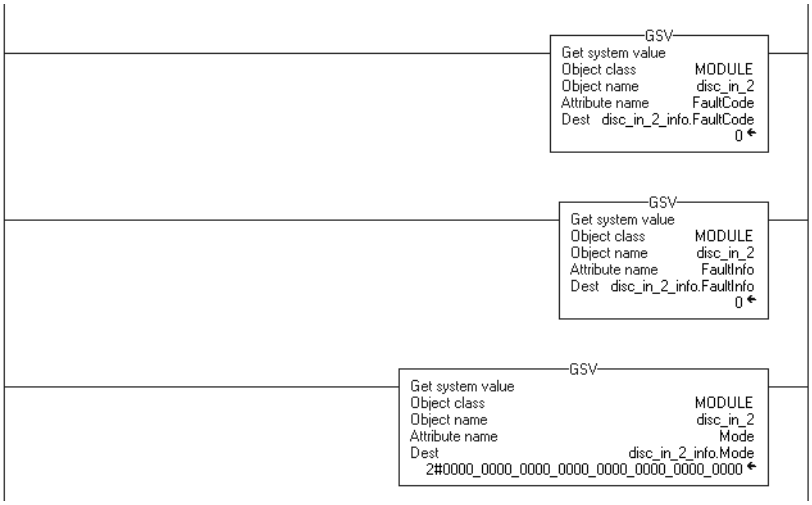
GSV/SSV Programming Example

Get 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

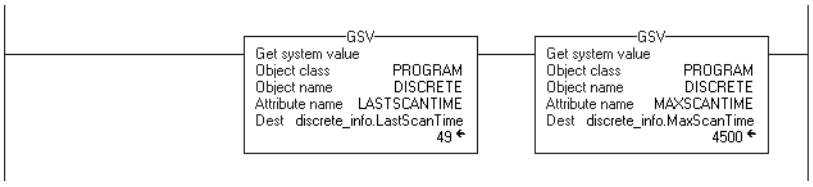


Structured Text

```
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_2info.Mode);
```

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

Relay Ladder

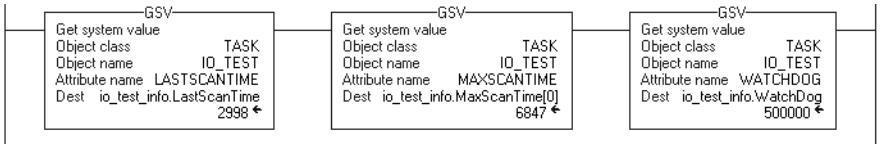


Structured Text

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

GSV(PROGRAM,DISCRETE,MAXSCANTIME,discrete_info.MaxScanTim
e);
```

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

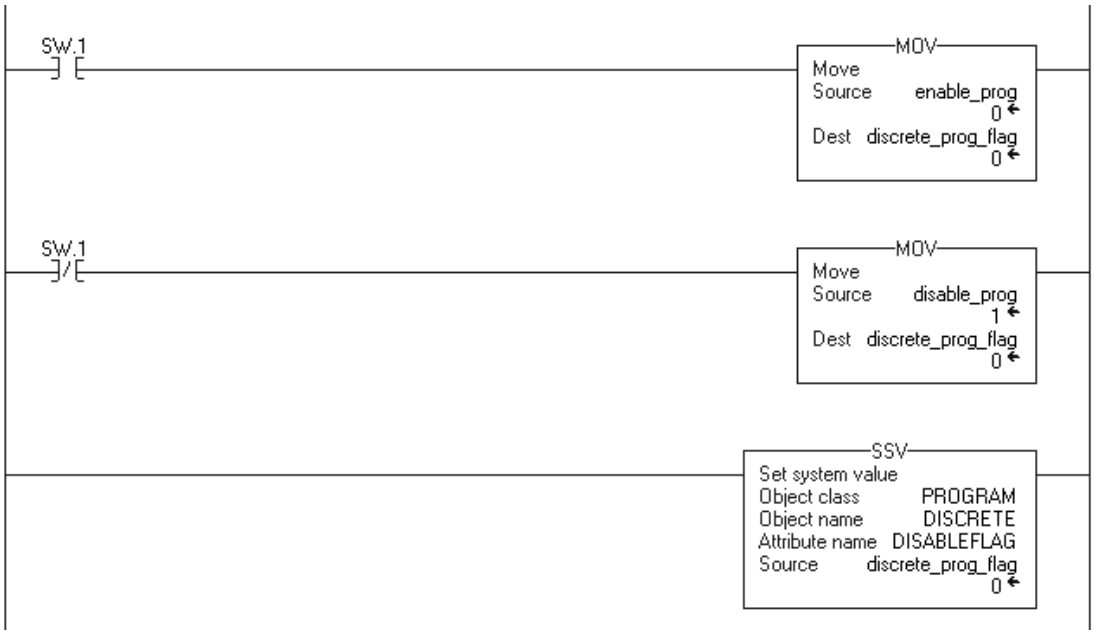
```
GSV(TASK,IO_TEST,LASTSCANTIME,io_test_info.LastScanTime);

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

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

Set 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 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

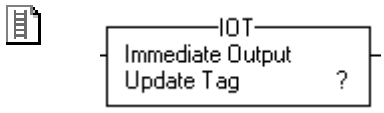
```
IF SW.1 THEN

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: • output tag of an I/O module • produced tag
			Do not choose a member or element of a tag. For example, Local:5:0 is OK but Local:5:0.Data is ***not*** OK.

Structured Text

```
IOT(output_tag);
```

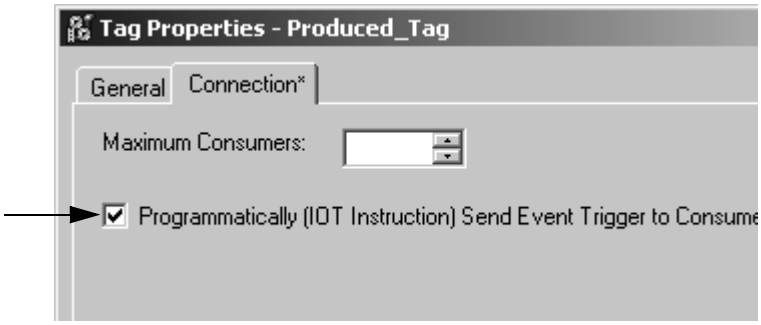
The operands are the same as those for the relay ladder IOT instruction.

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 shown below.

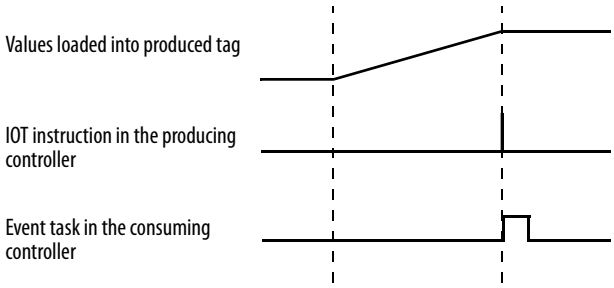
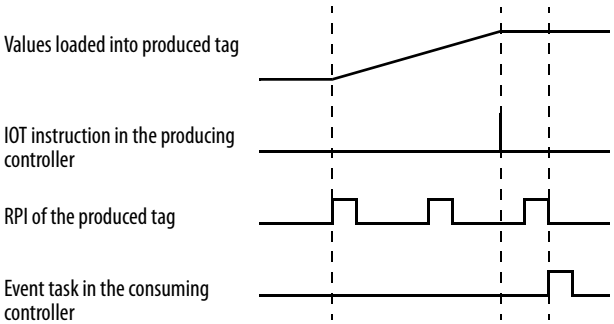


Check this box.
This configures the tag to update its event trigger only via an IOT instruction.

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.

EtherNet/IP Network	ControlNet Network
	

Arithmetic Status Flags: Not affected

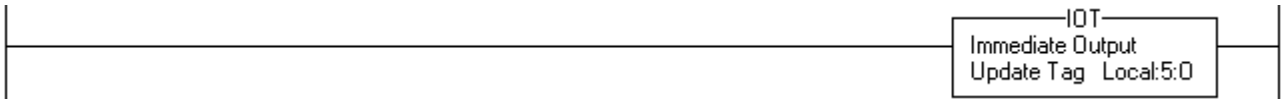
Fault Conditions: None

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.	N/A
Rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	N/A
EnableIn is set	N/A	EnableIn is always set. The instruction executes.
Instruction execution	The instruction: • 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

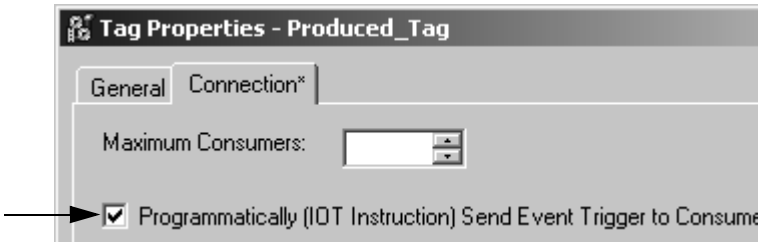


Structured Text

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

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 shown below:

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

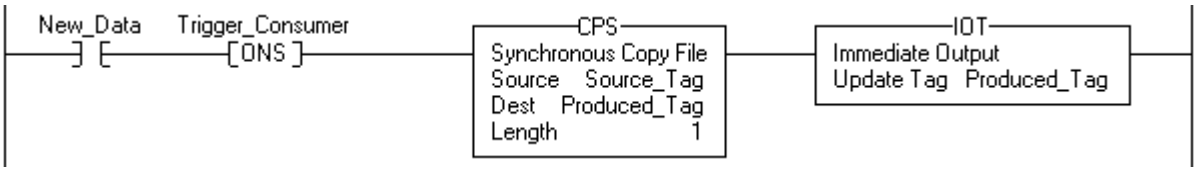


Relay Ladder

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;
```

Notes:

Compare Instructions

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

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	Page
Compare values based on an expression	CMP	Relay ladder Structured text(1)	215
Test whether two values are equal	EQU	Relay ladder Structured text(2) Function block	220
Test whether one value is greater than or equal to a second value	GEQ	Relay ladder Structured text(1) Function block	224
Test whether one value is greater than a second value	GRT	Relay ladder Structured text(1) Function block	228
Test whether one value is less than or equal to a second value	LEQ	Relay ladder Structured text(1) Function block	232
Test whether one value is less than a second value	LES	Relay ladder Structured text(1) Function block	236

If you want to	Use this instruction	Available in these languages	Page
Test whether one value is between two other values	LIM	Relay ladder Structured text(1) Function block	240
Pass two values through a mask and test whether they are equal	MEQ	Relay ladder Structured text(1) Function block	246
Test whether one value is not equal to a second value	NEQ	Relay ladder Structured text(1) Function block	251

(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.

IMPORTANT REAL values are rarely absolutely equal. If you need to determine the equality of two REAL values, use the LIM instruction.

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

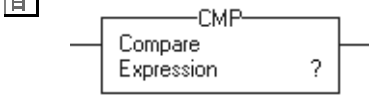
For relay ladder instructions, **bold** data types indicate optimal data types. An instruction executes faster and requires less memory if all the operands of the instruction use the same optimal data type, typically DINT or REAL.

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 by using an IF...THEN construct and expression.

IF BOOL_expression THEN

<statement>;

END_IF;

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

Description:

Define the CMP expression by 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:

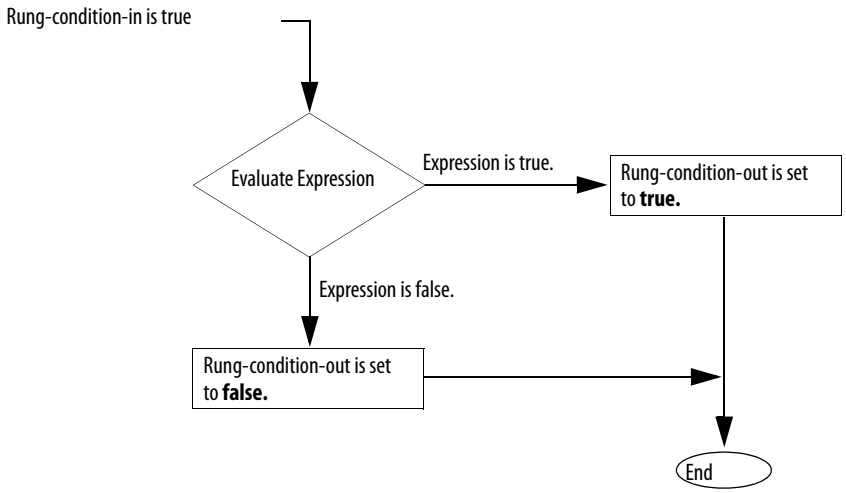
The CMP instruction affects only the arithmetic status flags if the expression contains an operator (for example, +, –, *, /) that affects the arithmetic status flags.

Fault Conditions:

None

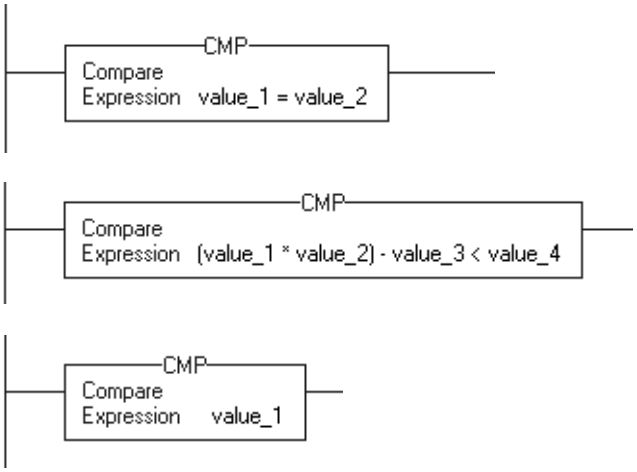
Execution:

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.



Postscan	The rung-condition-out is set to false.

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 explained in the table.

If the expression is	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
DEG	Radians to degrees	DINT, REAL
FRD	BCD to integer	DINT

Operator:	Description	Optimal
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

Format Expressions

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

Operators that operate on	Use this format	Examples
One operand	Operator (operand)	ABS(*tag_a*)
Two operands	Operand_a operator operand_b	•*tag_b* + *5* •*tag_c* AND *tag_d* •*(tag_e* ** *2)* MOD *(tag_f* / *tag_g)*

Determine 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

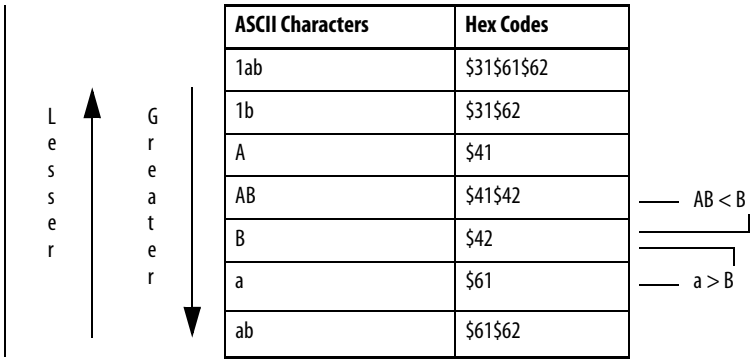
Use Strings in an Expression

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.



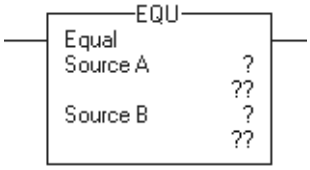
ASCII Characters	Hex Codes
1ab	$31$61$62
1b	$31$62
A	$41
AB	$41$42
B	$42
a	$61
ab	$61$62

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.

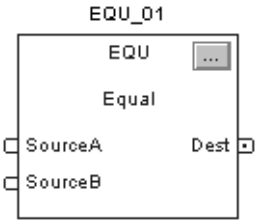
Structured Text

IF sourceA = sourceB THEN

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

See Structured Text Programming 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, note the following:

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

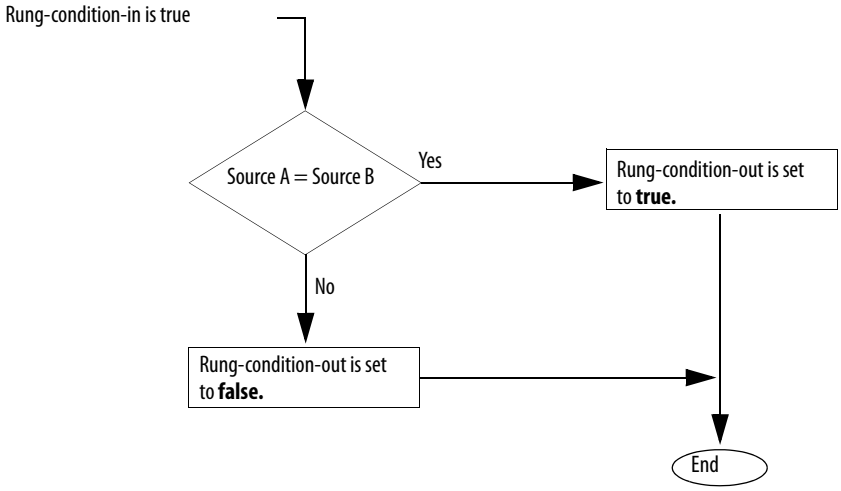
Arithmetic Status Flags: Not affected

Fault Conditions: None

Execution:

Relay Ladder

Condition	Action
Prescan	The rung-condition-out is set to false.
Rung-condition-in is false	The rung-condition-out is set to false.



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.

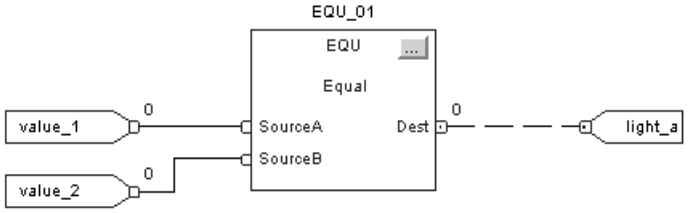
Example: If *value_1* is equal to *value_2*, set *light_a*. If *value_1* is not equal to *value_2*, clear *light_a*.

Relay Ladder

Structured Text

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

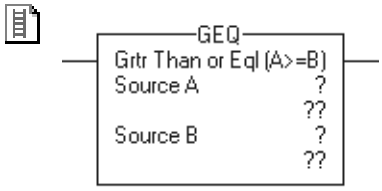
Function Block



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

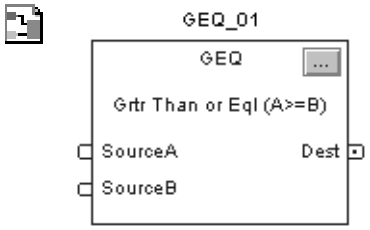
- If you enter a SINT or INT tag, the value converts to a DINT value by sign-extension.
- String data types are:
  - default STRING data type.
  - any new string data type that you create.
- To test the characters of a string, enter a string tag for both Source A and Source B.

Structured Text

IF sourceA >= sourceB THEN

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



Function Block

Operand	Type	Format	Description
GEQ tag	FBD_COMPARE	Structure	GEQ structure

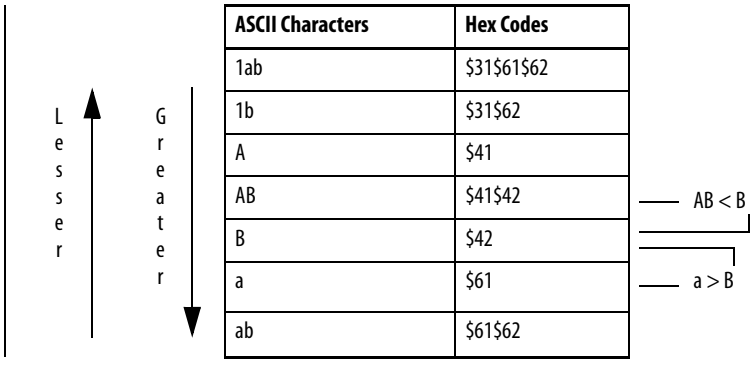
FBD_COMPARE Structure

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, note the following:

- 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

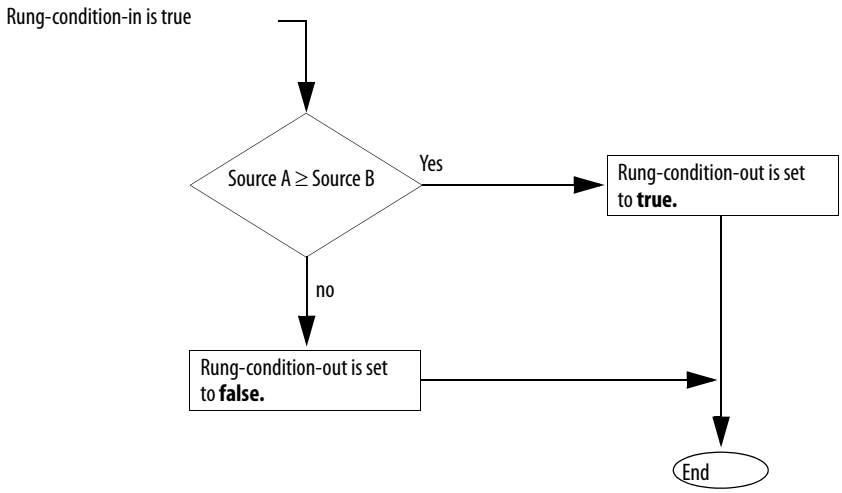
Arithmetic Status Flags: Not affected

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.



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.

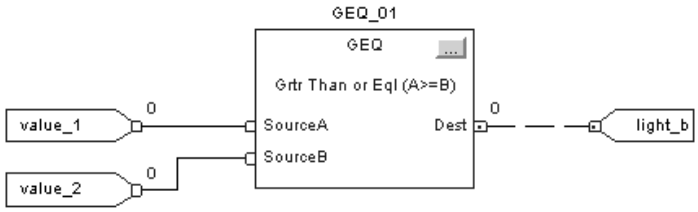
Example: If *value_1* is greater than or equal to *value_2*, set *light_b*. If *value_1* is less than *value_2*, clear *light_b*.

Relay Ladder

Structured Text

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

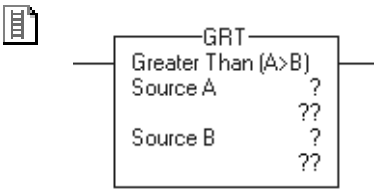
Function Block



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

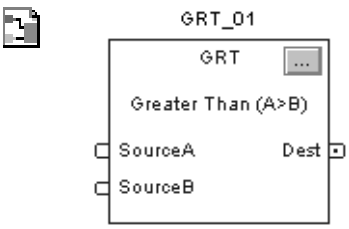
- If you enter a SINT or INT tag, the value converts to a DINT value by sign-extension.
- String data types are:
  - default STRING data type.
  - any new string data type that you create.
- To test the characters of a string, enter a string tag for both Source A and Source B.

Structured Text

IF sourceA > sourceB THEN

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

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



Function Block

Operand	Type	Format	Description
GRT tag	FBD_COMPARE	Structure	GRT structure

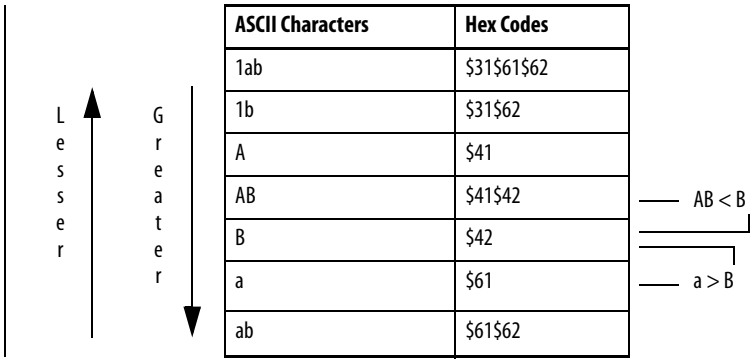
FBD_COMPARE Structure

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, note the following:

- 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

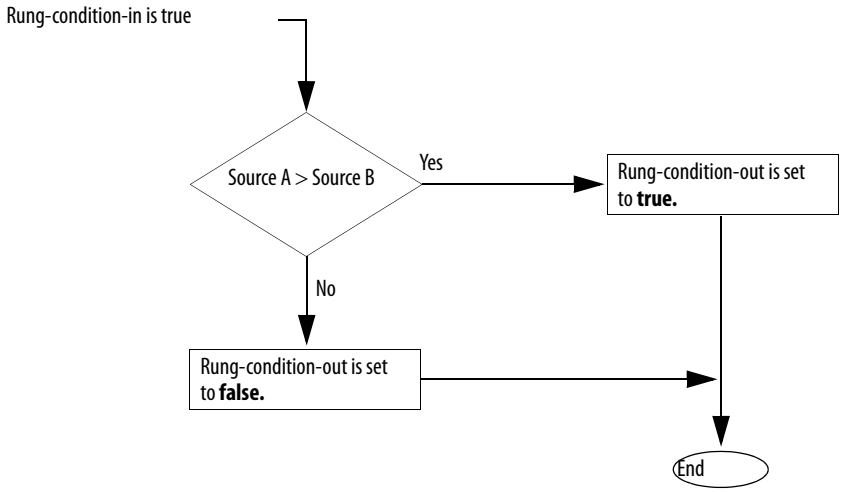
Arithmetic Status Flags: Not affected

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.



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.

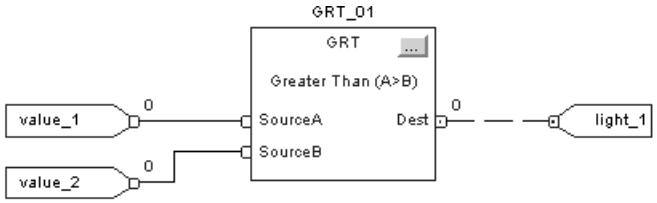
Example: If *value_1* is greater than *value_2*, set *light_1*. If *value_1* is less than or equal to *value_2*, clear *light_1*.

Relay Ladder

Structured Text

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

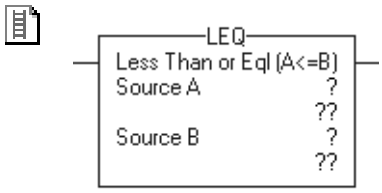
Function Block



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

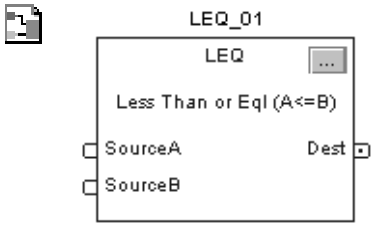
- If you enter a SINT or INT tag, the value converts to a DINT value by sign-extension.
- String data types are:
  - default STRING data type.
  - any new string data type that you create.
- To test the characters of a string, enter a string tag for both Source A and Source B.

Structured Text

IF sourceA <= sourceB THEN

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



Function Block

Operand	Type	Format	Description
LEQ tag	FBD_COMPARE	Structure	LEQ structure

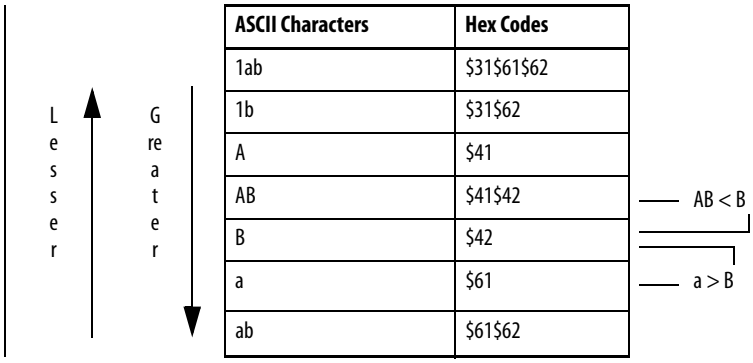
FBD_COMPARE Structure

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, note the following:

- 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

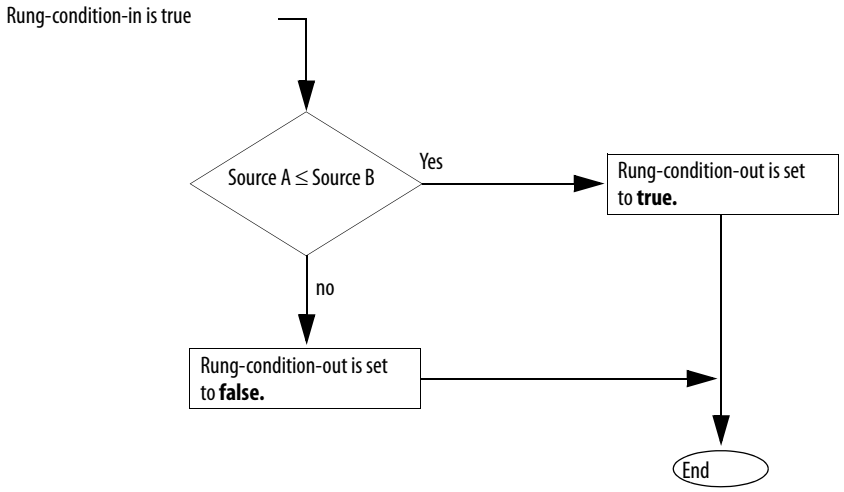
Arithmetic Status Flags: Not affected

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.



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.

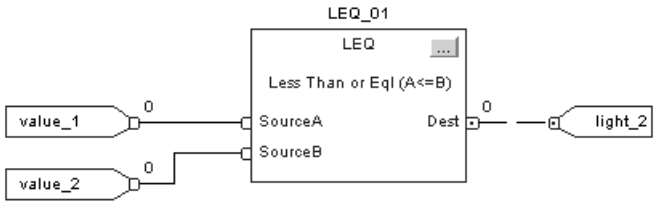
Example: If *value_1* is less than or equal to *value_2*, set *light_2*. If *value_1* is greater than *value_2*, clear *light_2*.

Relay Ladder

Structured Text

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

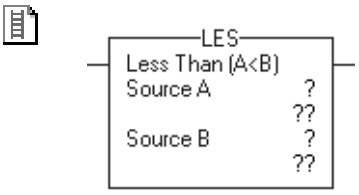
Function Block



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.

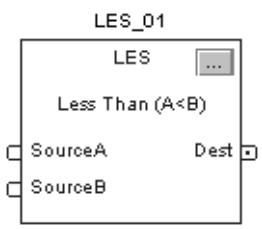
Structured Text

IF sourceA < sourceB THEN

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

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

Function Block



Operand	Type	Format	Description
LES tag	FBD_COMPARE	Structure	LES structure

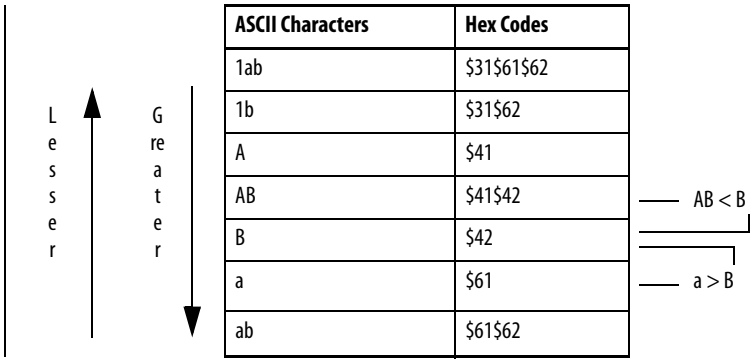
FBD_COMPARE Structure

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, note the following:

- 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

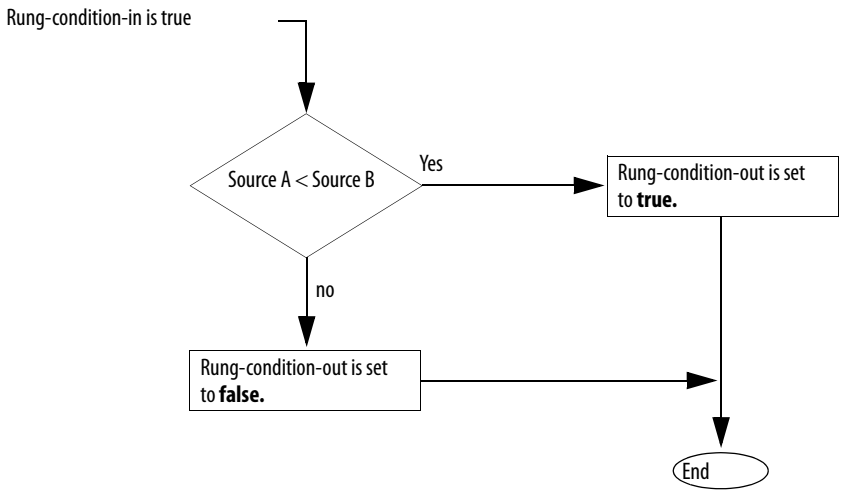
Arithmetic Status Flags: Not affected

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.



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 false	EnableOut is cleared.
EnableIn is true	The instruction executes. EnableOut is set.
Postscan	No action taken.

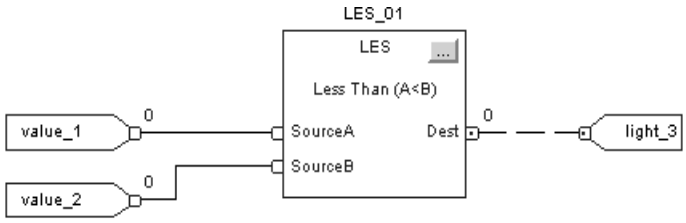
Example: If *value_1* is less than *value_2*, set *light_3*. If *value_1* is greater than or equal to *value_2*, clear *light_3*.

Relay Ladder

Structured Text

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

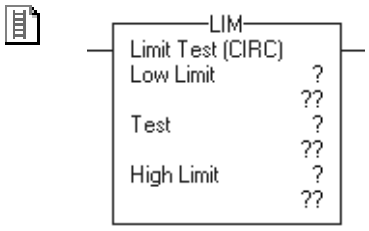
Function Block



Limit (LIM)

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

Operands:



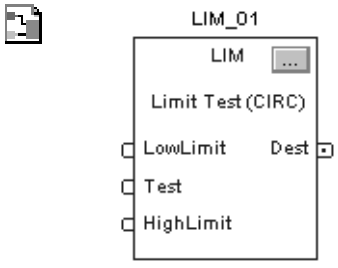
Relay Ladder

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 by 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

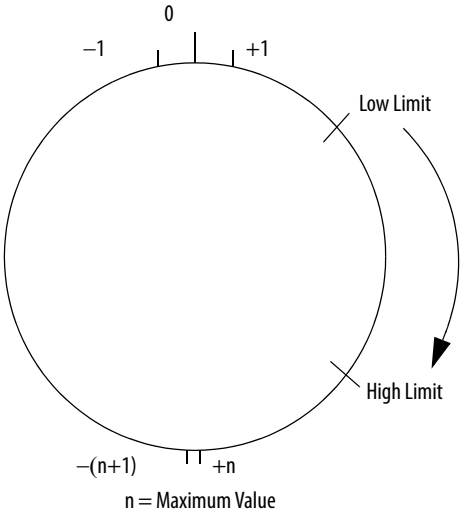
Description: The LIM instruction tests whether the Test value is within the range of the Low Limit to the High Limit.

If Low Limit	And Test Value Is	The Rung-condition-out Is
≤ High Limit	Equal to or between limits	True
	Not equal to or outside limits	False
≥ High Limit	Equal to or outside limits	True
	Not equal to or inside limits	False

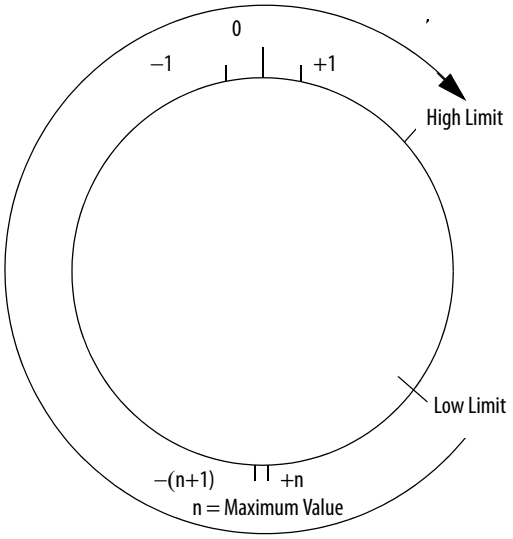
Signed integers 'roll over' from the maximum positive number to the maximum negative number when the most significant bit is set. For example, in 16-bit integers (INT type), the maximum positive integer is 32,767, which is represented in hexadecimal as 16#7FFF (bits 0 ...14 are all set). If you increment 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 ≤ High Limit
The instruction is true if the test value is equal to or between the low and high limit



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



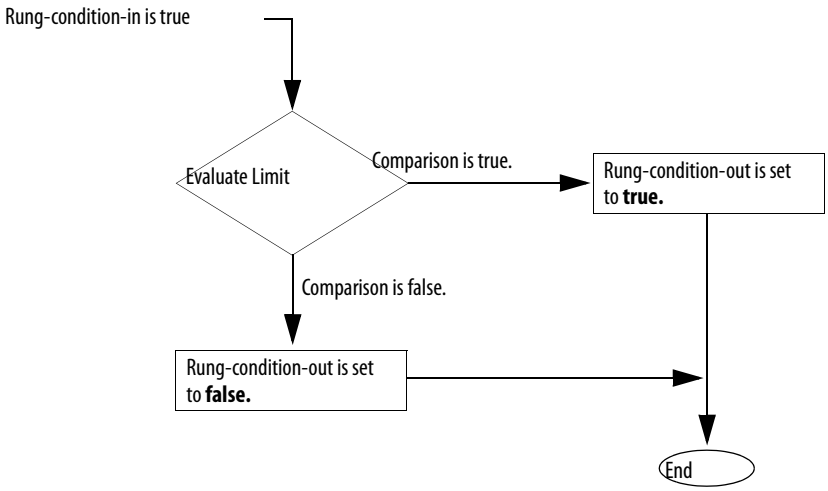
Arithmetic Status Flags: Not affected

Fault Conditions: None

Execution:

Relay Ladder

Condition	Action
Prescan	The rung-condition-out is set to false.
Rung-condition-in is false	The rung-condition-out is set to false.



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, 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:

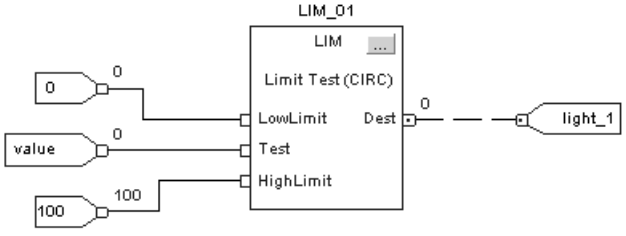
When 0 ≤ *value* ≥ 100, set *light_1*. If *value* < 0 or *value* >100, clear *light_1*.

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 ≥ High Limit:

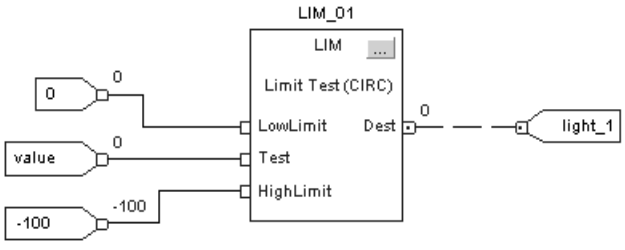
When *value* ≥ 0 or *value* ≤ –**100**, set *light_1*. If *value* < 0 or *value* >–100, clear *light_1*.

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;
```

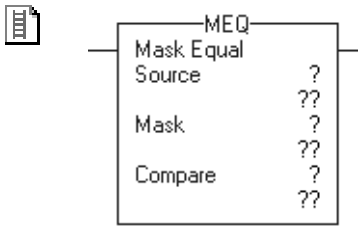
Function Block



Mask Equal To (MEQ)

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

Operands:



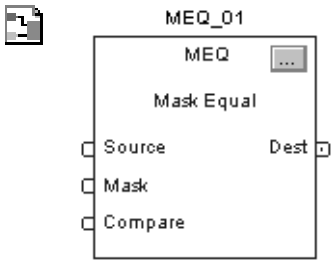
Relay Ladder

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 by 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 by 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

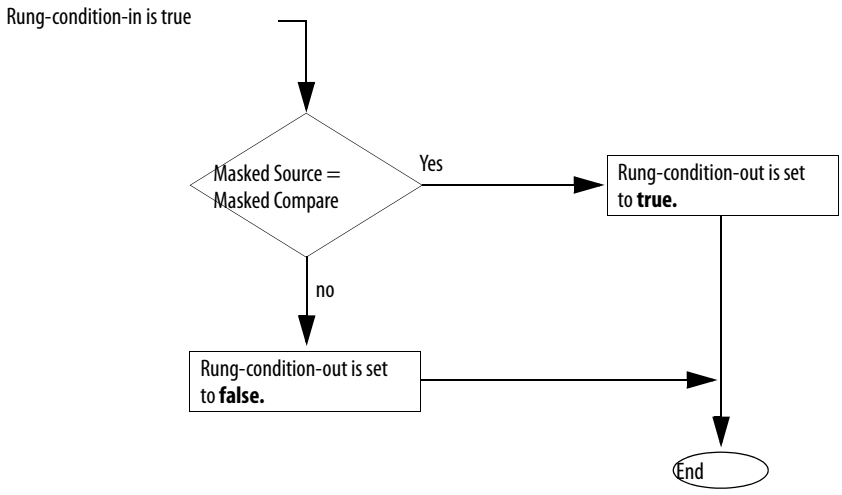
Arithmetic Status Flags: Not affected

Fault Conditions: None

Execution:

Relay Ladder

Condition	Action
Prescan	The rung-condition-out is set to false.
Rung-condition-in is false	The rung-condition-out is set to false.



Postscan	The rung-condition-out is set to false.

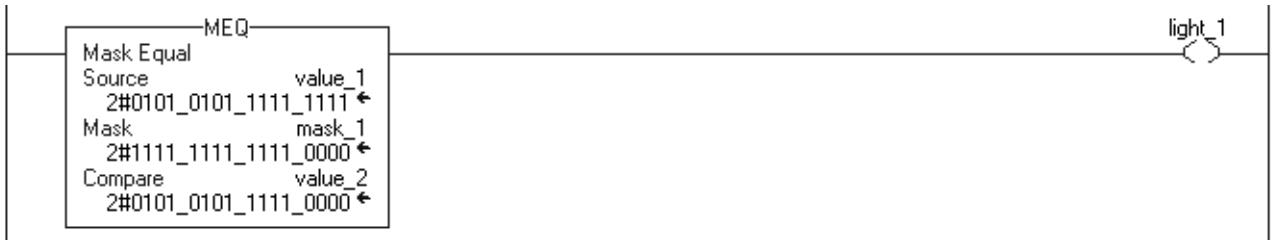
Function Block

Condition	Action
Prescan	No action taken.
Instruction first scan	No action taken.
Instruction first run	No action taken.
EnableIn is cleared	EnableOut is cleared, 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).

value_1	0	1	0	1	0	1	0	1	1	1	1	1	1	1	1	1	value_2	0	1	0	1	0	1	0	1	1	1	1	1	0	0	0	0
mask_1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	mask_1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
masked value_1	0	1	0	1	0	1	0	1	1	1	1	1	x	x	x	x	masked value_2	0	1	0	1	0	1	0	1	1	1	1	1	x	x	x	x

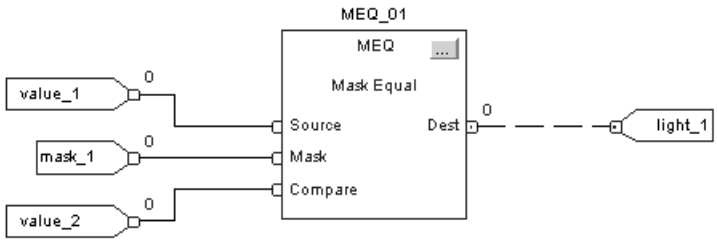
Relay Ladder



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).

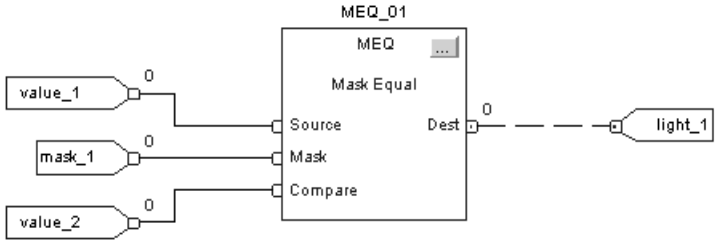
value_1	0	1	0	1	0	1	0	1	1	1	1	1	1	1	1	1	value_2	0	1	0	1	0	1	0	1	1	1	1	1	0	0	0	0
mask_1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	mask_1	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	
masked value_1	x	x	x	x	x	x	x	x	x	x	x	x	1	1	1	1	masked value_2	x	x	x	x	x	x	x	x	x	x	x	0	0	0	0	

Relay Ladder

Structured Text

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

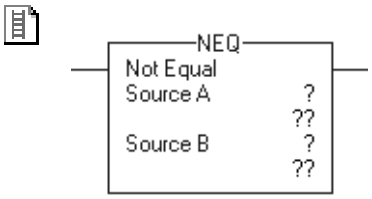
Function Block



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

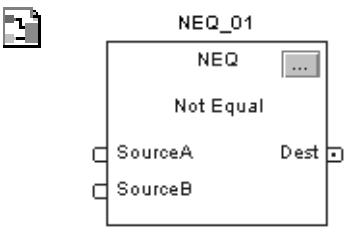
- If you enter a SINT or INT tag, the value converts to a DINT value by sign-extension.
- String data types are:
  - default STRING data type.
  - any new string data type that you create.
- To test the characters of a string, enter a string tag for both Source A and Source B.

Structured Text

IF sourceA <> sourceB THEN

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



Function Block

Operand	Type	Format	Description
NEQ tag	FBD_COMPARE	Structure	NEQ structure

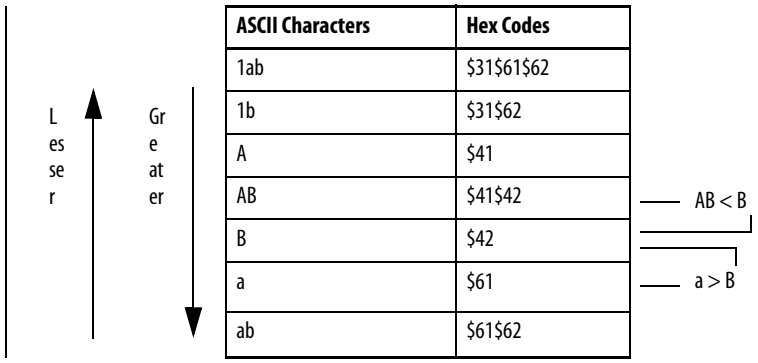
FBD_COMPARE Structure

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

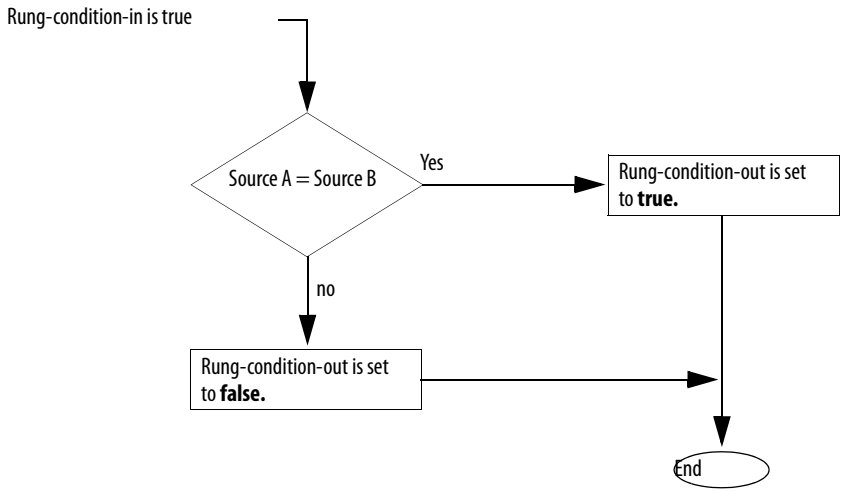
Arithmetic Status Flags: Not affected

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.



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.

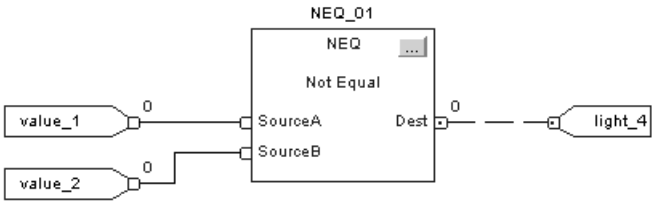
Example: If *value_1* is not equal to *value_2*, set *light_4*. If *value_1* is equal to *value_2*, clear *light_4*.

Relay Ladder

Structured Text

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

Function Block



Compute/Math Instructions

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

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

If you want to	Use this instruction	Available in these languages	Page
Evaluate an expression	CPT	Relay ladder Structured text(1)	257
Add two values	ADD	Relay ladder Structured text(2) Function block	261
Subtract two values	SUB	Relay ladder Structured text(2) Function block	265
Multiply two values	MUL	Relay ladder Structured text(2) Function block	268
Divide two values	DIV	Relay ladder Structured text(2) Function block	271
Determine the remainder after one value is divided by another	MOD	Relay ladder Structured text(2) Function block	276

If you want to	Use this instruction	Available in these languages	Page
Calculate the square root of a value	SQR SQRT(3)	Relay ladder Structured text Function block	280
Take the opposite sign of a value	NEG	Relay ladder Structured text(2) Function block	283
Take the absolute value of a value	ABS	Relay ladder Structured text Function block	286

(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.

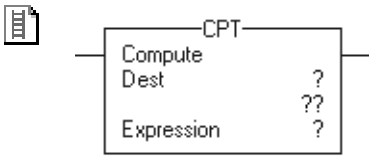
For relay ladder instructions, **bold** data types indicate optimal data types. An instruction executes faster and requires less memory if all the operands of the instruction use the same optimal data type, typically DINT or REAL.

Compute (CPT)

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

IMPORTANT	If x**y is used, all variables (including constants) will be converted to REALs. This can result in the Destination being rounded or the loss of precision/ resolution.

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 by using an assignment and expression.

destination := *numeric_expression*;

See Structured Text Programming 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.

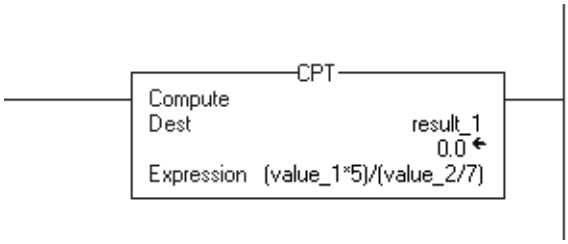
Arithmetic Status Flags: Arithmetic status flags are affected.

Fault Conditions: None

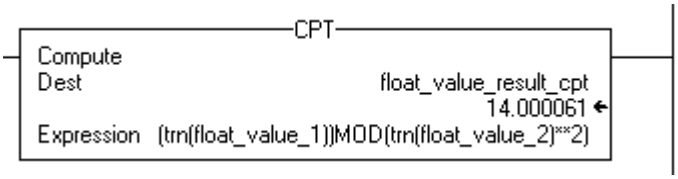
Execution:

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
LOG	log base 10	REAL

Operator	Description	Optimal
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

Format 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	•*tag_b* + 5 •*tag_c* AND *tag_d* •(*tag_e* ** 2) MOD (*tag_f* / *tag_g*)

Determine 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.

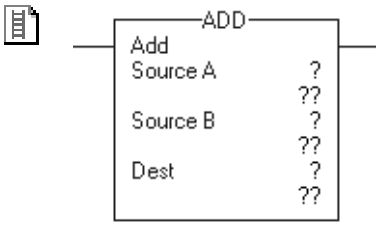
Operations of equal order are performed from left to right.

Order	Operation
1.	( )
2.	ABS, ACS, ASN, ATN, COS, DEG, FRD, LN, LOG, RAD, SIN, SQR, TAN, TOD, TRN
3.	** **IMPORTANT:** If x**y is used, the CPT instruction will be calculated using REALs. This may result in the Destination being rounded or the loss of precision/resolution.
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

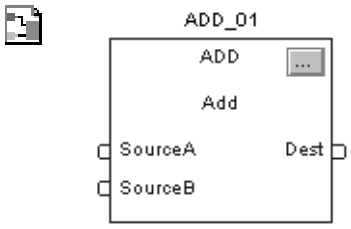
Operand	Type	Format	Description
Source A	SINT INT **DINT** **REAL**	Immediate Tag	Value to add to Source B
	A SINT or INT tag converts to a DINT value by sign-extension.		
Source B	SINT INT **DINT** **REAL**	Immediate Tag	Value to add to Source A
	A SINT or INT tag converts to a DINT value by sign-extension.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

Structured Text

```
dest := sourceA + sourceB;
```

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

See Structured Text Programming 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

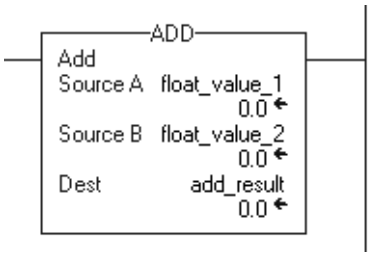
Condition	Action
Prescan	The rung-condition-out is set to false.
Rung-condition-in is false	The rung-condition-out is set to false.
Rung-condition-in is true	Destination = Source A + Source B The rung-condition-out is set to true.
Postscan	The rung-condition-out is set to false.

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*.

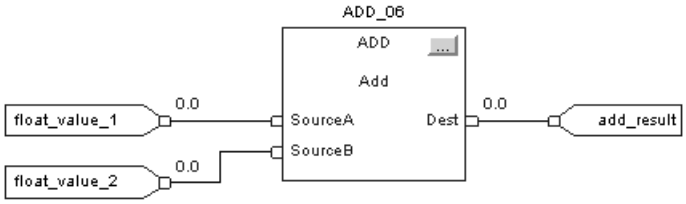
Relay Ladder



Structured Text

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

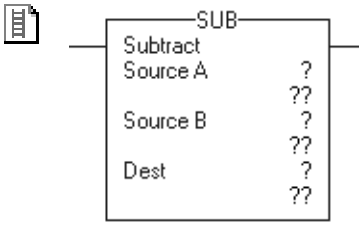
Function Block



Subtract (SUB)

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

Operands:



Relay Ladder

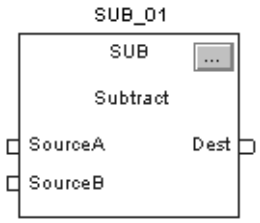
Operand	Type	Format	Description
Source A	SINT INT **DINT** **REAL**	Immediate Tag	Value from which to subtract Source B
	A SINT or INT tag converts to a DINT value by sign-extension.		
Source B	SINT INT **DINT** **REAL**	Immediate Tag	Value to subtract from Source A
	A SINT or INT tag converts to a DINT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

Structured Text

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

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

See Structured Text Programming 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

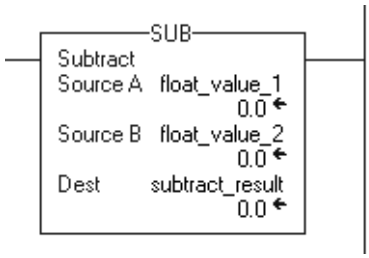
Condition	Action
Prescan	The rung-condition-out is set to false.
Rung-condition-in is false	The rung-condition-out is set to false.
Rung-condition-in is true	Destination = Source B - Source A The rung-condition-out is set to true.
Postscan	The rung-condition-out is set to false.

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*.

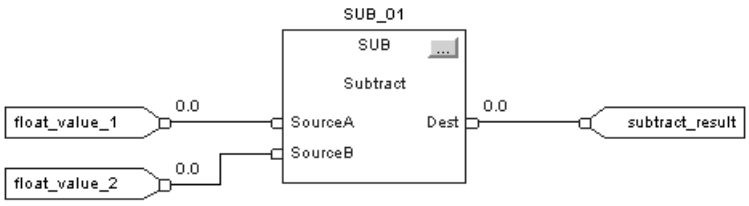
Relay Ladder



Structured Text

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

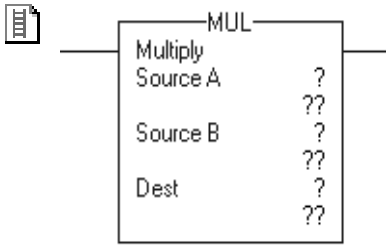
Function Block



Multiply (MUL)

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

Operands:



Relay Ladder

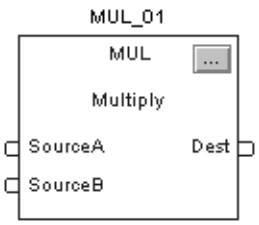
Operand	Type	Format	Description
Source A	SINT INT **DINT** **REAL**	Immediate Tag	Value of the multiplicand
	A SINT or INT tag converts to a DINT value by sign-extension.		
Source B	SINT INT **DINT** **REAL**	Immediate Tag	Value of the multiplier
	A SINT or INT tag converts to a DINT value by sign-extension.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

Structured Text

dest := sourceA * sourceB;

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

See Structured Text Programming 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

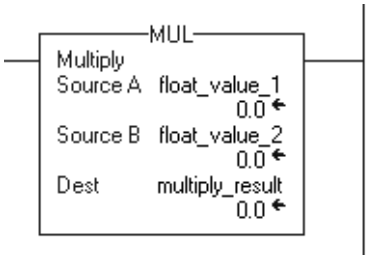
Condition	Action
Prescan	The rung-condition-out is set to false.
Rung-condition-in is false	The rung-condition-out is set to false.
Rung-condition-in is true	Destination = Source B x Source A The rung-condition-out is set to true.
Postscan	The rung-condition-out is set to false.

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*.

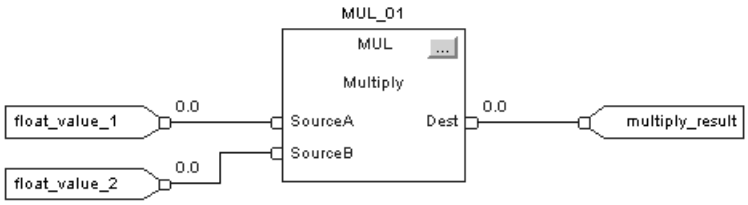
Relay Ladder



Structured Text

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

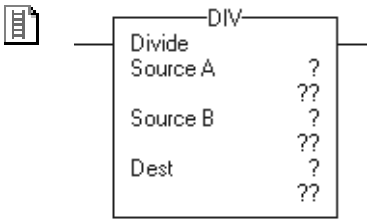
Function Block



Divide (DIV)

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

Operands:



Relay Ladder

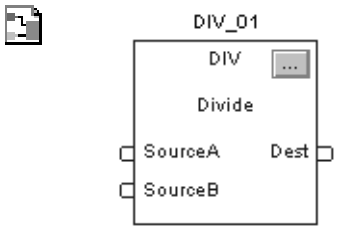
Operand	Type	Format	Description
Source A	SINT INT **DINT** **REAL**	Immediate Tag	Value of the dividend
	A SINT or INT tag converts to a DINT value by sign-extension.		
Source B	SINT INT **DINT** **REAL**	Immediate Tag	Value of the divisor
	A SINT or INT tag converts to a DINT value by sign-extension.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

Structured Text

dest := sourceA / sourceB;

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

See Structured Text Programming 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 *not* 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

The destination is set as follows.

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

If source B is zero and	And the destination is a	And the result is	Then the destination is set to
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 Programming Manual, 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

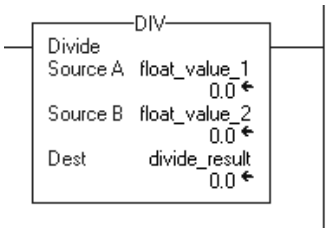
Condition	Action
Prescan	The rung-condition-out is set to false.
Rung-condition-in is false	The rung-condition-out is set to false.
Rung-condition-in is true	Destination = Source A / Source B The rung-condition-out is set to true.
Postscan	The rung-condition-out is set to false.

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*.

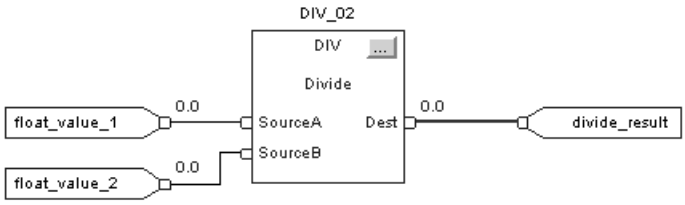
Relay Ladder



Structured Text

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

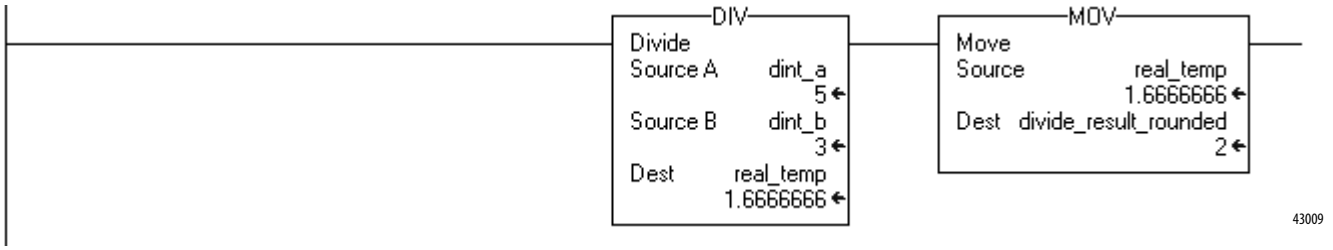
Function Block



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.

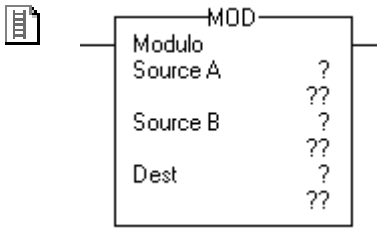
Relay Ladder



Modulo (MOD)

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

Operands:



Relay Ladder

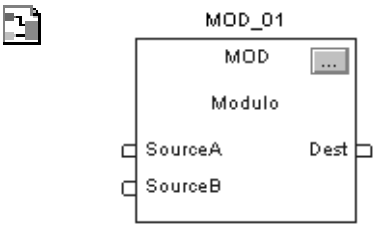
Operand	Type	Format	Description
Source A	SINT INT **DINT** **REAL**	Immediate Tag	Value of the dividend
	A SINT or INT tag converts to a DINT value by sign-extension.		
Source B	SINT INT **DINT** **REAL**	Immediate Tag	Value of the divisor
	A SINT or INT tag converts to a DINT value by sign-extension.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

Structured Text

dest := sourceA MOD sourceB;

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

See Structured Text Programming 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, the following happens:

- 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 The S:N and S:Z flags are not set.
		Negative	0
	REAL	Positive	1.$ (positive infinity)
		Negative	-1.$ (negative infinity)

IMPORTANT The MOD instruction does not capture an overflow that may have occurred during the calculation. If an overflow occurs, a minor fault will be generated, but the S:V bit will not be set.

To detect a possible divide-by-zero, examine the minor fault bit (S:MINOR).

See Logix5000 Controllers Common Procedures Programming Manual, 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

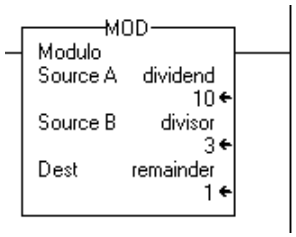
Condition	Action
Prescan	The rung-condition-out is set to false.
Rung-condition-in is false	The rung-condition-out is set to false.
Rung-condition-in is true	Destination = Source A – (TRN (Source A / Source B) * Source B) The rung-condition-out is set to true.
Postscan	The rung-condition-out is set to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

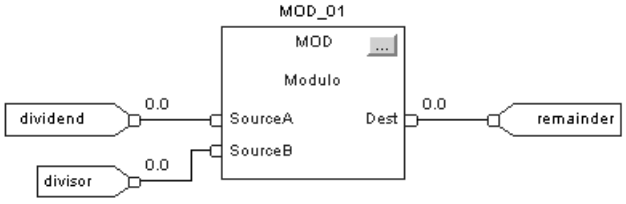
Relay Ladder



Structured Text

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

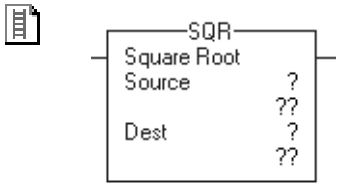
Function Block



Square Root (SQR)

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

Operands:



Relay Ladder

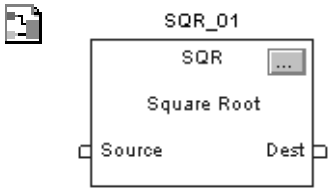
Operand	Type	Format	Description
Source	SINT INT **DINT** **REAL**	Immediate Tag	Find the square root of this value
	A SINT or INT tag converts to a DINT value by sign-extension.		
Destination	SINT INT **DINT** **REAL**	Tag	Tag to store the result

Structured Text

dest := SQRT(source);

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

See Structured Text Programming 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		
Not 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

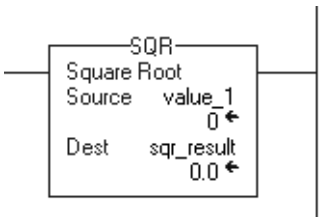
Condition	Action
Prescan	The rung-condition-out is set to false.
Rung-condition-in is false	The rung-condition-out is set to false.
Rung-condition-in is true	$Destination = \sqrt{Source}$ The rung-condition-out is set to true.
Postscan	The rung-condition-out is set to false.

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*.

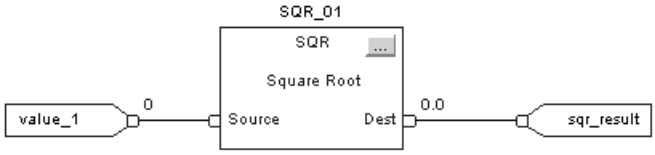
Relay Ladder



Structured Text

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

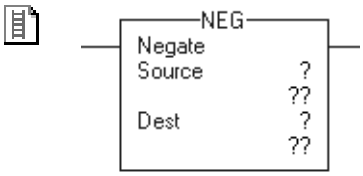
Function Block



Negate (NEG)

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

Operands:



Relay Ladder

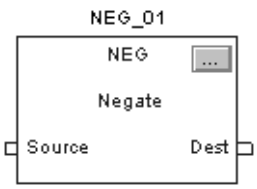
Operand	Type	Format	Description
Source	SINT INT **DINT** **REAL**	Immediate Tag	Value to negate
	A SINT or INT tag converts to a DINT value by sign-extension.		
Destination	SINT INT **DINT** **REAL**	Tag	Tag to store the result

Structured Text

dest := -source;

Use the minus sign '–' as an operator in an expression. This expression changes the sign of *source* and stores the result in *dest*.

See Structured Text Programming 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

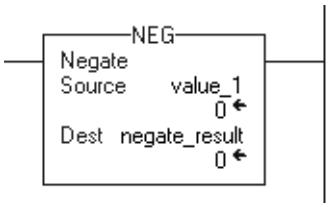
Condition	Action
Prescan	The rung-condition-out is set to false.
Rung-condition-in is false	The rung-condition-out is set to false.
Rung-condition-in is true	Destination = 0 – Source The rung-condition-out is set to true.
Postscan	The rung-condition-out is set to false.

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*.

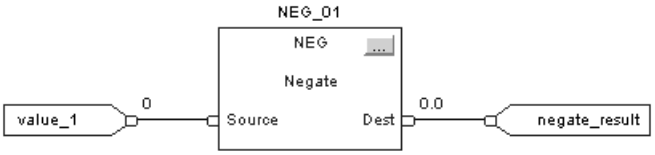
Relay Ladder



Structured Text

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

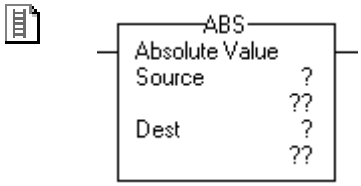
Function Block



Absolute Value (ABS)

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

Operands:



Relay Ladder

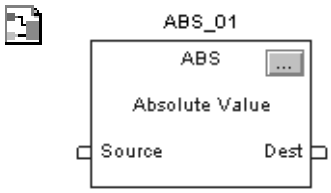
Operand	Type	Format	Description
Source	SINT INT **DINT** **REAL**	Immediate Tag	Value of which to take the absolute value
	A SINT or INT tag converts to a DINT value by sign-extension.		
Destination	SINT INT **DINT** **REAL**	Tag	Tag to store the result

Structured Text

dest := ABS(source);

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

See Structured Text Programming 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

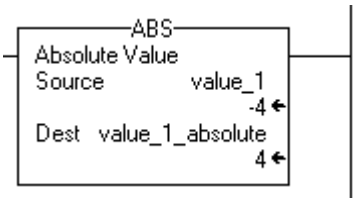
Condition	Action
Prescan	The rung-condition-out is set to false.
Rung-condition-in is false	The rung-condition-out is set to false.
Rung-condition-in is true	Destination = | Source | The rung-condition-out is set to true.
Postscan	The rung-condition-out is set to false.

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.

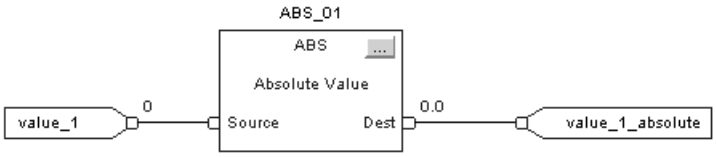
Relay Ladder



Structured Text

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

Function Block



Move/Logical Instructions

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

The move instructions modify and move its, and the logical instructions perform operations on bits.

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	Page
Copy a value	MOV	Relay ladder Structured text(1)	291
Copy a specific part of an integer	MVM	Relay ladder	293
Copy a specific part of an integer in function block	MVMT	Structured text Function block	296
Move bits within an integer or between integers	BTD	Relay ladder	299
Move bits within an integer or between integers in function block	BTDT	Structured text Function block	302
Clear a value	CLR	Structured text(1) Relay ladder	306
Rearrange the bytes of a INT, DINT, or REAL tag	SWPB	Relay ladder Structured text	308

(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	Page
Bitwise AND Operation	Bitwise AND &(1)	Relay ladder Structured text(2) Function block	312
Bitwise OR operation	Bitwise OR	Relay ladder Structured text(2) Function block	316
Bitwise, exclusive OR operation	Bitwise XOR	Relay ladder Structured text(2) Function block	320
Bitwise NOT operation	Bitwise NOT	Relay ladder Structured text(2) Function block	324
Logically AND as many as eight boolean inputs.	Boolean AND (BAND)	Structured text(2) Function block	327
Logically OR as many as eight boolean inputs.	Boolean OR (BOR)	Structured text(2) Function block	330
Perform an exclusive OR on two boolean inputs.	Boolean Exclusive OR (BXOR)	Structured text(2) Function block	333
Complement a boolean input.	Boolean NOT (BNOT)	Structured text(2) Function block	336

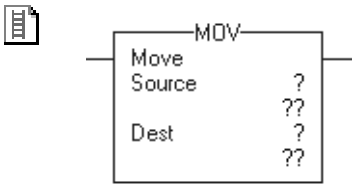
(1) Structured text only.

(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

Structured Text

dest := source;

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.

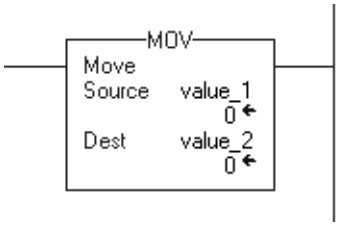
Fault Conditions: None

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

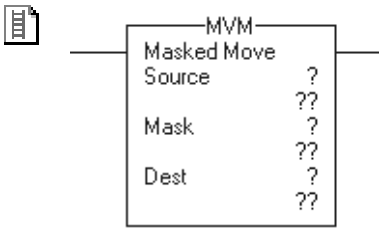
```
value_2 := value _1;
```

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 296.

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

Structured Text

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

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.

Enter an Immediate Mask Value

When you enter a mask, the programming software defaults to decimal values. If you want to enter a mask by 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

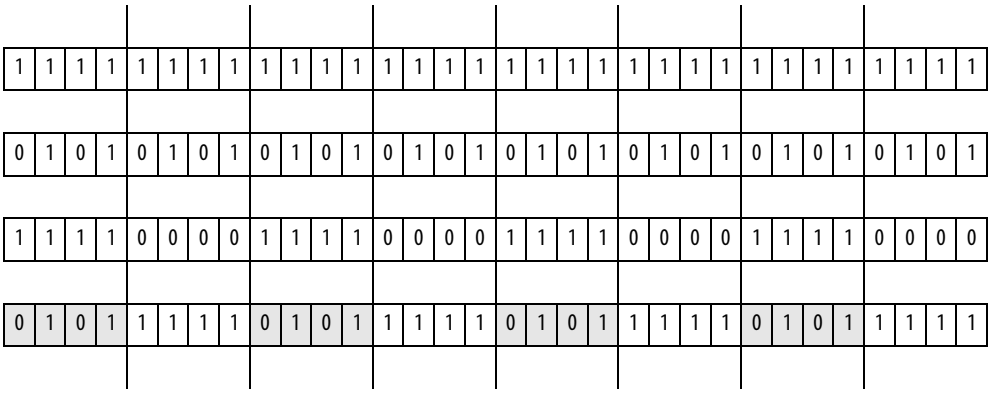
None

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.

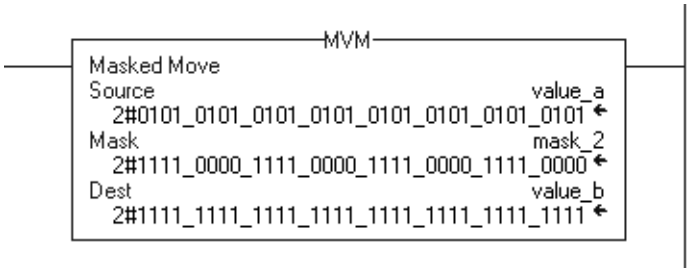
Example:

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.

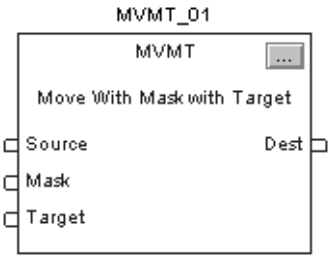
This instruction is available in relay ladder as MVM, see page 293.

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.

Enter an Immediate Mask Value by Using an Input Reference

When you enter a mask, the programming software defaults to decimal values. If you want to enter a mask by 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.	N/A
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.

Target	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Dest	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

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).

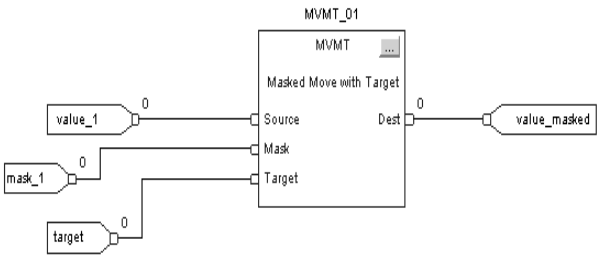
Source	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
Mask1	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
Dest	0	1	0	1	1	1	1	1	0	1	0	1	1	1	1	1	0	1	0	1	1	1	1	1	0	1	0	1	1	1	1	1

The shaded boxes show the bits that changed.

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

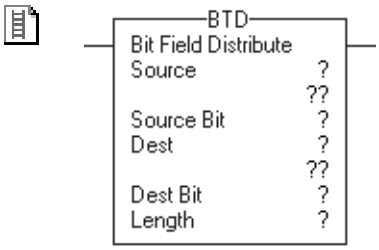


Bit Field Distribute (BTD)

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 302.

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:

Not affected

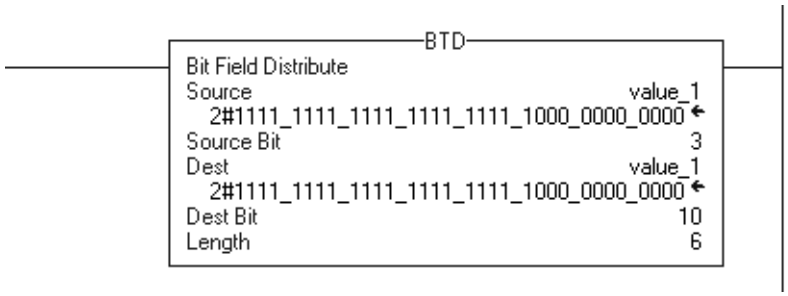
Fault Conditions:

None

Execution:

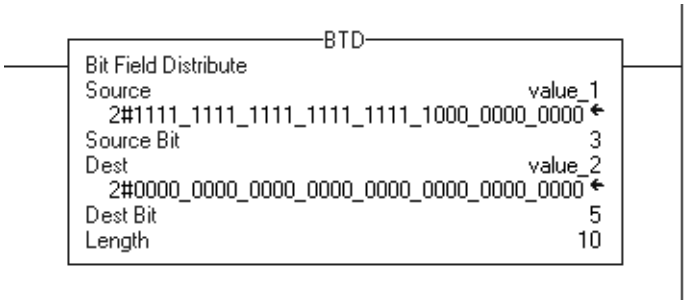
Condition	Relay Ladder Action
Prescan	The rung-condition-out is set to false.
Rung-condition-in is false	The rung-condition-out is set to false.
Rung-condition-in is true	The instruction copies and shifts the Source bits to the Destination. The rung-condition-out is set to true.
Postscan	The rung-condition-out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

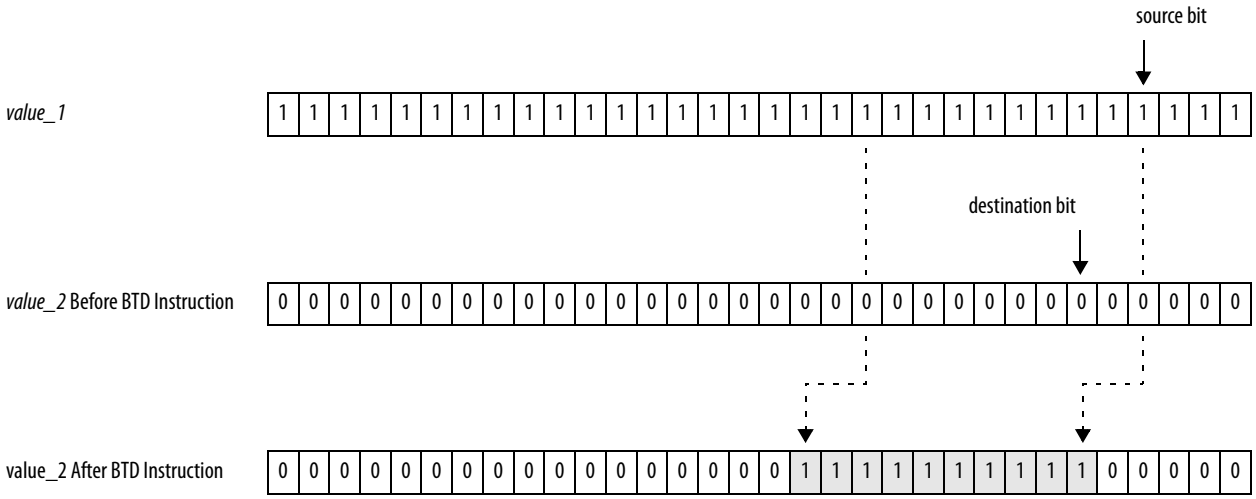
Example 1: When enabled, the BTD instruction moves bits within *value_1*.



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.

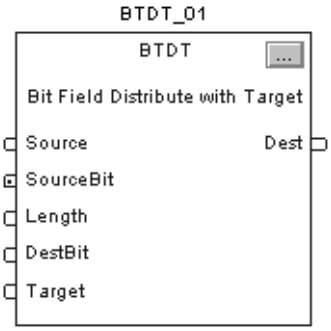
This instruction is available in relay ladder as BTD, see page 299.

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

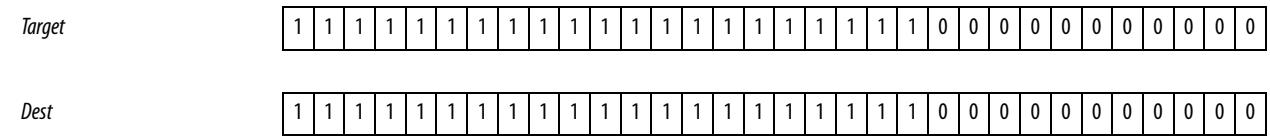
Fault Conditions: None

Execution:

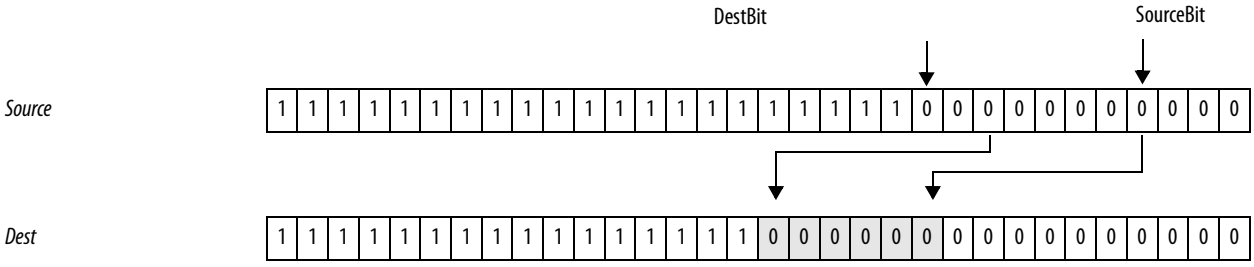
Condition	Function Block Action	Structured Text Action
Prescan	No action taken.	No action taken.
Instruction first scan	No action taken.	No action taken.
Instruction first run	No action taken.	No action taken.
EnableIn is cleared	EnableOut is cleared, the instruction does nothing, and the outputs are not updated.	N/A
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.



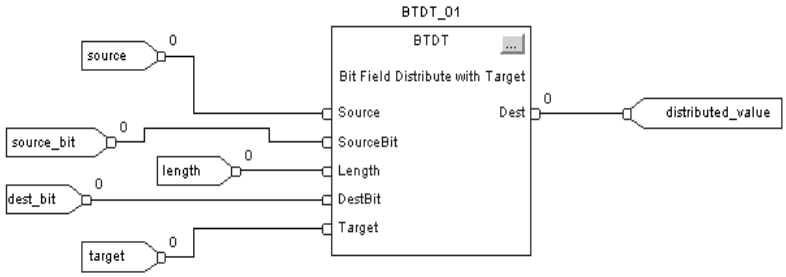
2. The SourceBit and the Length specify which bits in Source to copy into Dest, starting at DestBit. Source and Target remain unchanged.



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;
```

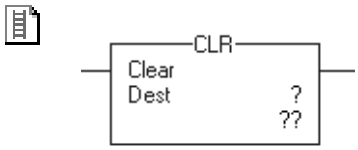
Function Block



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:

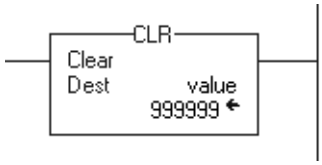
None

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

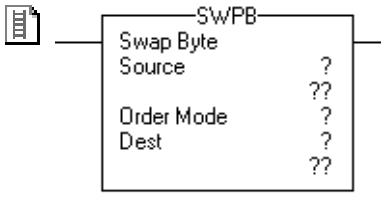
```
value := 0;
```

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	

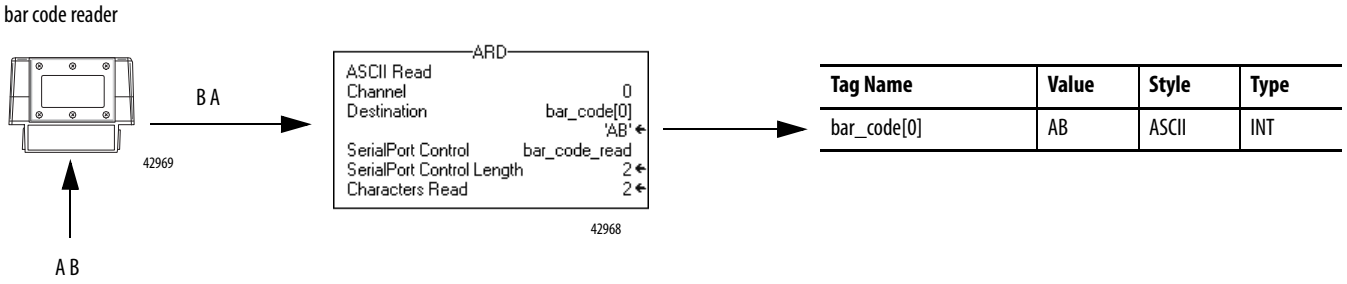
Structured Text

```
SWPB(Source,OrderMode,Dest);
```

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.



Tag Name	Value	Style	Type
bar_code[0]	AB	ASCII	INT

Arithmetic Status Flags: Not affected

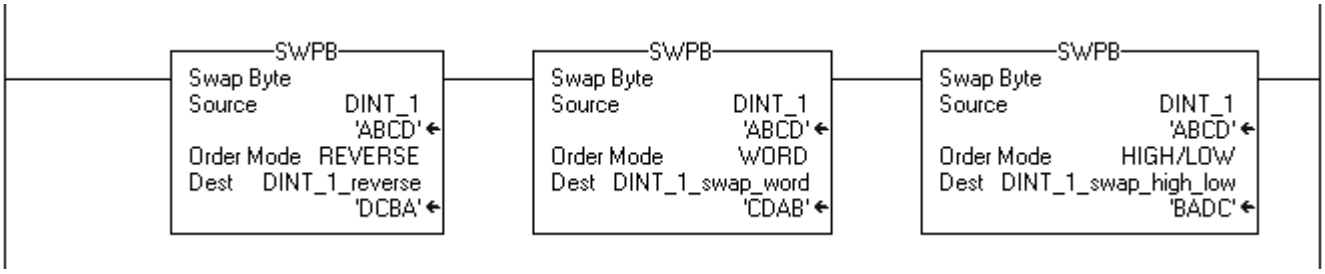
Fault Conditions: None

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.	N/A
Rung-condition-in is true	The rung-condition-out is set to true.	N/A
EnableIn is set	N/A	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*.

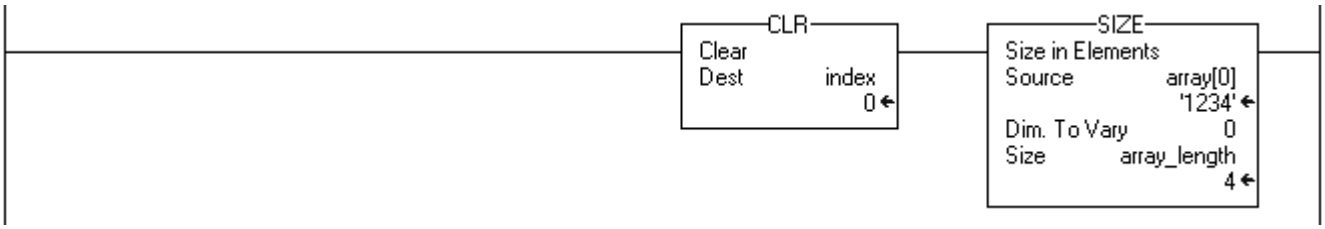
- The SWPB instruction reverses the bytes of the element number that is indicated by the value of *index*. For example, when *index* equals 0, the SWPB instruction acts on *array[0]*.
- The ADD instruction increments *index*. The next time the instruction executes, the SWPB instruction acts on the next element in *array*.

3. Determine when the SWPB instruction has acted on all the elements in the array.

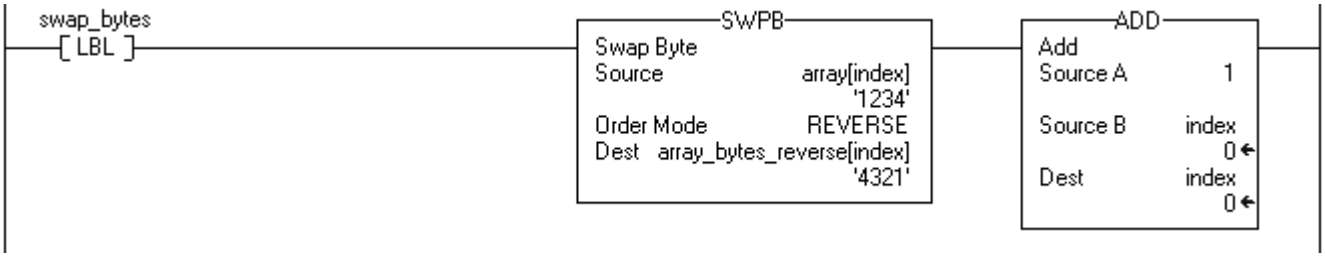
- If *index* is less then the number of elements in the array (*array_length*), then continue with the next element in the array.
- If *index* equals *array_length*, then the SWPB has acted on all the elements in the array.

Relay Ladder

Initialize the tags.



Reverse the bytes.



Determine whether the SWPB instruction has acted on all the elements in the array.

Structured Text

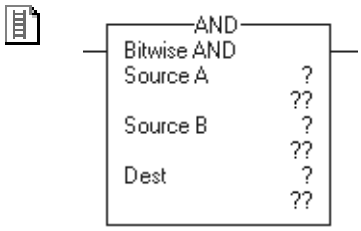
```
index := 0;

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 by using the bits in Source A and Source B and places the result in the Destination.

To perform a logical AND, see page 327.

Operands:



Relay Ladder

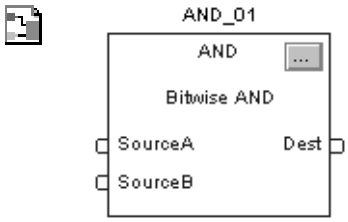
Operand	Type	Format	Description
Source A	SINT INT **DINT**	Immediate Tag	Value to AND with Source B
	A SINT or INT tag converts to a DINT value by zero-fill.		
Source B	SINT INT **DINT**	Immediate Tag	Value to AND with Source A
	A SINT or INT tag converts to a DINT value by zero-fill.		
Destination	SINT 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:

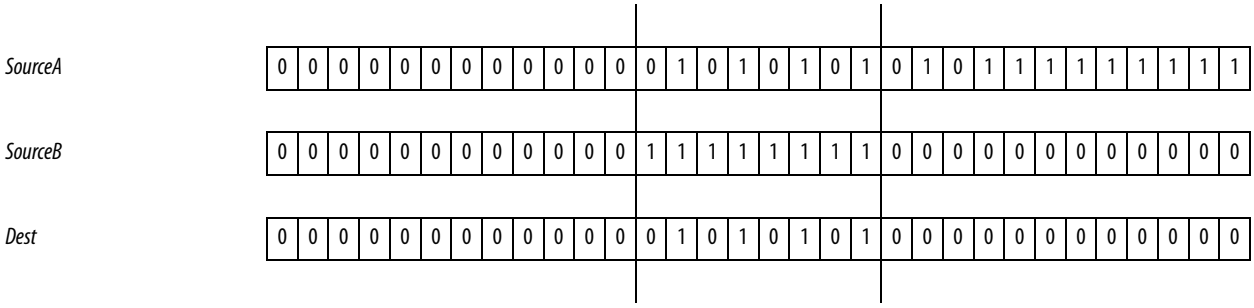
Relay Ladder

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.

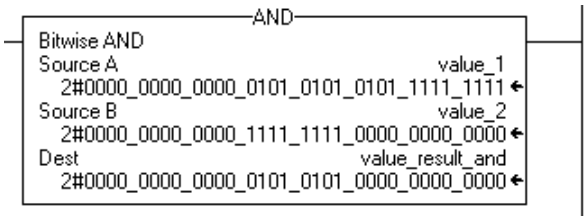
Function Block

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.



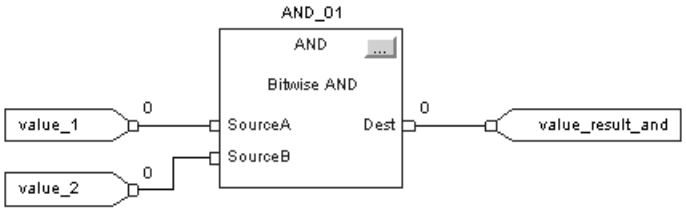
Relay Ladder



Structured Text

value_result_and := value_1 AND value_2;

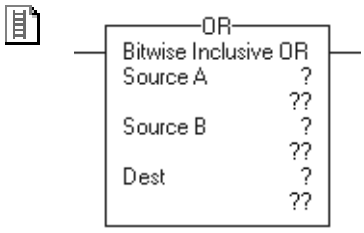
Function Block



Bitwise OR (OR)

The OR instruction performs a bitwise OR operation by using the bits in Source A and Source B and places the result in the Destination.

Operands:



Relay Ladder

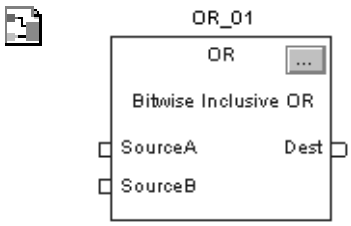
Operand	Type	Format	Description
Source A	SINT INT **DINT**	Immediate Tag	Value to OR with Source B
	A SINT or INT tag converts to a DINT value by zero-fill.		
Source B	SINT INT **DINT**	Immediate Tag	Value to OR with Source A
	A SINT or INT tag converts to a DINT value by zero-fill.		
Destination	SINT INT **DINT**	Tag	Stores the result

Structured Text

dest := sourceA OR sourceB

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:

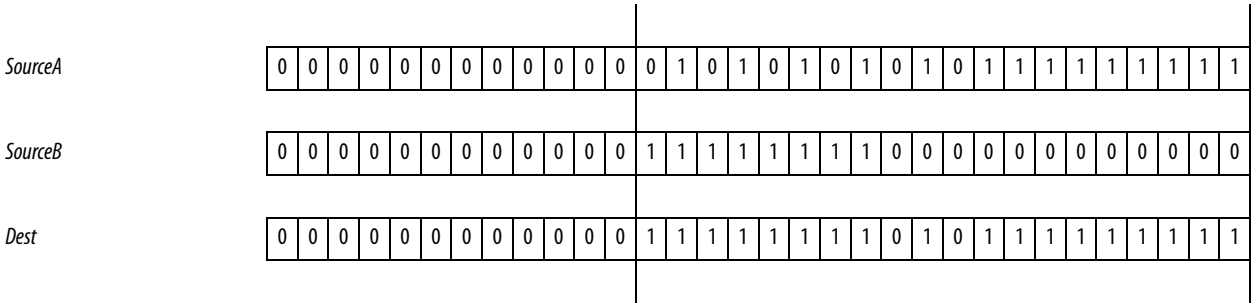
Relay Ladder

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.

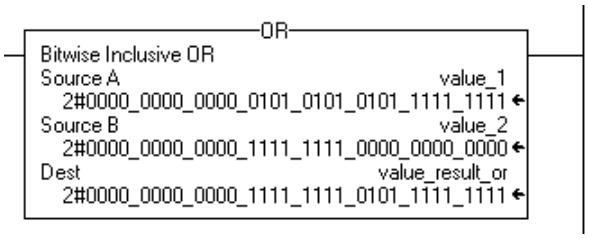
Function Block

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.



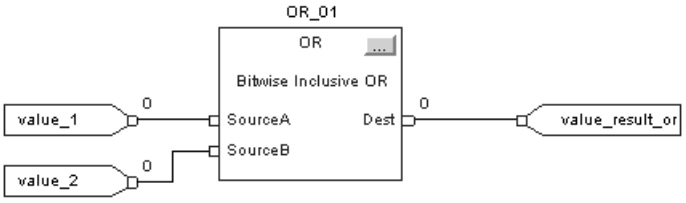
Relay Ladder



Structured Text

value_result_or := value_1 OR value_2;

Function Block

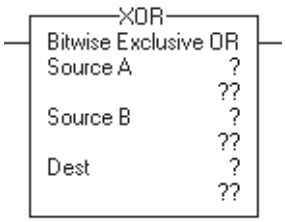


Bitwise Exclusive OR (XOR)

The XOR instruction performs a bitwise XOR operation by using the bits in Source A and Source B and places the result in the Destination.

To perform a logical XOR, see page 333.

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:

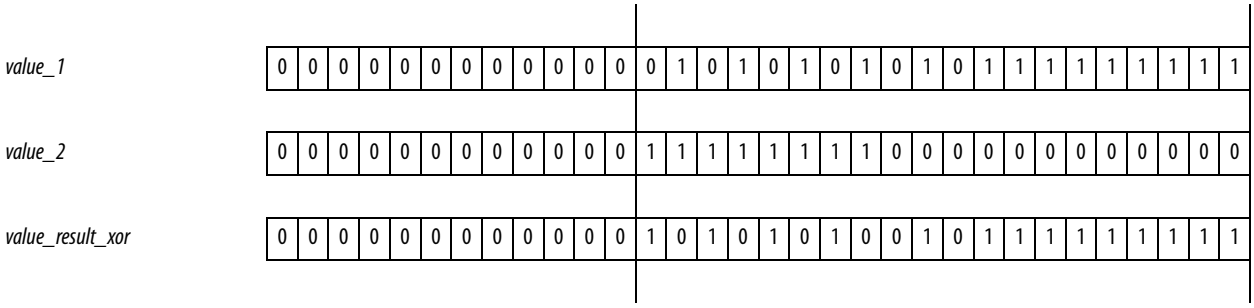
Relay Ladder

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.

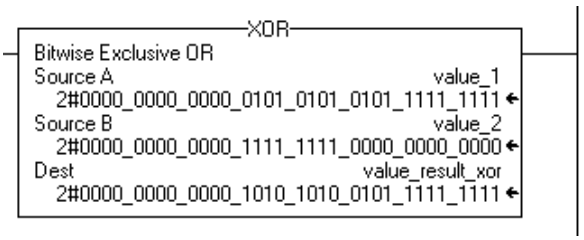
Function Block

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 XOR instruction performs a bitwise XOR operation on SourceA and SourceB and places the result in the destination tag.



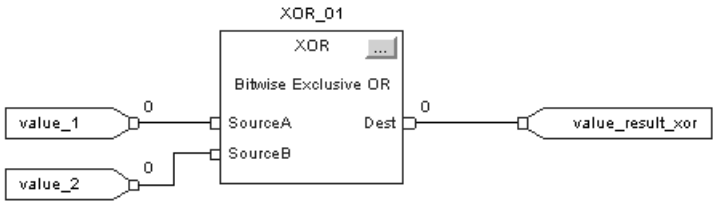
Relay Ladder



Structured Text

value_result_xor := value_1 XOR value_2;

Function Block

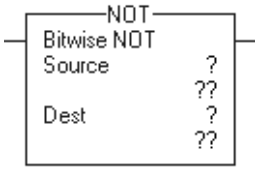


Bitwise NOT (NOT)

The NOT instruction performs a bitwise NOT operation by using the bits in the Source and places the result in the Destination.

To perform a logical NOT, see page 336.

Operands:



Relay Ladder

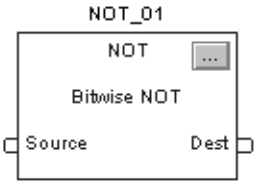
Operand	Type	Format	Description
Source	SINT INT DINT	Immediate Tag	Value to NOT
	A SINT or INT tag converts to a DINT value by zero-fill.		
Destination	SINT INT DINT	Tag	Stores the result

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.

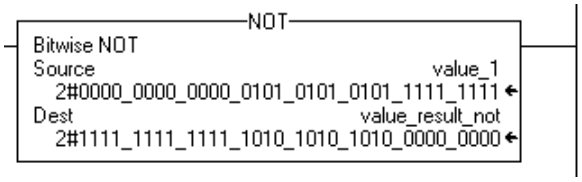
Function Block

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	0	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

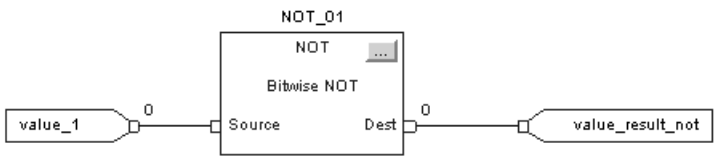
Relay Ladder



Structured Text

```
value_result_not := NOT value_1;
```

Function Block



Boolean AND (BAND)

The BAND instruction logically AND as many as eight boolean inputs.

To perform a bitwise AND, see page 312.

Operands:

Structured Text

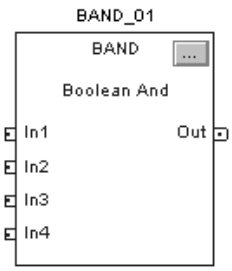
IF operandA AND operandB THEN

<statement>;

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 Appendix B 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.

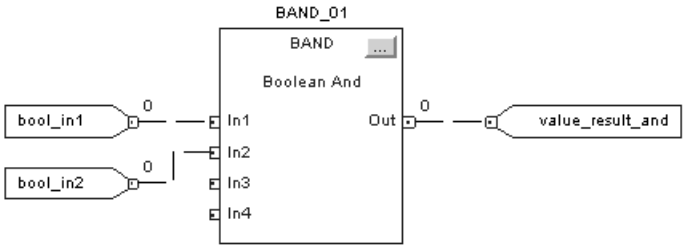
Example 1: This example ANDs *bool_in1* and *bool_in2* and places the result in *value_result_and*.

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 OR as many as eight boolean inputs.

To perform a bitwise OR, see page 316.

Operands:

Structured Text

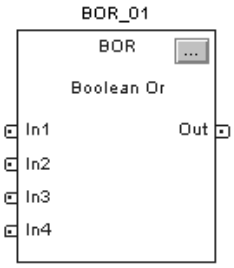
IF operandA OR operandB THEN

<statement>;

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 Appendix B 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.

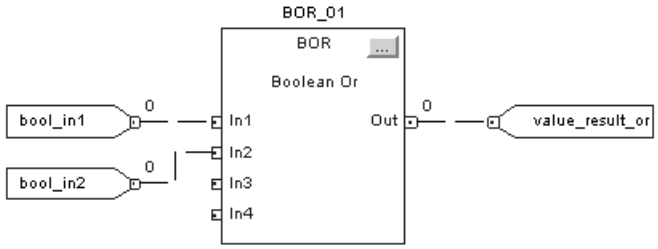
Example 1: This example ORs *bool_in1* and *bool_in2* and places the result in *value_result_or*.

If BOOL_IN1 is	If BOOL_IN2 is	Then VALUE_RESULT_OR 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 the following occurs:

- 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 320.

Operands:

Structured Text

IF operandA XOR operandB THEN

<statement>;

END_IF;

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 Appendix B 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.

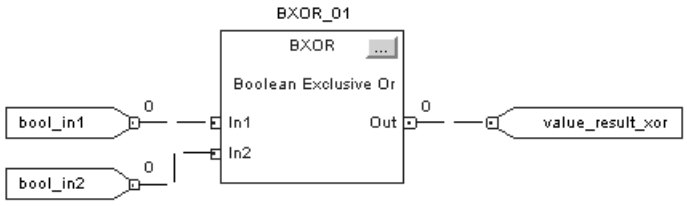
Example 1: This example performs an exclusive OR on *bool_in1* and *bool_in2* and places the result in *value_result_xor*.

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 324.

Operands:

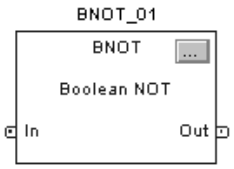
IF NOT operand THEN

<statement>;

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.

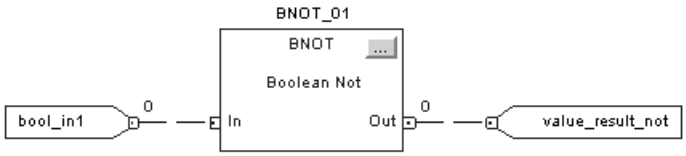
Example 1: This example complements *bool_in1* and places the result in *value_result_not*.

If BOOL_IN1 is	Then VALUE_RESULT_NOT 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)

The file/miscellaneous instructions operate on arrays of data.

If you want to	Use this instruction	Available in these languages	Page
Perform arithmetic, logic, shift, and function operations on values in arrays	FAL	Relay ladder Structured text(1)	345
Search for and compare values in arrays	FSC	Relay ladder	357
Copy the contents of one array into another array	COP	Relay ladder Structured text	365
Copy the contents of one array into another array without interruption	CPS	Relay ladder Structured text	365
Fill an array with specific data	FLL	Relay ladder Structured text(1)	371
Calculate the average of an array of values	AVE	Relay ladder Structured text(1)	375
Sort one dimension of array data into ascending order	SRT	Relay ladder Structured text	380
Calculate the standard deviation of an array of values	STD	Relay ladder Structured text(1)	385
Find the size of a dimension of an array	SIZE	Relay ladder Structured text	392

(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.

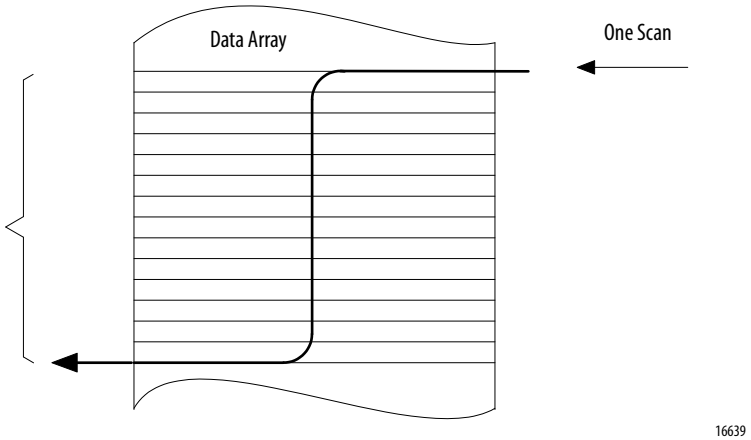
Selecting Mode of Operation

For FAL and FSC instructions, the mode tells the controller how to distribute the array operation.

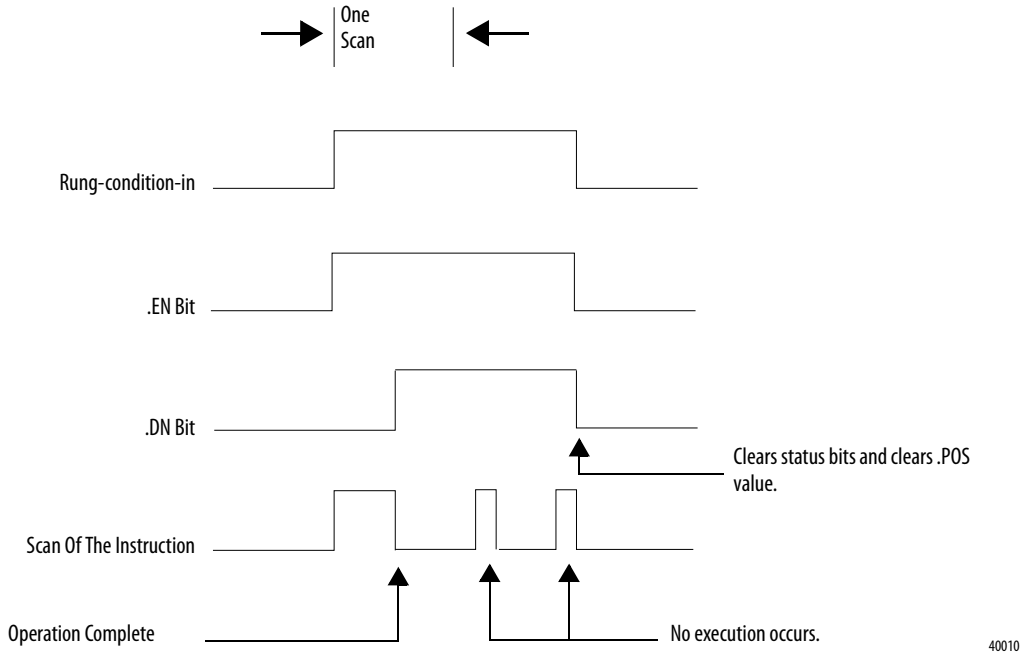
If You Want To	Select This Mode
Operate on all of the specified elements in an array before continuing on to the next instruction	All
Distribute array operation over a number of scans Enter the number of elements to operate on per scan (1-2147483647)	Numerical
Manipulate one element of the array each 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.



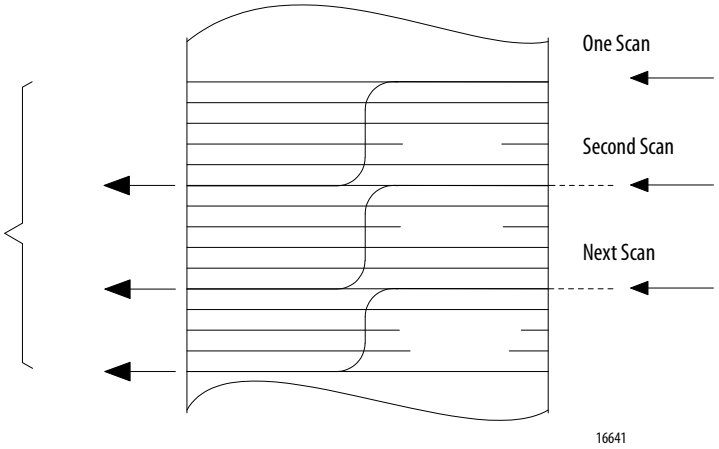
The following timing diagram shows the relationship between status bits and instruction operation. When the instruction execution is complete, the .DN bit is set. The .DN bit, the .EN bit, and the .POS value are cleared when the rung-condition-in is false. Only then can another execution of the instruction be triggered by a false-to-true transition of rung-condition-in.



Numerical Mode

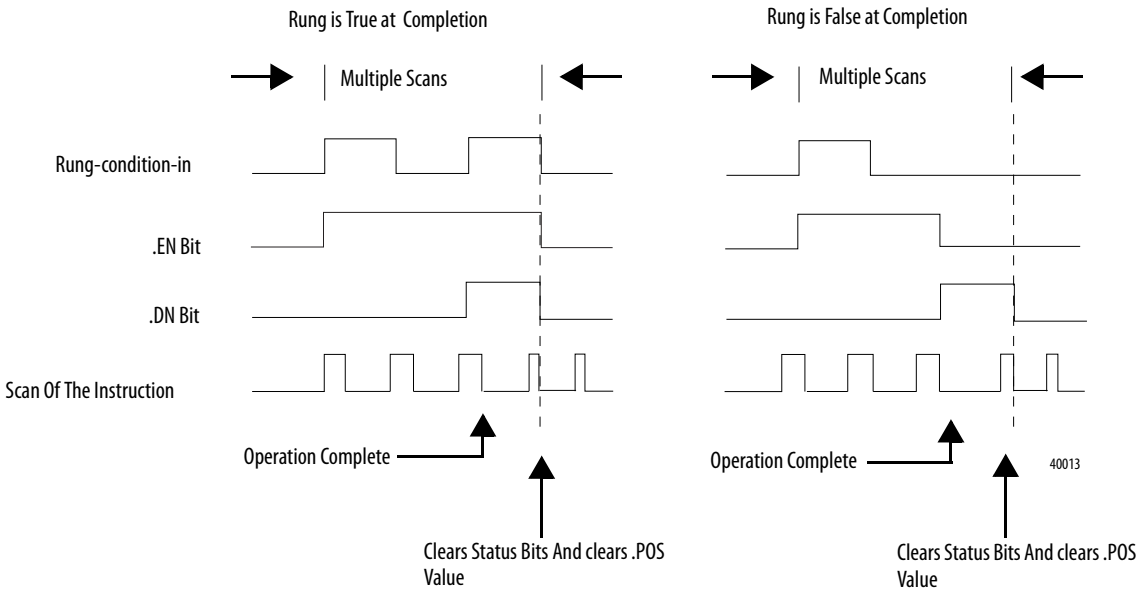
Numerical mode distributes the array operation over a number of scans. This mode is useful when working with non-time-critical data or large amounts of data. You enter the number of elements to operate on for each scan, which keeps scan time shorter.

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.

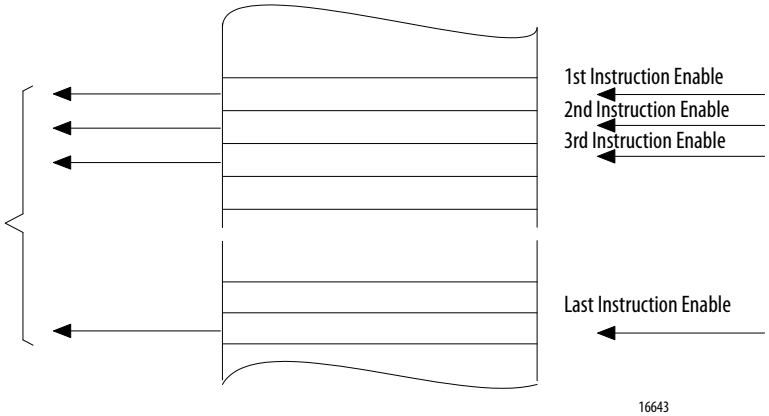


If the rung-condition-in is true at completion, the .EN and .DN bit are set until the rung-condition-in goes false. When the rung-condition-in goes false, these bits are cleared and the .POS value is cleared.

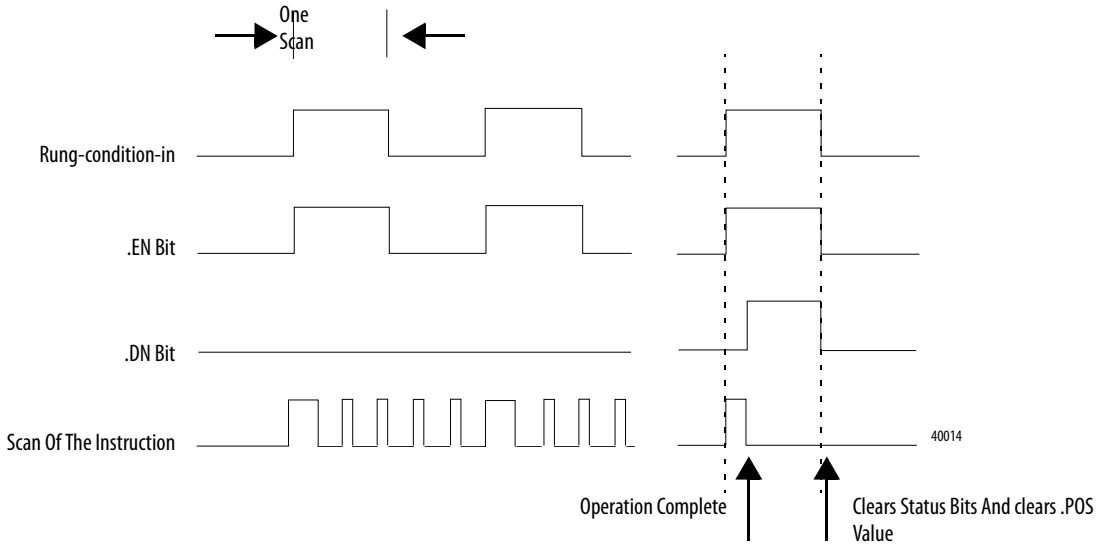
If the rung-condition-in is false at completion, the .EN bit is cleared immediately. One scan after the .EN bit is cleared, the .DN bit and the .POS value are cleared.

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 as follows:

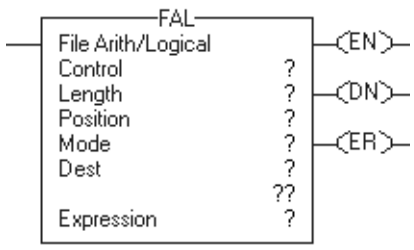
- Numerical mode with any number of elements per scan requires only one false-to-true transition of the rung-condition-in to start execution. The instruction continues to execute the specified number of elements each scan until completion regardless of the state of the rung-condition-in.
- Incremental 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 by 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 Structured Text Programming 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 352 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:

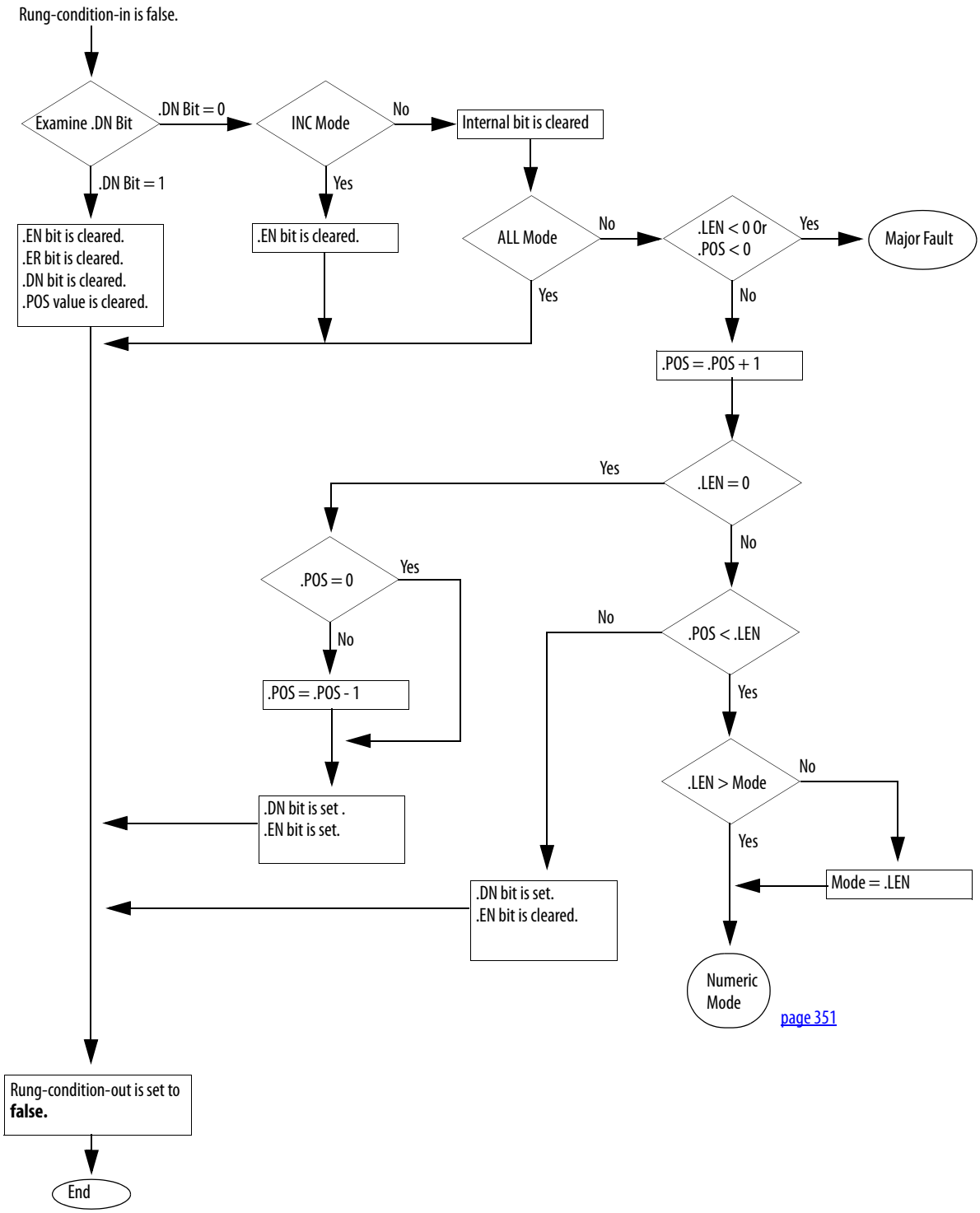
Arithmetic status flags are affected.

Fault Conditions:

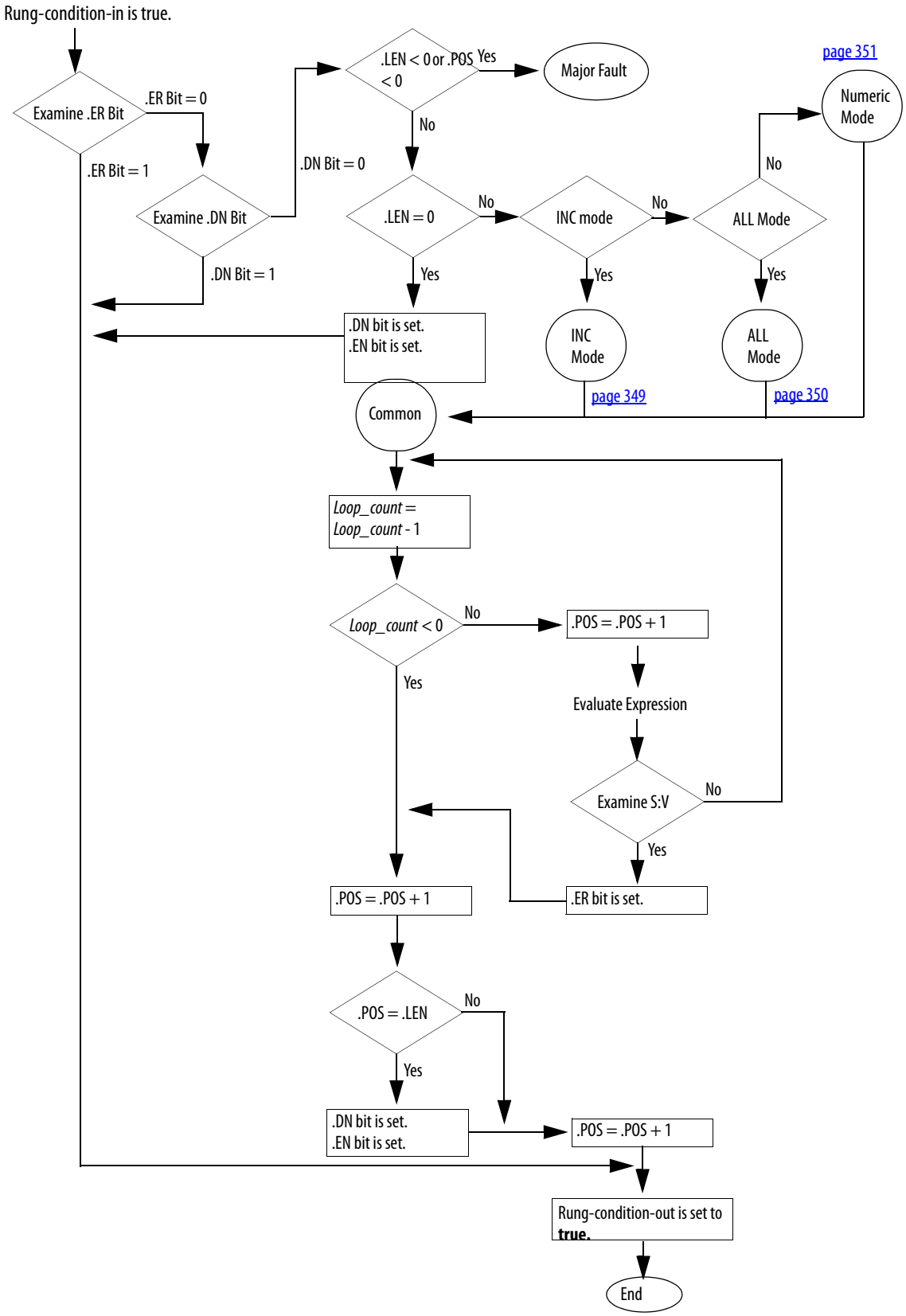
A major fault will occur if	Fault type	Fault code
Subscript is out of range	4	20
.POS < 0 or .LEN < 0	4	21

Execution:

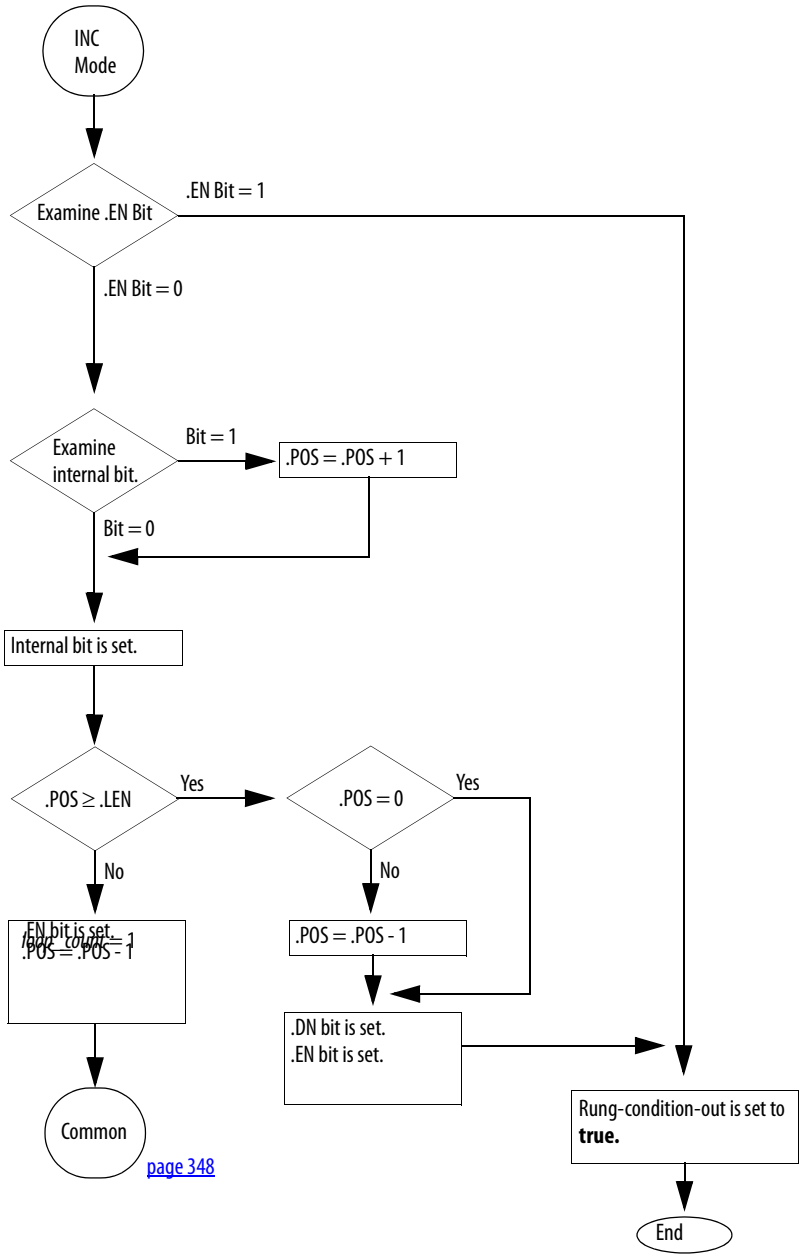
Condition	Relay Ladder Action
Prescan	The rung-condition-out is set to false.



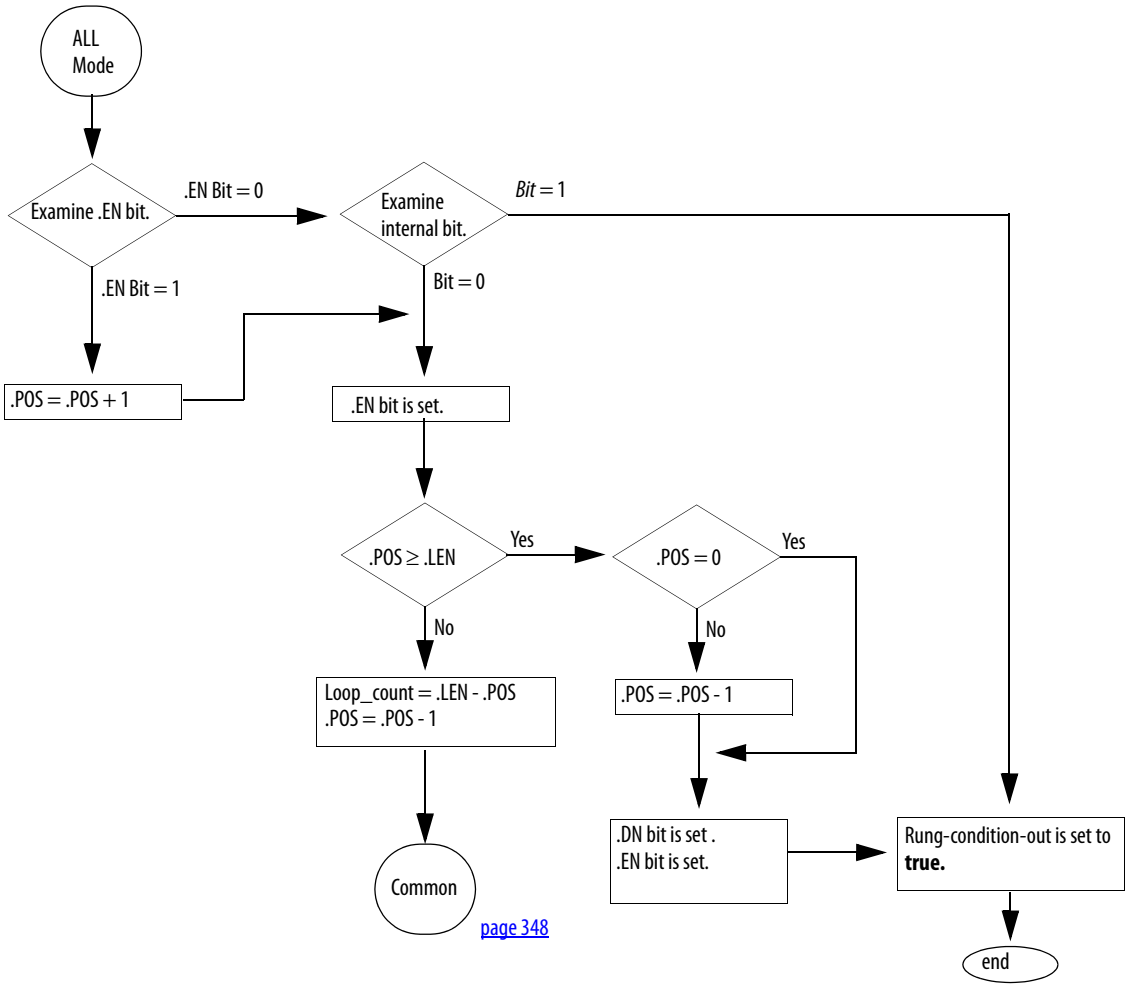
Condition	Relay Ladder Action



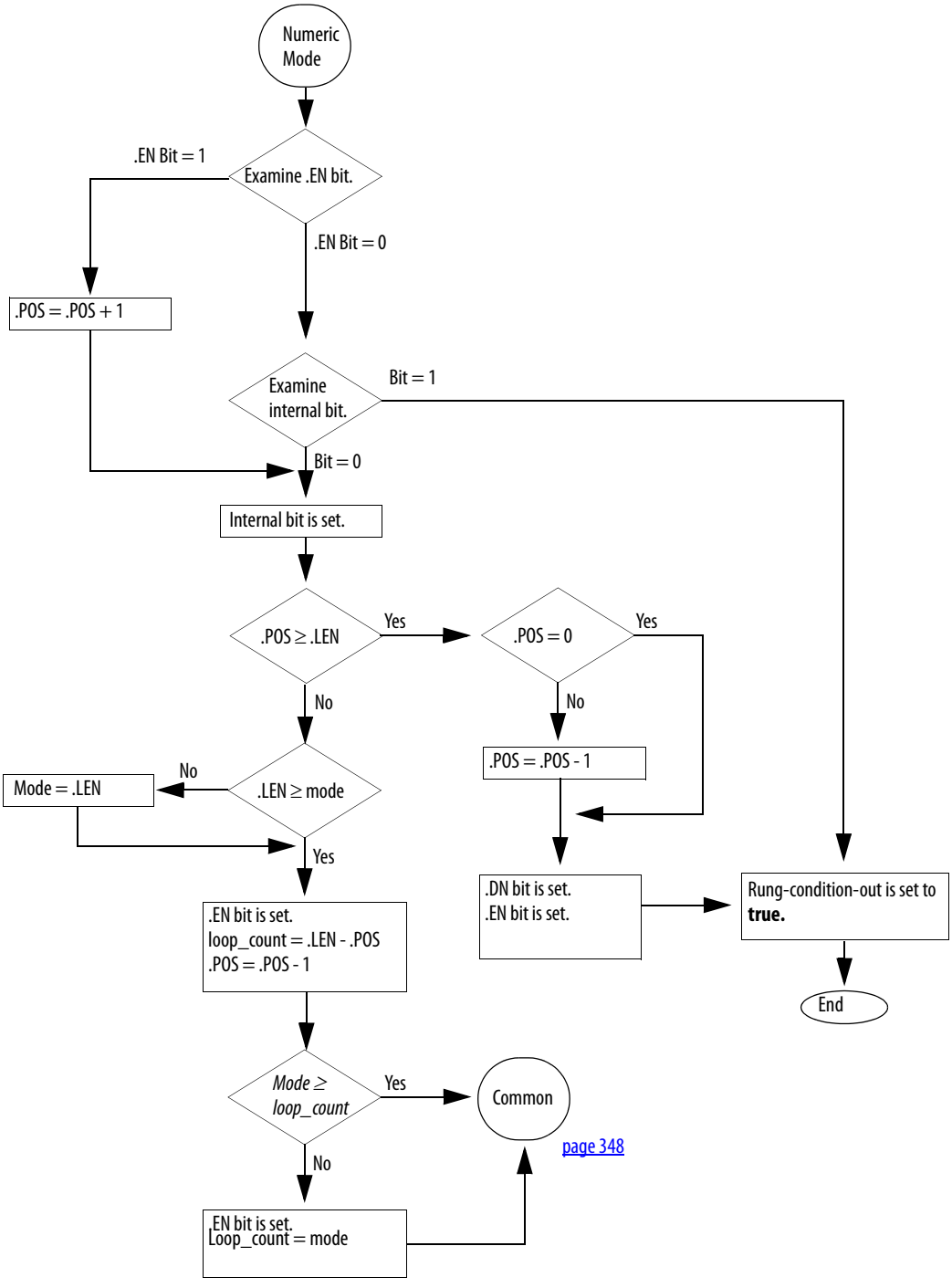
Condition	Relay Ladder Action



Condition	Relay Ladder Action

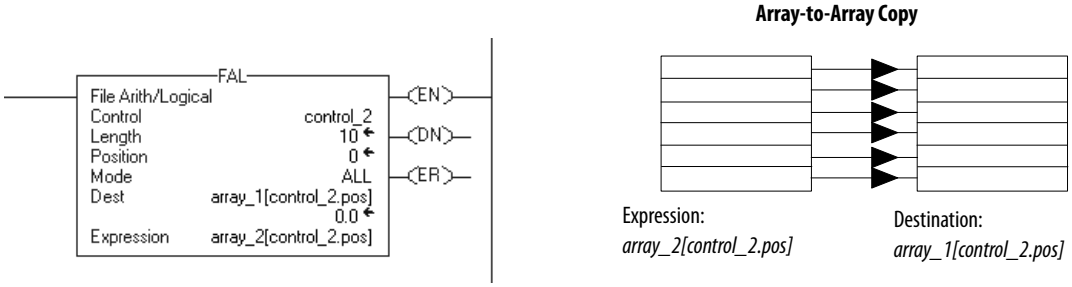


Condition	Relay Ladder Action

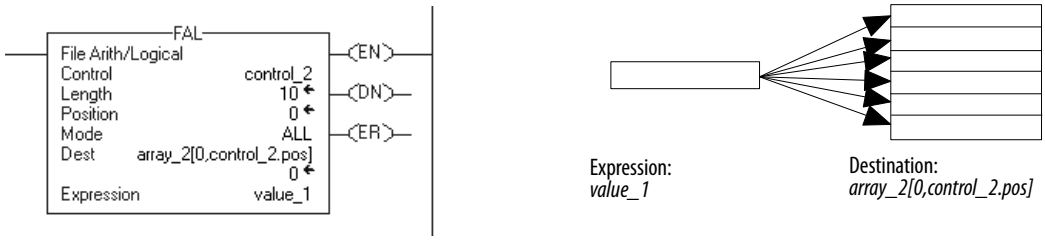


Postscan	The rung-condition-out is set to false.

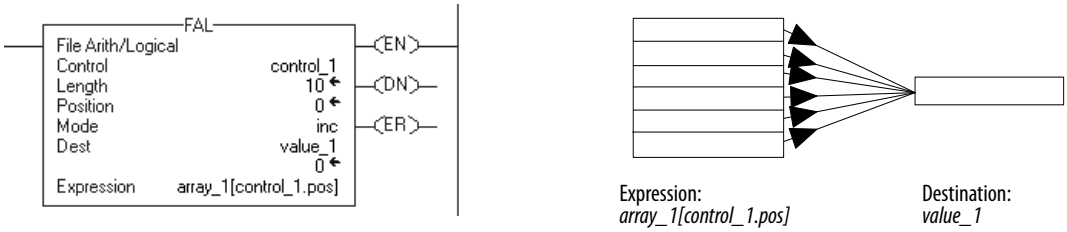
Example 1: When enabled, the FAL instruction copies each element of *array_2* into the same position within *array_1*.



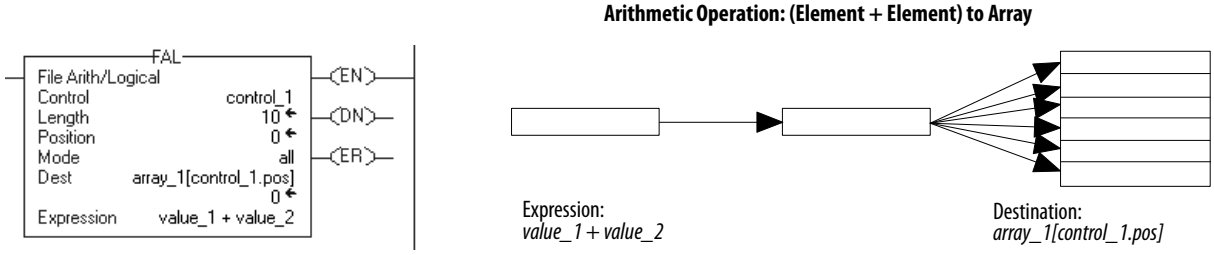
Example 2: When enabled, the FAL instruction copies *value_1* into the first 10 positions of the second dimension of *array_2*.



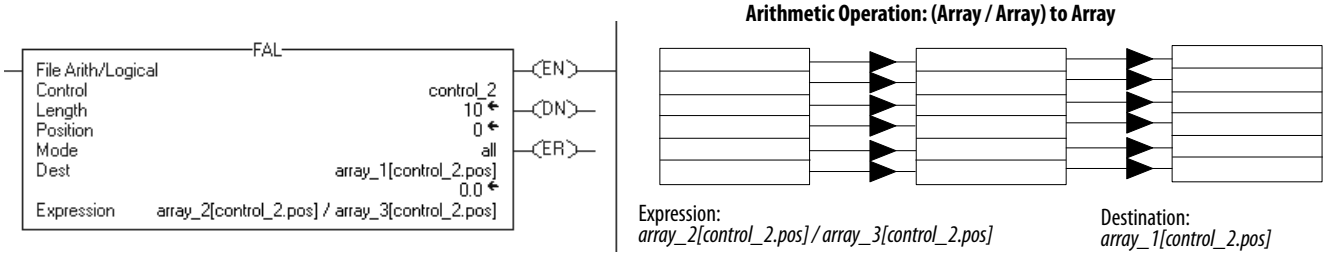
Example 3: Each time the FAL instruction is enabled, it copies the current value of *array_1* to *value_1*. The FAL instruction uses incremental mode, so only one array value is copied each time the instruction is enabled. The next time the instruction is enabled, the instruction overwrites *value_1* with the next value in *array_1*.



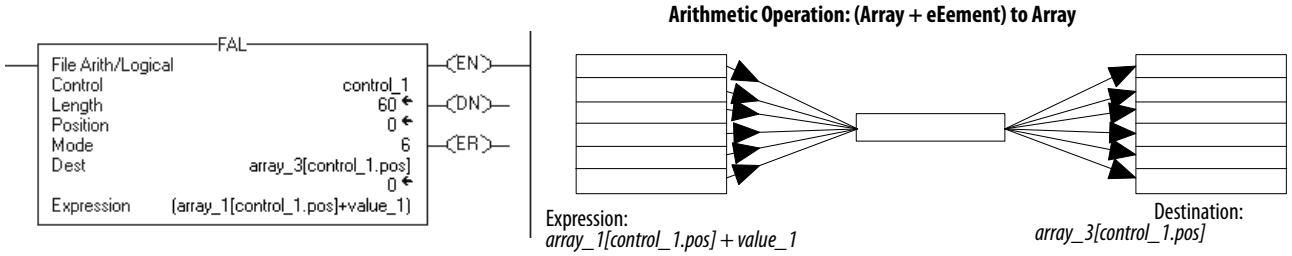
Example 4: When enabled, the FAL instruction adds *value_1* and *value_2* and stores the result in the current position of *array_1*.



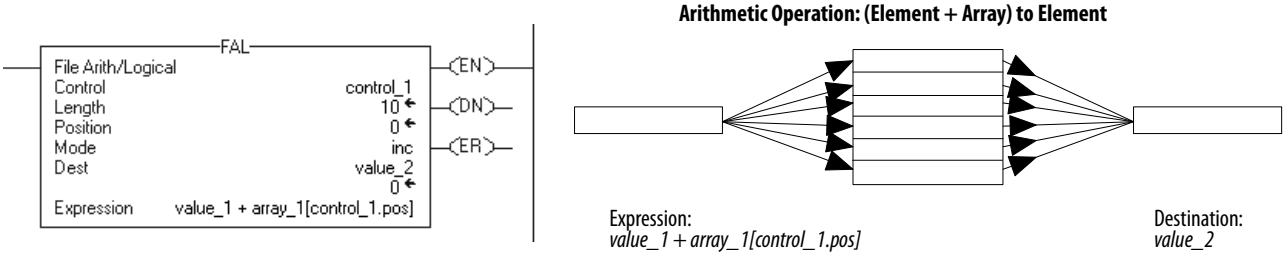
Example 5: When enabled, the FAL instruction divides the value in the current position of *array_2* with the value in the current position of *array_3* and stores the result in the current position of *array_1*.



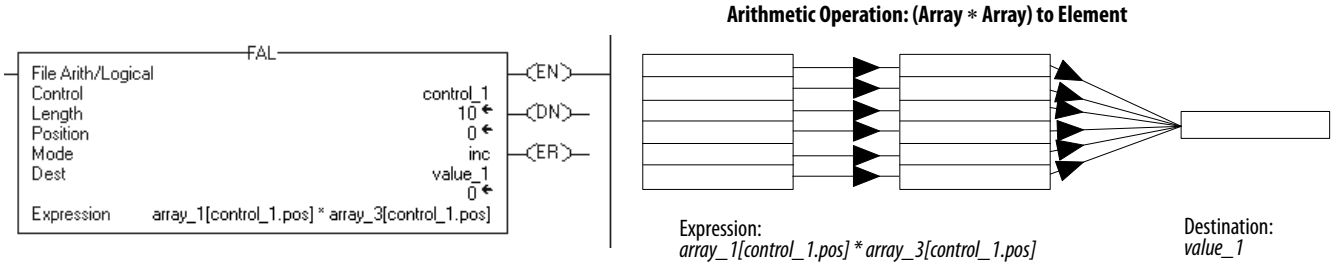
Example 6: When enabled, the FAL instruction adds the value at the current position in *array_1* to *value_1* and stores the result in the current position in *array_3*. The instruction must execute 10 times for the entire *array_1* and *array_3* to be manipulated.



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
LOG	Log base 10	REAL

Operator	Description	Optimal
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

Format 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	• *tag_b* + 5 • *tag_c* AND *tag_d* • (*tag_e* ** 2) MOD (*tag_f* / *tag_g*)

Determine 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.

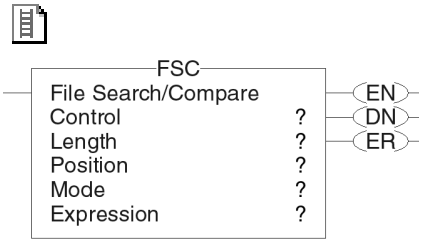
Order	Operation
1.	( )
2.	ABS, ACS, ASN, ATN, COS, DEG, FRD, LN, LOG, RAD, SIN, SQR, TAN, TOD, TRN
3.	**
4.	– (negate), NOT
5.	*, /, MOD
6.	– (subtract), +
7.	AND
8.	XOR
9.	OR

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:

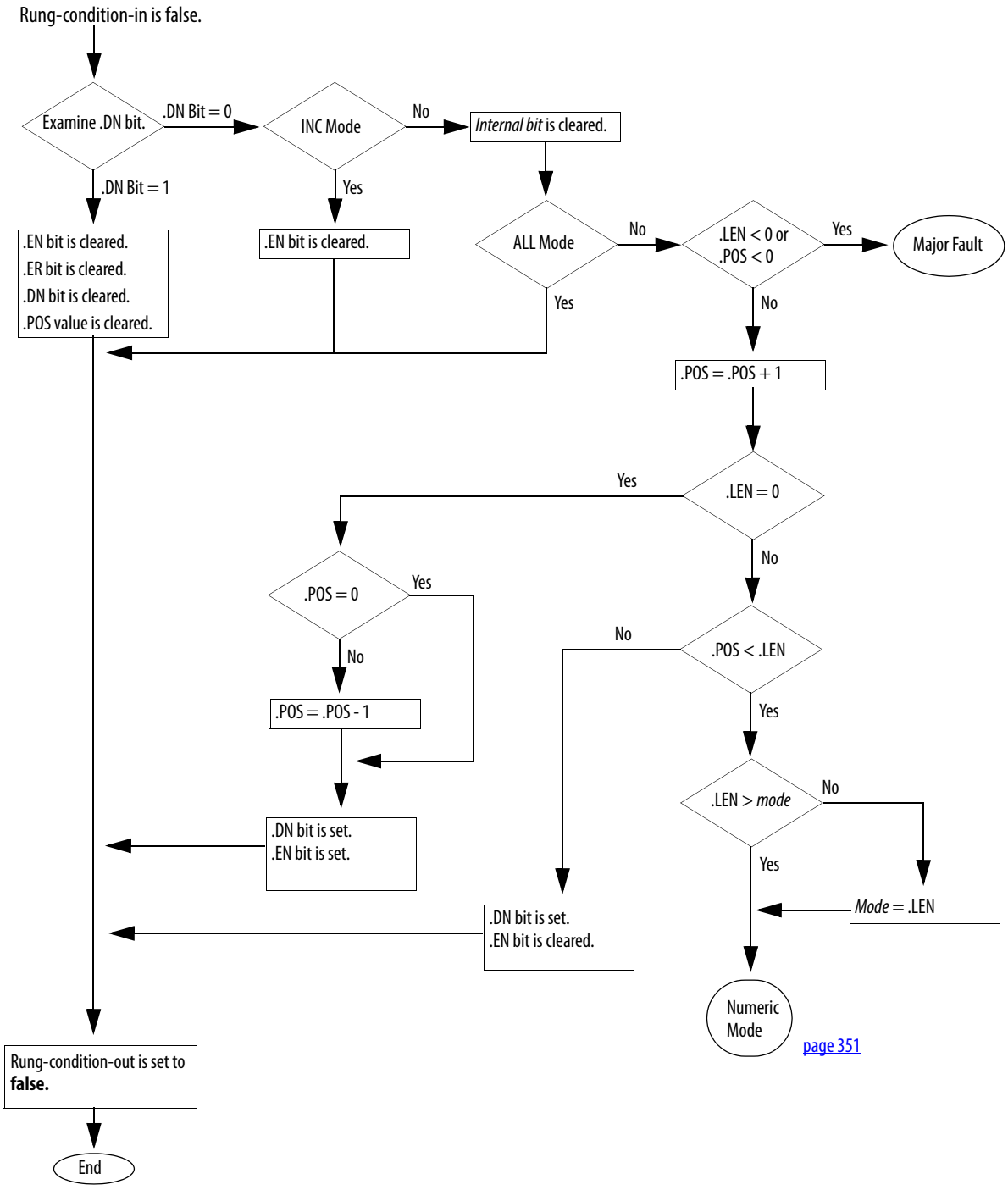
Arithmetic status flags are affected.

Fault Conditions:

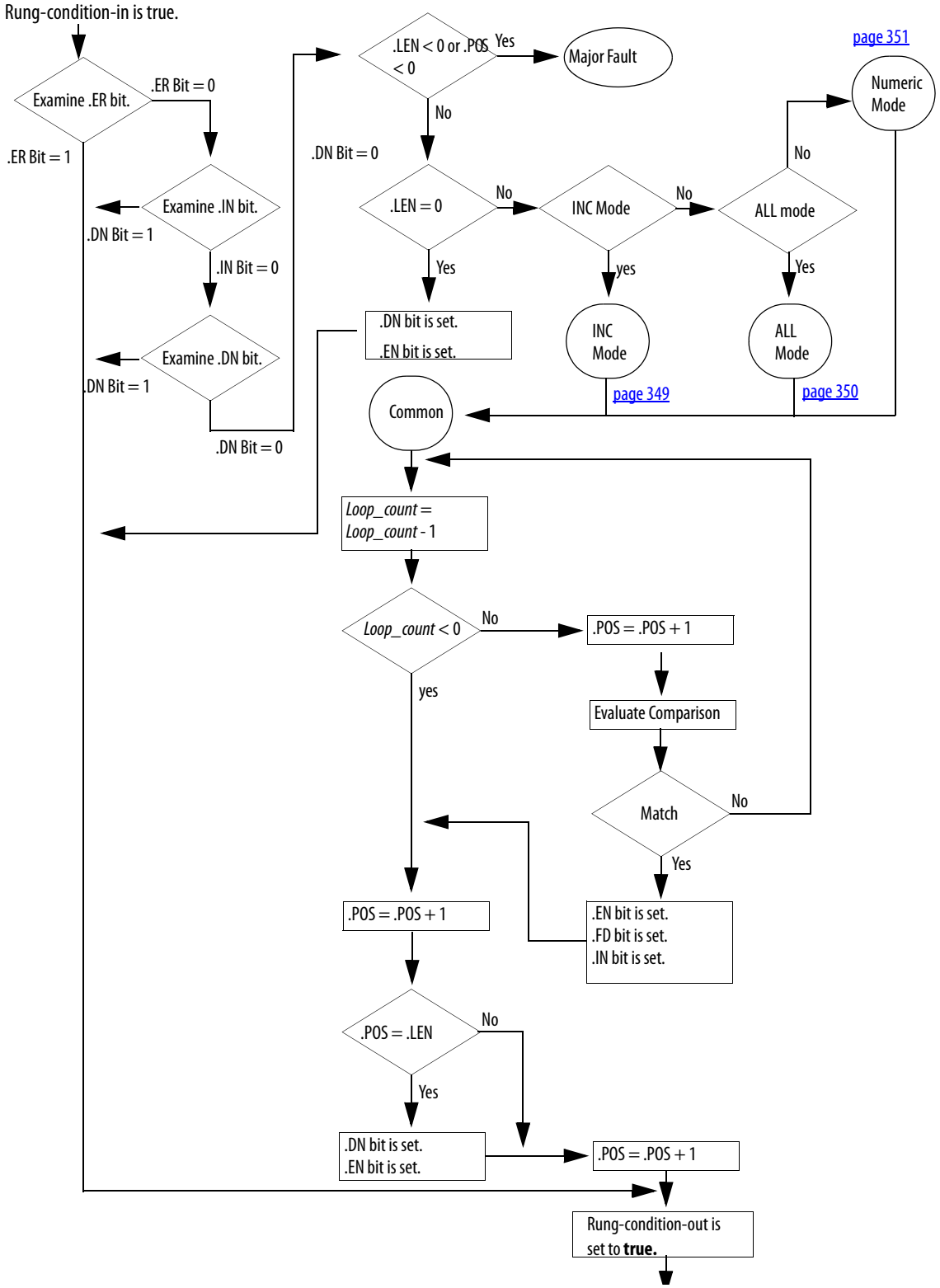
A Major Fault Will Occur If	Fault Type	Fault Code
.POS < 0 or .LEN < 0	4	21

Execution:

Condition	Relay Ladder Action
Prescan	The rung-condition-out is set to false.

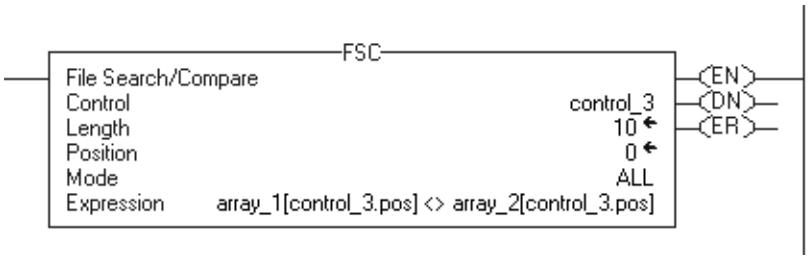


Condition	Relay Ladder Action



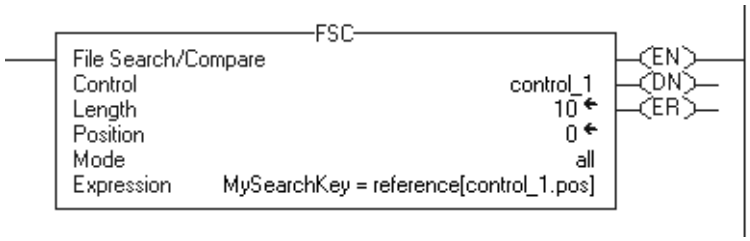
Postscan	The rung-condition-out is set to false.

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	
00000000000000001111111111111111	11111111111111110000000000000000	4	→ 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.
11111111111111111111111111111111	11111111111111111111111111111111	5	
11111111111111111111111111111111	11111111111111111111111111111111	6	
11111111111111111111111111111111	11111111111111111111111111111111	7	
11111111111111111111111111111111	11111111111111111111111111111111	8	
11111111111111111111111111111111	11111111111111111111111111111111	9	

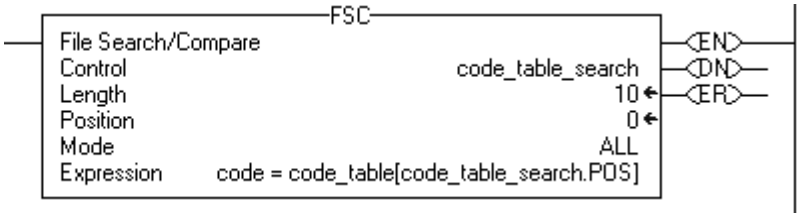
Example 2: Search for a match in an array. When enabled, the FSC instruction compares the *MySearchKey* to 10 elements in *array_1*.



MySearchKey	*Reference*	*Control_3.pos*
	00000000000000000000000000000000	0
	00000000000000000000000000000000	1
	00000000000000000000000000000000	2
	00000000000000000000000000000000	3
11111111111111110000000000000000	11111111111111110000000000000000	4
	11111111111111111111111111111111	5
	11111111111111111111111111111111	6
	11111111111111111111111111111111	7
	11111111111111111111111111111111	8
	11111111111111111111111111111111	9

The FSC instruction finds that this array element equals *MySearchKey*. The instruction sets the .FD and .IN bits. The .POS (4) value indicates the position of the equal element. To continue comparing the rest of the array, clear the .IN bit.

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*.



Code	*Code_table*	*Code_table_search.POS*
	AFG	0
	BEH	1
	HUO	2
	SAK	3
SAM	SAM	4
	FQG	5
	CLE	6
	CAK	7
	DET	8
	BWG	9

The FSC instruction finds that this array element equals *code*. The instruction sets the .FD and .IN bits. The .POS (4) value indicates the position of the equal element. To continue comparing the rest of the array, clear the .IN bit.

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
DEG	Radians to degrees	DINT, REAL
FRD	BCD to integer	DINT

Operator	Description	Optimal
LN	Natural log	REAL
LOG	Log base 10	REAL
MOD	Modulo-divide	DINT, REAL
NOT	Bbitwise 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

Format 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	• *tag_b* + 5 • *tag_c* AND *tag_d* • (*tag_e* ** 2) MOD (*tag_f* / *tag_g*)

Determine 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

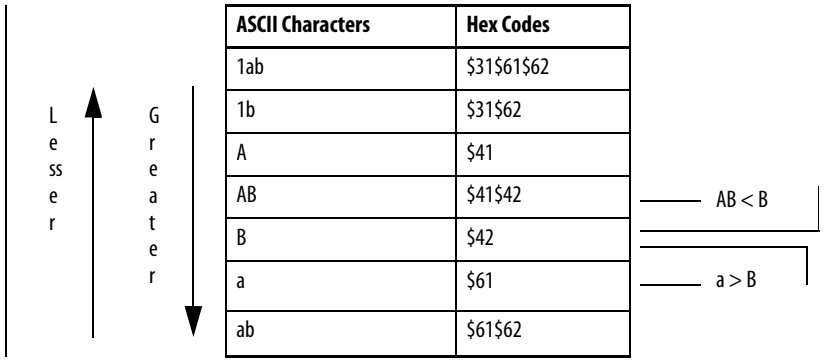
Use Strings in an Expression

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.

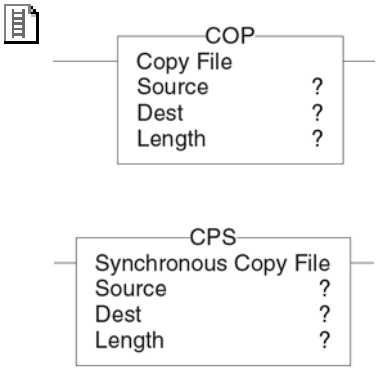


ASCII Characters	Hex Codes
1ab	$31$61$62
1b	$31$62
A	$41
AB	$41$42
B	$42
a	$61
ab	$61$62

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.

Structured Text

COP(Source,Dest,Length);

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
• Produced tag • Consumed tag • I/O data • Data that another task can overwrite	Prevent the data from changing during the copy operation	CPS	• 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 *ControlLogix System User Manual*, 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.

IMPORTANT

You **must** test and confirm that the instruction doesn't change data that you don't want it to change.

The COP and CPS instructions operate on contiguous memory. They do a straight byte-to-byte memory copy. In some cases, they write past the array into other members of the tag. This happens if the length is too big and the tag is a user-defined data type.

If the tag is	Then
User-defined data type	If the Length is too big, the instruction writes past the end of the array into other members of the tag. It stops at the end of the tag. No major fault is generated.
NOT user-defined data type	If the Length is too big, the instruction stops at the end of the array. No major fault is generated.

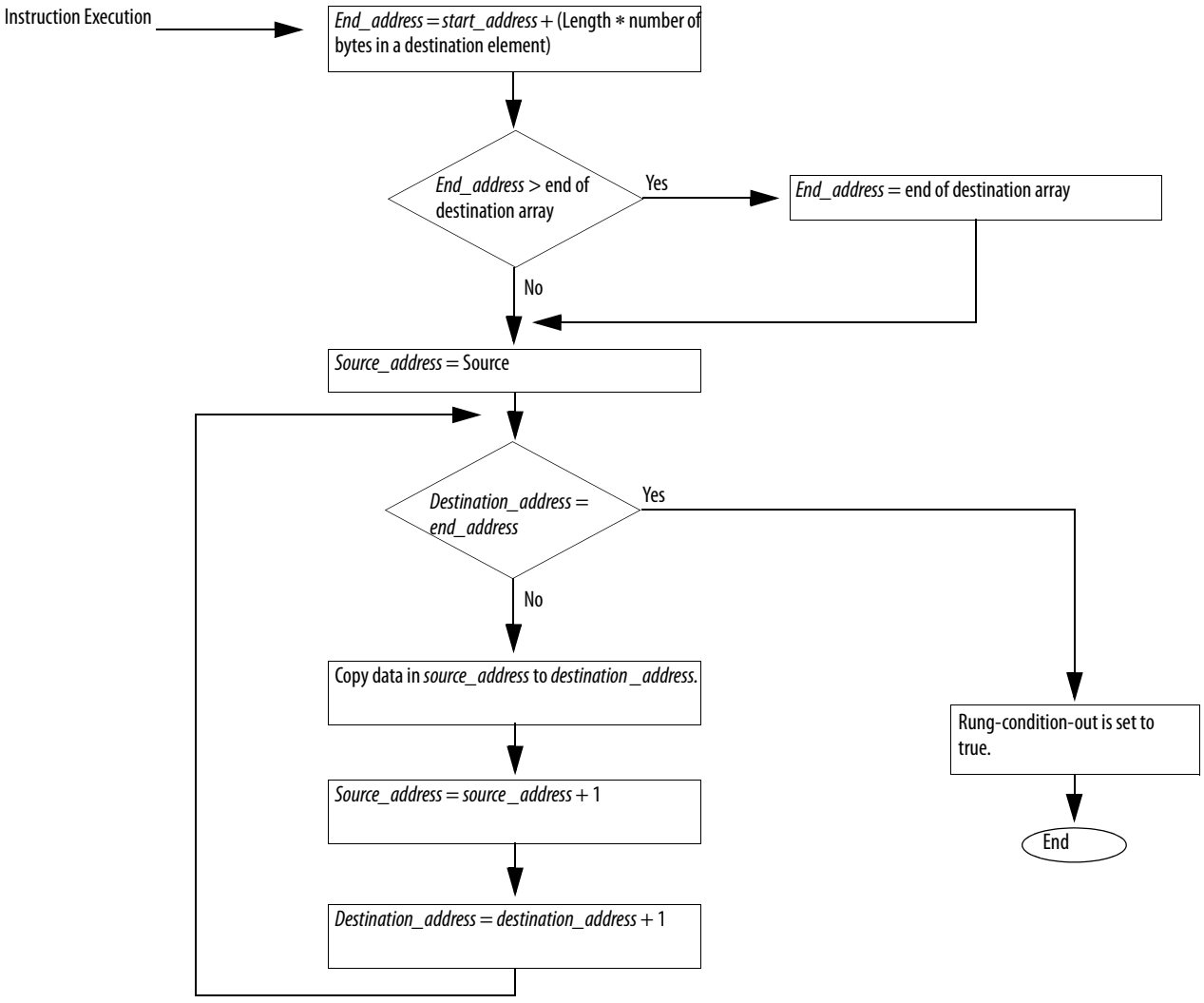
The Length is too big if it is more than the total number of elements in the Destination array.

Arithmetic Status Flags: Not affected

Fault Conditions: None

Execution:

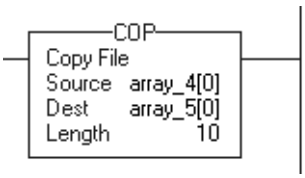
Condition	Relay Ladder Action	Structured Text Action
Prescan	The rung-condition-out is set to false.	No action taken.
Rung-condition-in is false	The rung-condition-out is set to false.	N/A
Rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	N/A
EnableIn is set	N/A	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

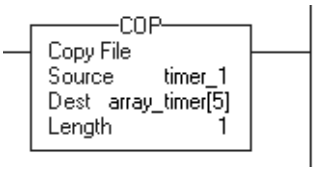


Structured Text

```
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



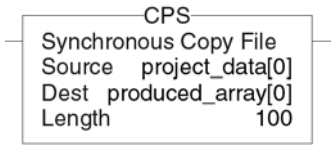
Structured Text

```
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 (that is, 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



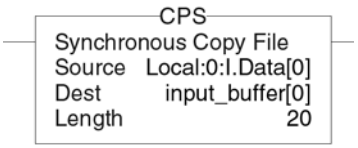
Structured Text

```
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*.

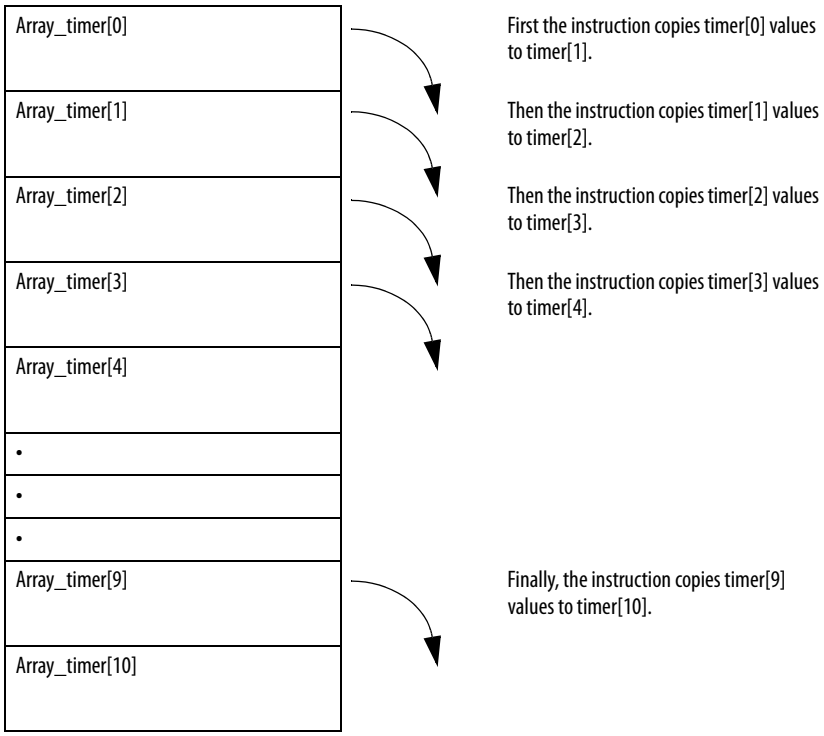
Relay Ladder



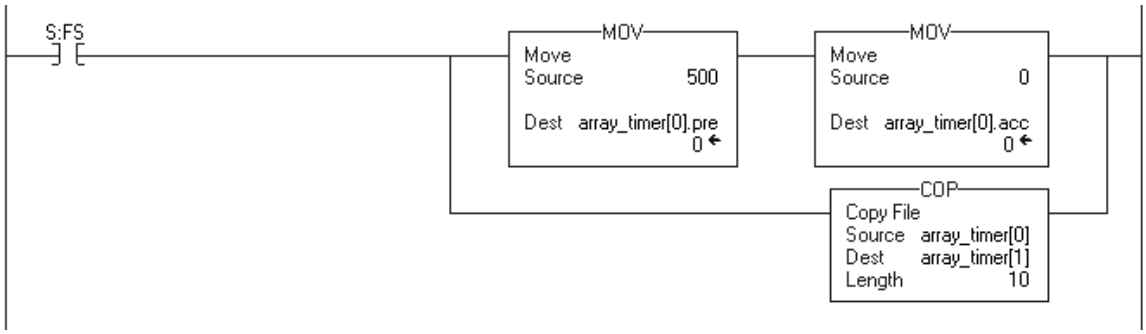
Structured Text

```
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

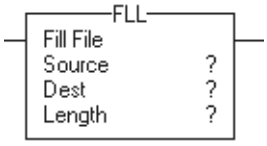
```
IF S:FS THEN

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 by 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 Structured Text Programming 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)

IMPORTANT You **must** test and confirm that the instruction doesn't change data that you don't want it to change.

The FLL instruction operates on contiguous data memory. In some cases, the instruction writes past the array into other members of the tag. This happens if the length is too big and the tag is a user-defined data type.

If the tag is	Then
User-defined data type	If the Length is too big, the instruction writes past the end of the array into other members of the tag. It stops at the end of the tag. No major fault is generated.
NOT user-defined data type	If the Length is too big, the instruction stops at the end of the array. No major fault is generated.

The Length is too big if it is more than the total number of elements in the Destination array.

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 362). If you mix data types for the Source and Destination, the Destination elements are filled with converted Source values.

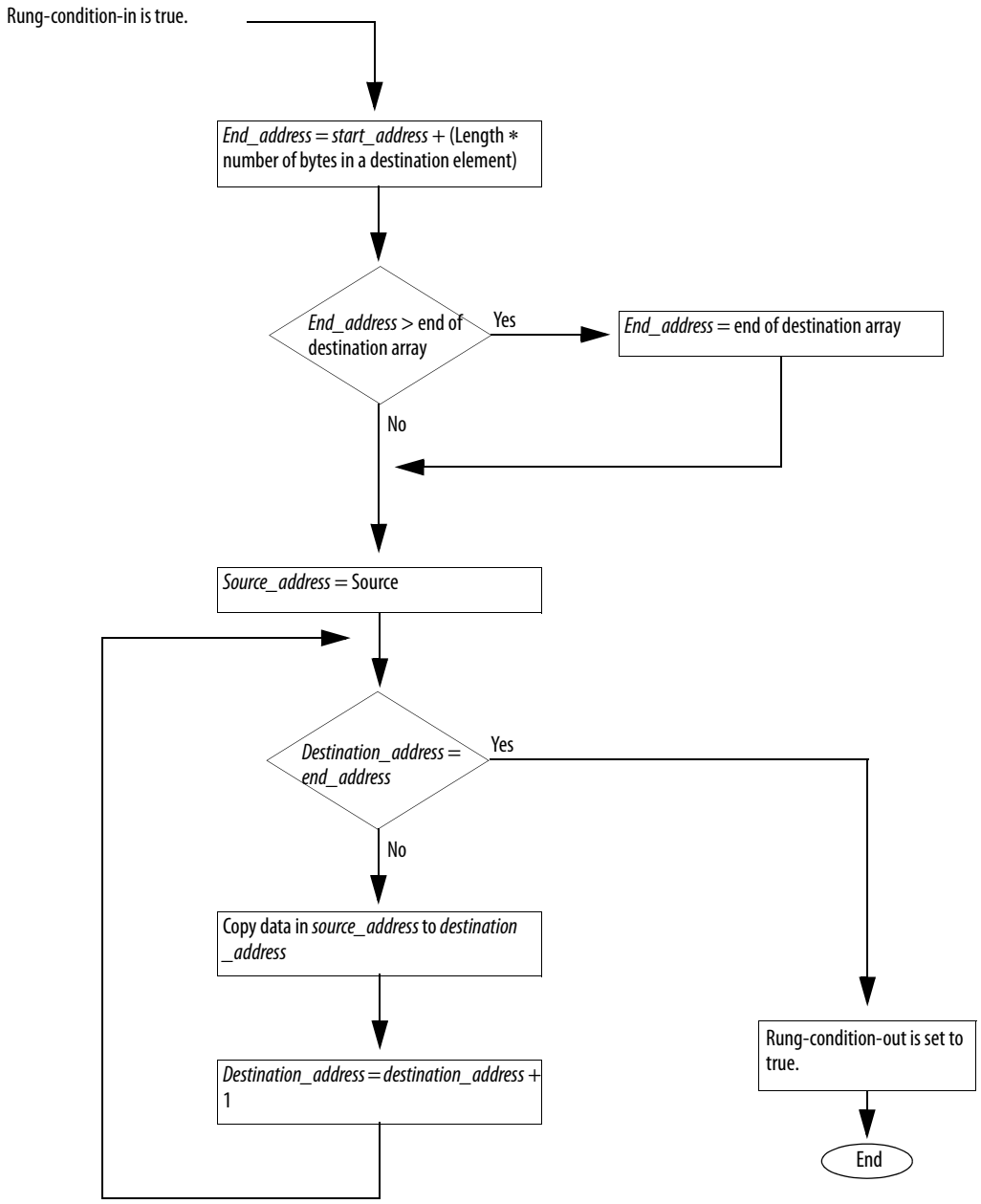
If The Source Is	And The Destination Is	The Source Is Converted To
SINT, INT, DINT, or REAL	SINT	SINT
SINT, INT, DINT, or REAL	INT	INT
SINT, INT, DINT, or REAL	DINT	DINT
SINT, INT, DINT, or REAL	REAL	REAL
SINT	Structure	SINT (not converted)
INT	Structure	INT (not converted)
DINT	Structure	DINT (not converted)
REAL	Structure	REAL (not converted)

Arithmetic Status Flags: Not affected

Fault Conditions: None

Execution:

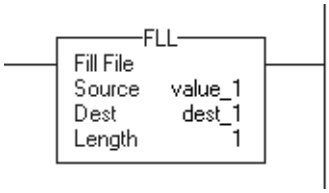
Condition	Relay Ladder Action
Prescan	The rung-condition-out is set to false.
Rung-condition-in is false	The rung-condition-out is set to false.



Postscan	The rung-condition-out is set to false.

Example: The FLL instruction copies the value in *value_1* into *dest_1*

Relay Ladder



Source (*value_1*) Data Type	Source (*value_1*) Value	Destination (*dest_1*) Data Type	Destination (*dest_1*) 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

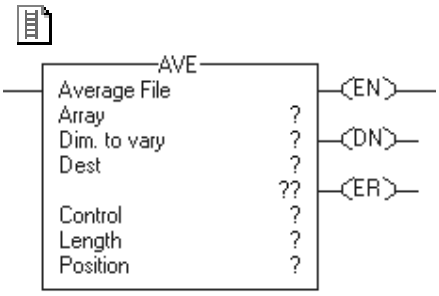
Structured Text

```
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 by 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 Structured Text Programming 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.

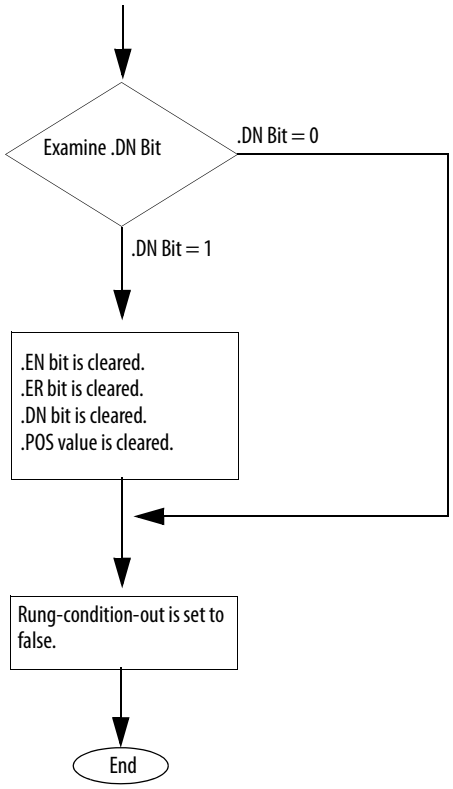
Fault Conditions:

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.



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 347.
Postscan	The rung-condition-out is set to false.

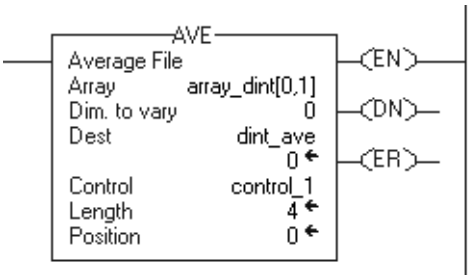
Example 1: Average *array_dint*, which is DINT[4,5].

Subscripts	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

$$AVE = \frac{19 + 14 + 9 + 4}{4} = \frac{46}{4} = 11.5$$

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;
```

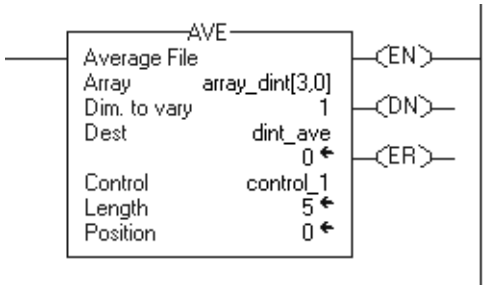
Example 2: Average *array_dint*, which is DINT[4,5].

Subscripts	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

$$AVE = \frac{5 + 4 + 3 + 2 + 1}{5} = \frac{15}{5} = 3$$

dint_ave = 3

Relay Ladder



Structured Text

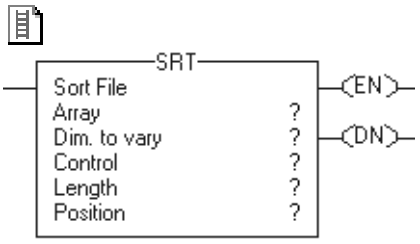
```
SIZE(array_dint,1,length);

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

Structured Text

SRT(Array,Dimtovary, Control);

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	You must test and confirm that the instruction doesn't change data that you don't want it to change. The SRT instruction operates on contiguous memory. In some cases, the instruction changes data in other members of the tag. This happens if the length is too big and the tag is a user-defined data type.

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 Structured Text Programming.

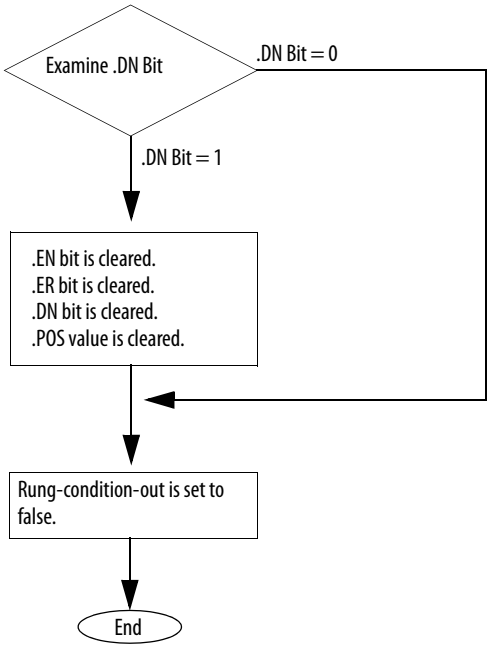
Arithmetic Status Flags:

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.		N/A
Rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	N/A
EnableIn is set	N/A	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].

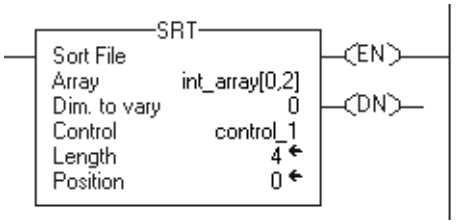
Before

Subscripts	Dimension 1: 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

After

Subscripts	Dimension 1: 0	1	2	3	4
Dimension 0: 0	20	19	3	17	16
1	15	14	8	12	11
2	10	9	13	7	6
3	5	4	18	2	1

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].

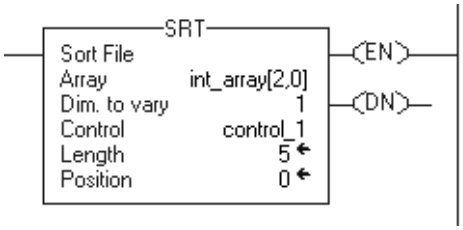
Before

Subscripts	Dimension 1: 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

After

Subscripts	Dimension 1: 0	1	2	3	4
Dimension 0: 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



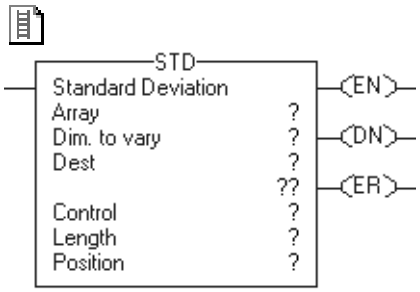
Structured Text

```
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 by 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 Structured Text Programming for information on the syntax of constructs within structured text.

Description: The standard deviation is calculated according to this formula:

$$\text{Standard Deviation} = \sqrt{\frac{\left(\sum_{i=1}^{N} [\langle X_{(start+i)} - AVE \rangle^2]\right)}{(N-1)}}$$

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
- AVE = $\frac{\left(\sum_{i=1}^{N} x_{(start+i)}\right)}{N}$

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:

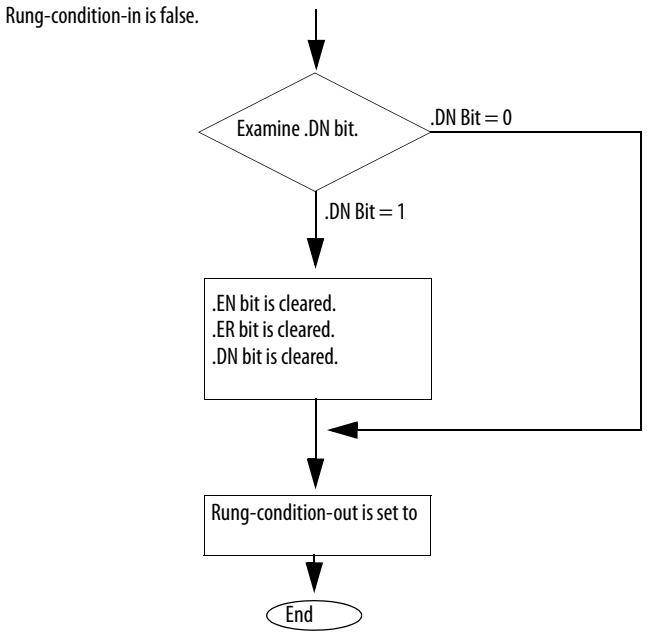
Arithmetic status flags are affected.

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.



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 347.
Postscan	The rung-condition-out is set to false.

Example 1: Calculate the standard deviation of *dint_array*, which is DINT[4,5].

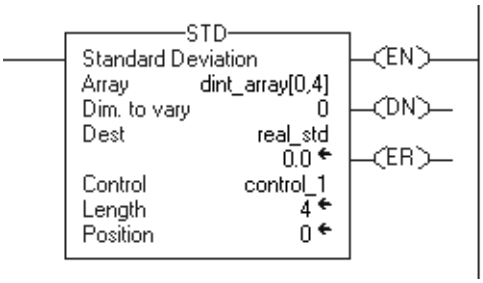
Subscripts	Dimension 1: 0	1	2	3	4
Dimension 0: 0	20	19	18	17	16
1	15	14	13	12	11
2	10	9	8	7	6
3	5	4	3	2	1

$$AVE = \frac{16 + 11 + 6 + 1}{4} = \frac{34}{4} = 8.5$$

$$STD = \sqrt{\frac{\langle 16 - 8.5\rangle^2 + \langle 11 - 8.5\rangle^2 + \langle 6 - 8.5\rangle^2 + \langle 1 - 8.5\rangle^2}{\langle 4 - 1\rangle}} = 6.454972$$

real_std = 6.454972

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 d*int_array*, which is DINT[4,5].

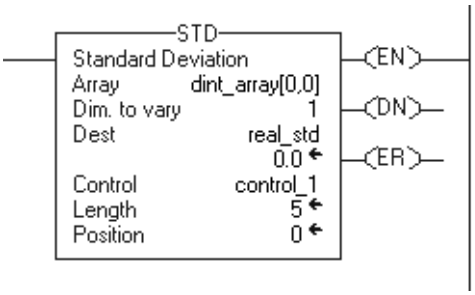
Subscripts	Dimension 1: 0	1	2	3	4
Dimension 0: 0	20	19	18	17	16
1	15	14	13	12	11
2	10	9	8	7	6
3	5	4	3	2	1

$$AVE = \frac{20 + 19 + 18 + 17 + 16}{5} = \frac{90}{5} = 18$$

$$STD = \sqrt{\frac{\langle 20 - 18\rangle^2 + \langle 19 - 18\rangle^2 + \langle 18 - 18\rangle^2 + \langle 17 - 18\rangle^2 + \langle 16 - 18\rangle^2}{\langle 5 - 1\rangle}} = 1.581139$$

real_std = 1.581139

Relay Ladder



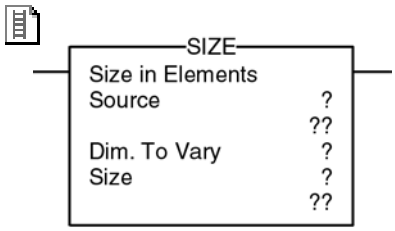
Structured Text

```
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:



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: **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

Structured Text

SIZE(Source,Dimtovary,Size);

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:

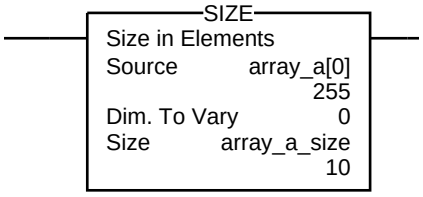
None.

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.	N/A
Rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	N/A
EnableIn is set	N/A	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

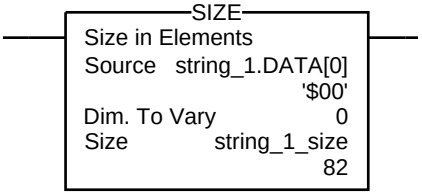


Structured Text

```
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

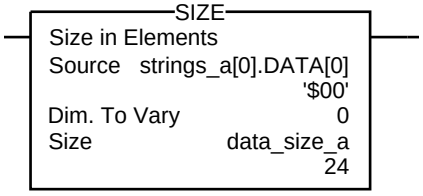


Structured Text

```
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)

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	Page
Load bits into, shift bits through, and unload bits from a bit array one bit at a time	BSL	Relay ladder	398
	BSR	Relay ladder	402
Load and unload values in the same order	FFL	Relay ladder	406
	FFU	Relay ladder	412
Load and unload values in reverse order	LFL	Relay ladder	418
	LFU	Relay ladder	424

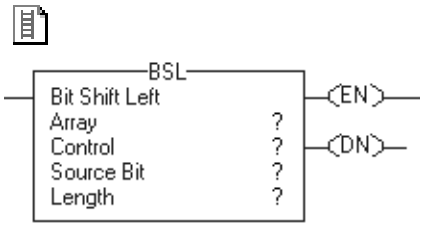
You can mix data types, but loss of accuracy and rounding errors might occur.

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 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 load
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.

IMPORTANT You must test and confirm that the instruction doesn't change data that you don't want it to change.

The BSL instruction operates on contiguous memory. If an Array is a member array, such as contained within a structure, it is possible that the instruction could shift beyond the Array's boundary into other members following it. You must take care in choosing a length whereby this does not happen.

Arithmetic Status Flags:

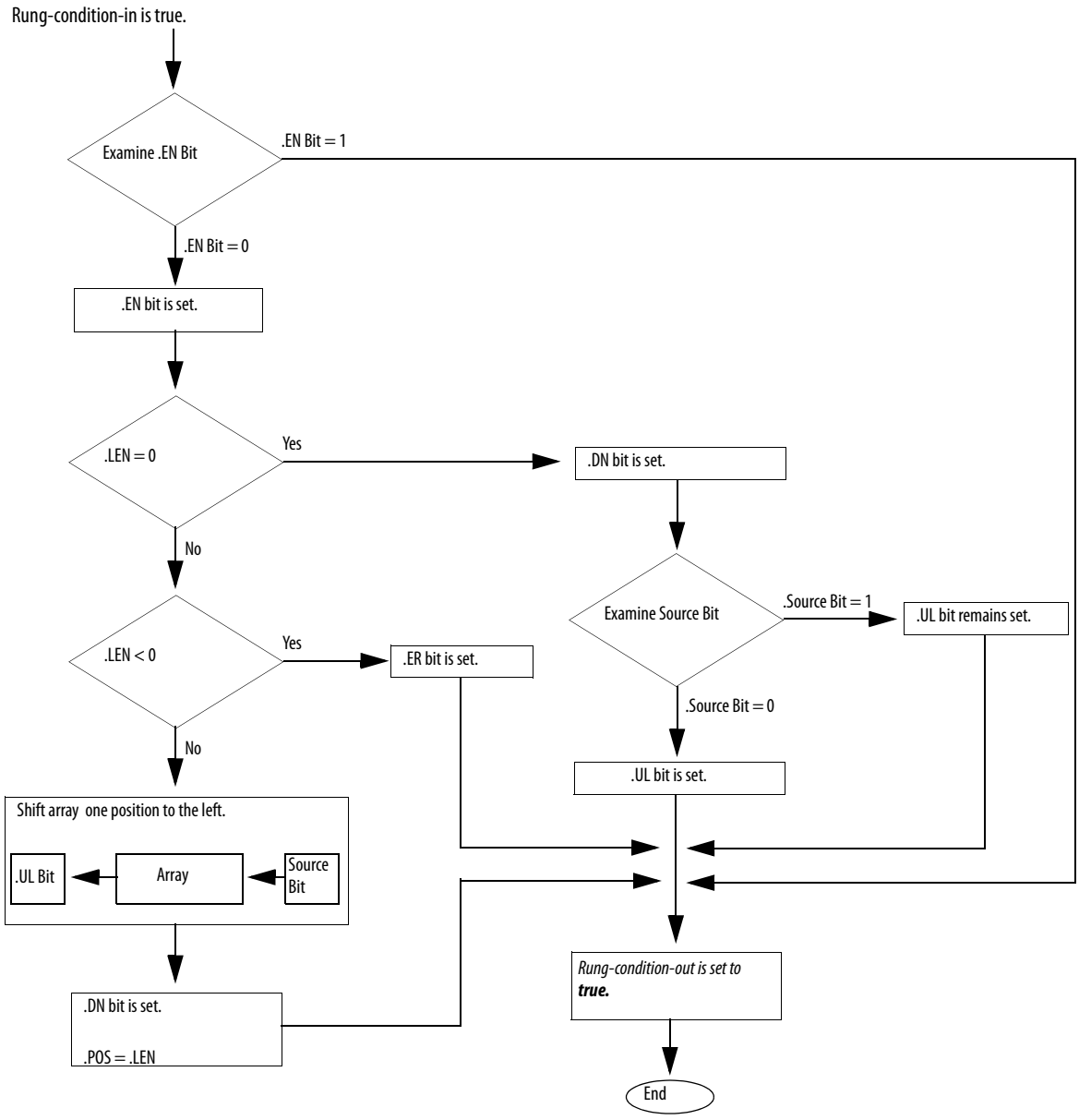
Not affected

Fault Conditions:

A major fault will occur if	Fault type	Fault code
Length exceeds the size of Array's storage area.	4	20

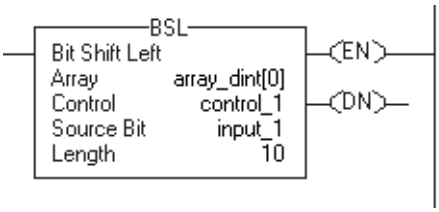
Execution:

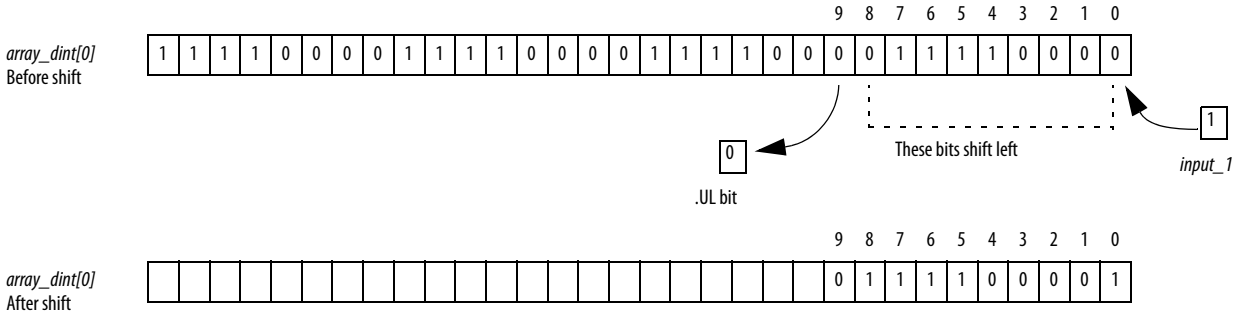
Condition	Relay Ladder Action
Prescan	The .EN bit is cleared. The .DN bit is cleared. The .ER bit is cleared. The .POS value is cleared. The rung-condition-out is set to false.
Rung-condition-in is false	The .EN bit is cleared. The .DN bit is cleared. The .ER bit is cleared. The .POS value is cleared. The rung-condition-out is set to false.

Condition	Relay Ladder Action
	
Postscan	The rung-condition-out is set to false.

Example 1:

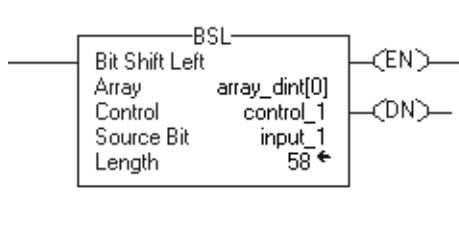
When enabled, the BSL instruction starts at bit 0 in *array_dint[0]*. The instruction unloads *array_dint[0].9* into the .UL bit, shifts the remaining bits, and loads *input_1* into *array_dint[0].0*. The values in the remaining bits (10...31) are invalid.

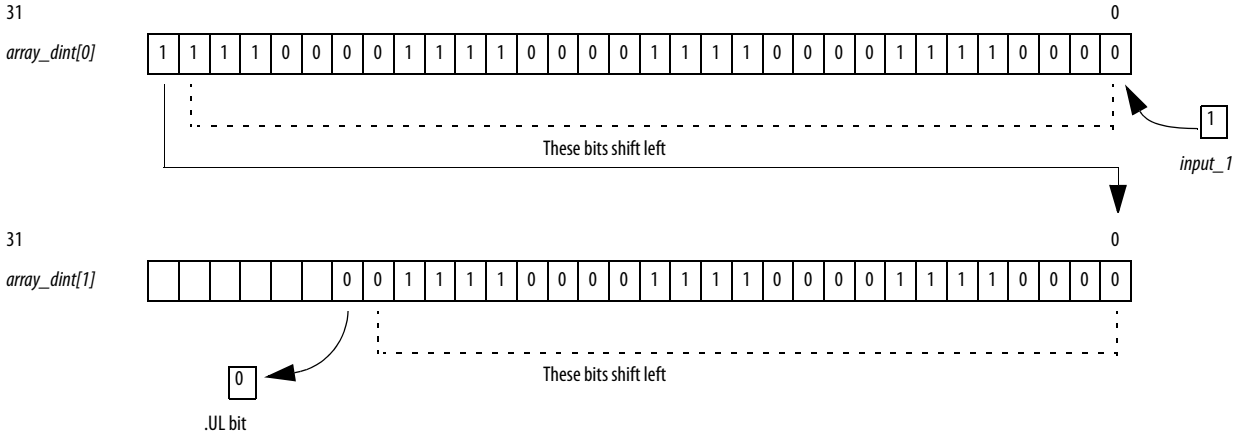




Example 2:

When enabled, the BSL instruction starts at bit 0 in *array_dint[0]*. The instruction unloads *array_dint[1].29* into the .UL bit, shifts the remaining bits, and loads *input_1* into *array_dint[0].0*. The values in the remaining bits (30...58 in *array_dint[1]*) are invalid. Note how *array_dint[0].31* shifts across words to *array_dint[1].0*.

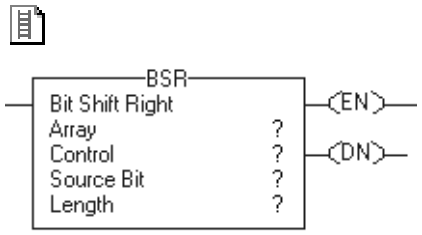




Bit Shift Right (BSR)

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 load
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.

IMPORTANT You must test and confirm that the instruction doesn't change data that you don't want it to change.

The BSR instruction operates on contiguous memory. If an Array is a member array, such as contained within a structure, it is possible that the instruction could shift beyond the Array's boundary into other members following it. You must take care in choosing a length whereby this does not happen.

Arithmetic Status Flags:

Not affected

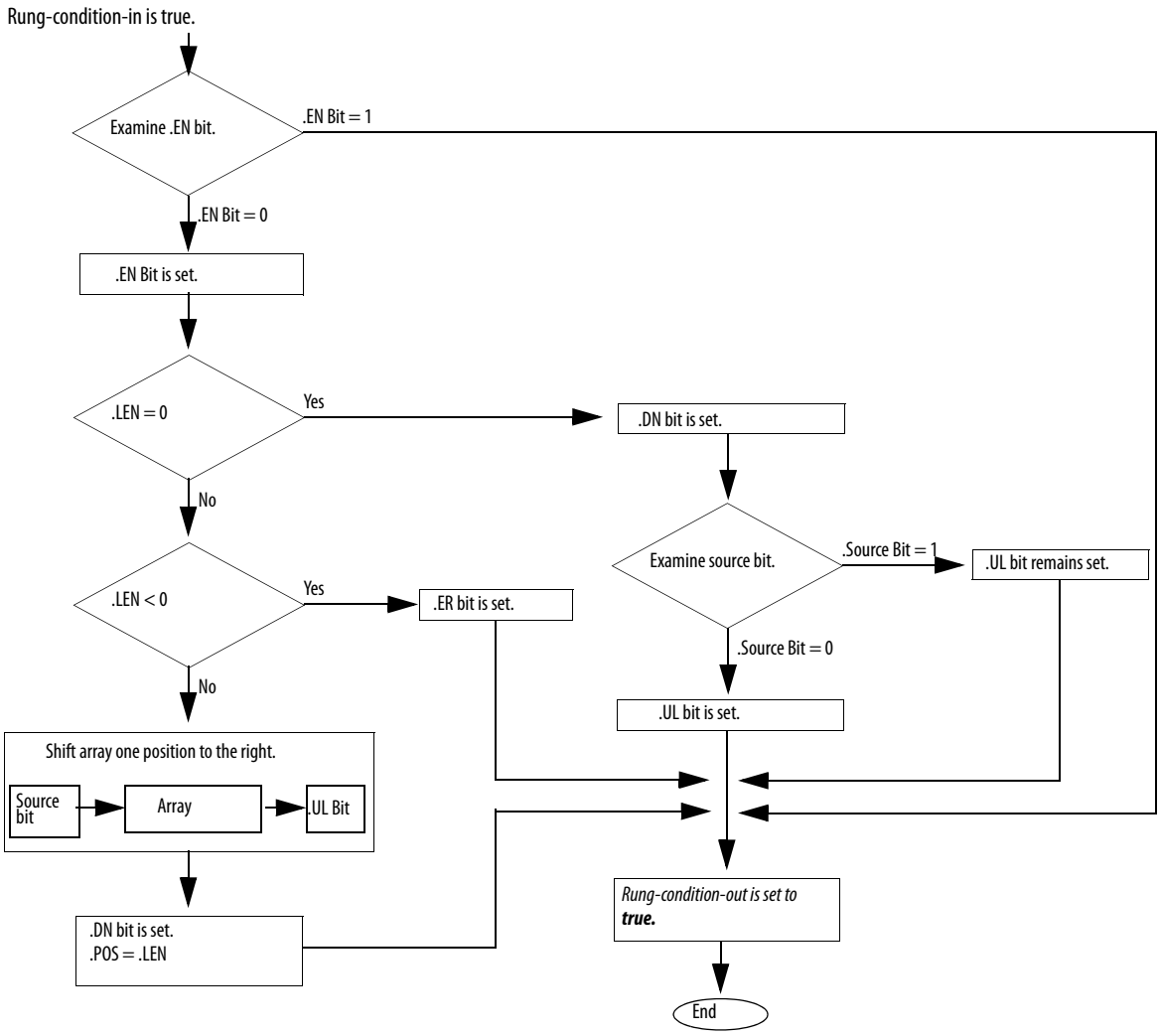
Fault Conditions:

A major fault will occur if	Fault type	Fault code
Length exceeds the size of Array's storage area.	4	20

Execution:

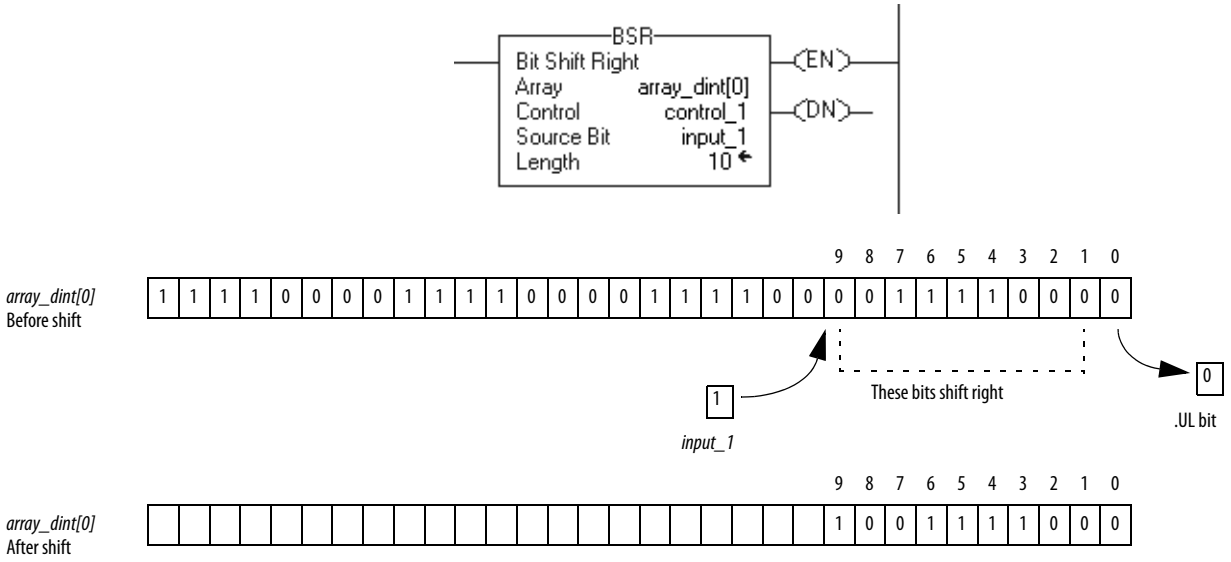
Condition	Relay Ladder Action
Prescan	The .EN bit is cleared. The .DN bit is cleared. The .ER bit is cleared. The .POS value is cleared. The rung-condition-out is set to false.
Rung-condition-in is false	The .EN bit is cleared. The .DN bit is cleared. The .ER bit is cleared. The .POS value is cleared. The rung-condition-out is set to false.

Condition	Relay Ladder Action

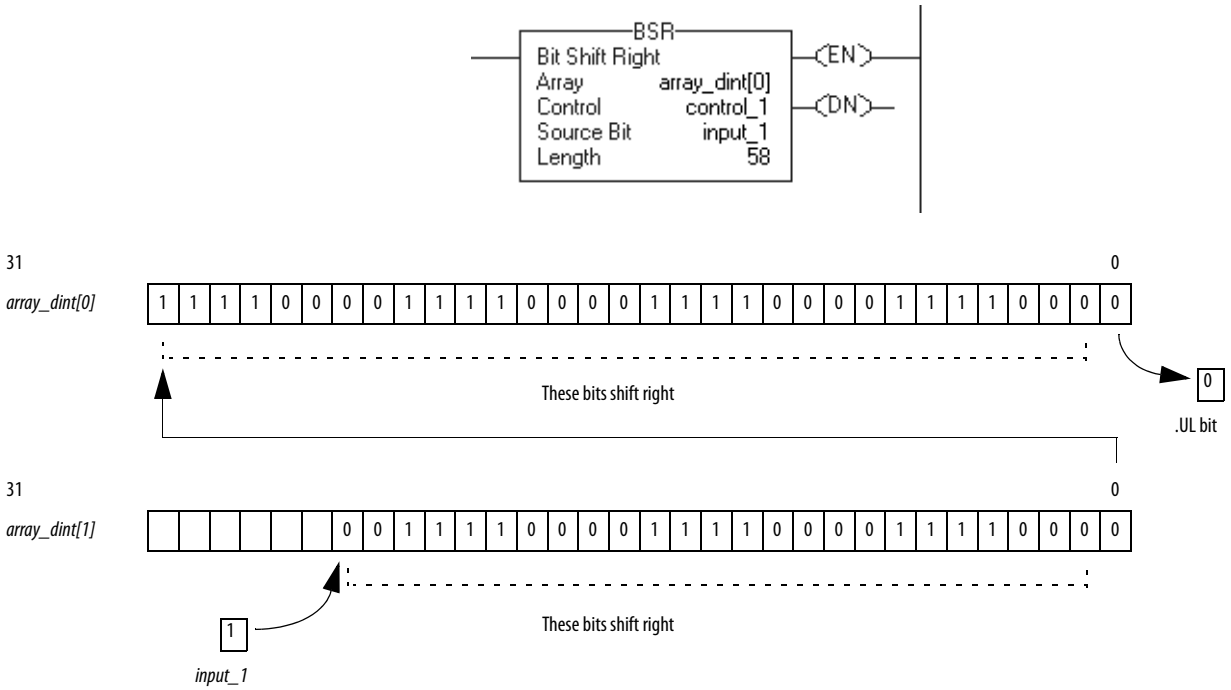


Postscan	The rung-condition-out is set to false.

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.



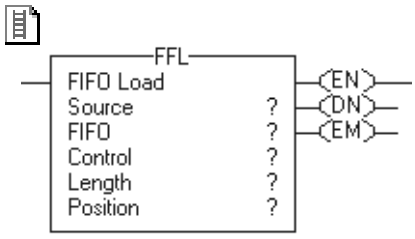
Example 2: When enabled, the BSR instruction starts at bit 25 in *array_dint[1].* The instruction unloads *array_dint[0].0* into the .UL bit, shifts the remaining bits right, and loads *input_1* into *array_dint[1].25.* The values in the remaining bits (31...26 in *dint_array[1]*) are invalid. Note how *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.

IMPORTANT You must test and confirm that the instruction doesn't change data that you don't want it to change.

The FFL instruction operates on contiguous memory. In some cases, the instruction loads data past the array into other members of the tag. This happens if the length is too big and the tag is a user-defined data type.

Arithmetic Status Flags:

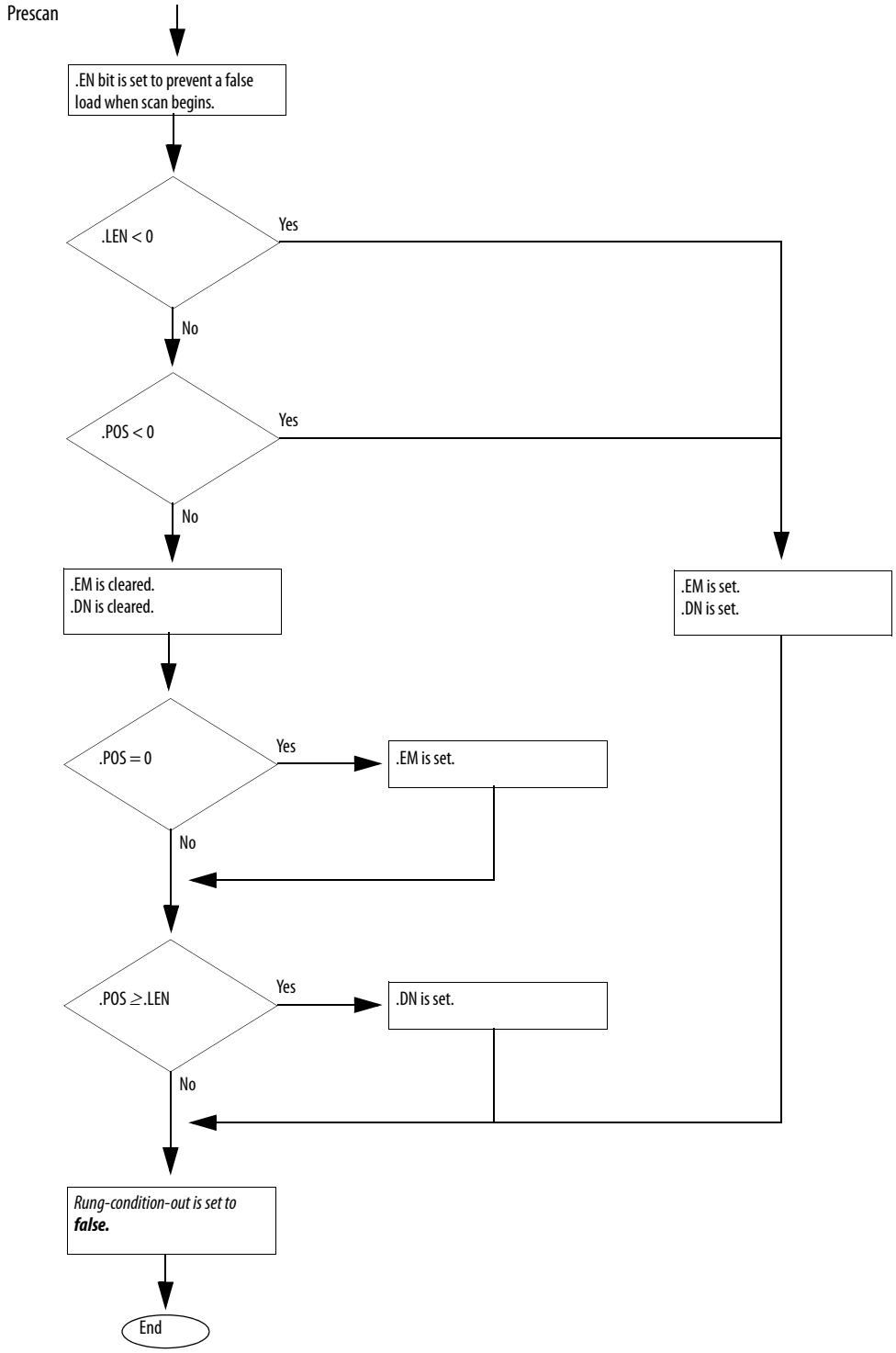
Not affected

Fault Conditions:

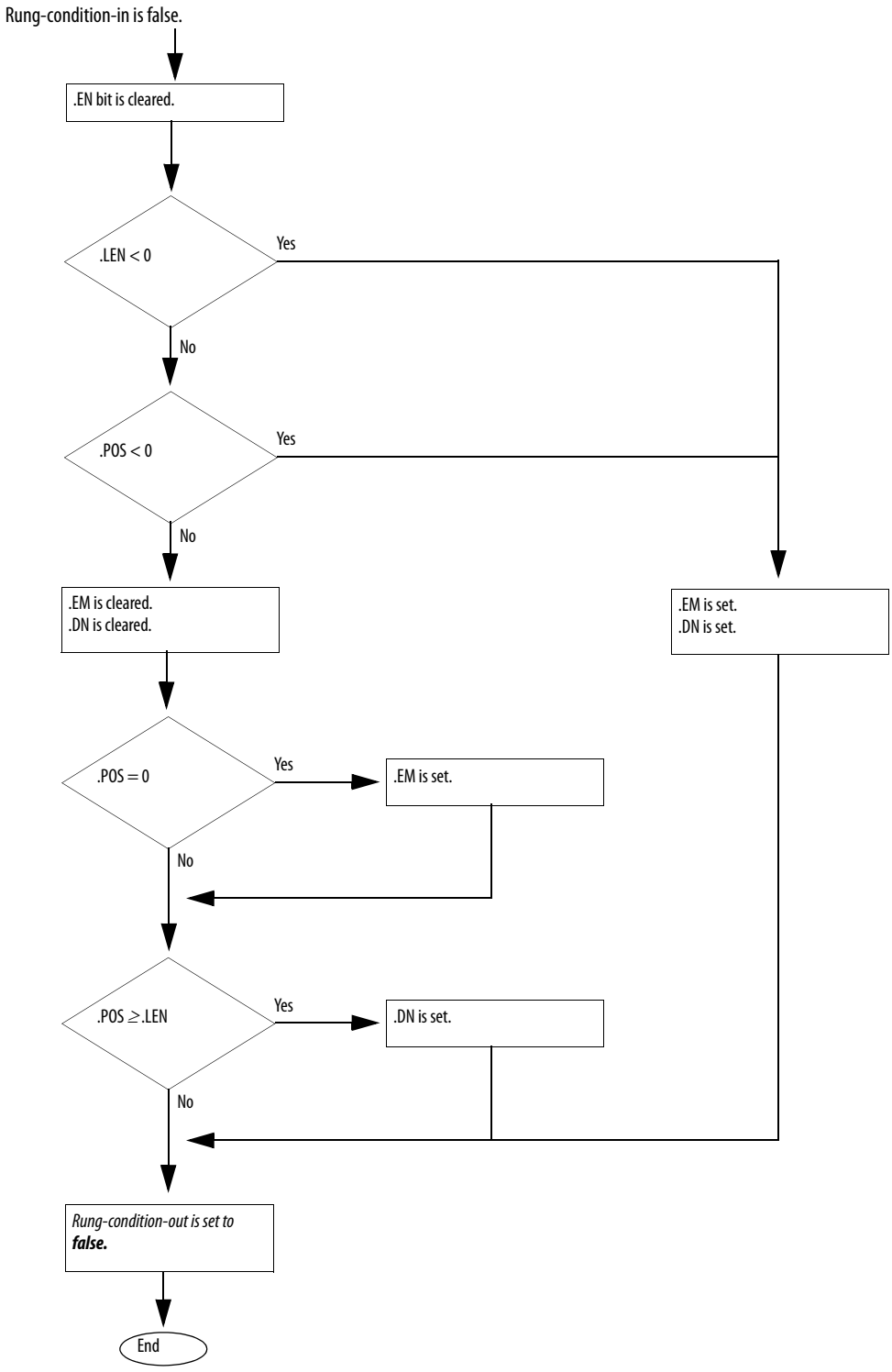
A major fault will occur if	Fault type	Fault code
(starting element + .POS) > FIFO array size	4	20

Execution:

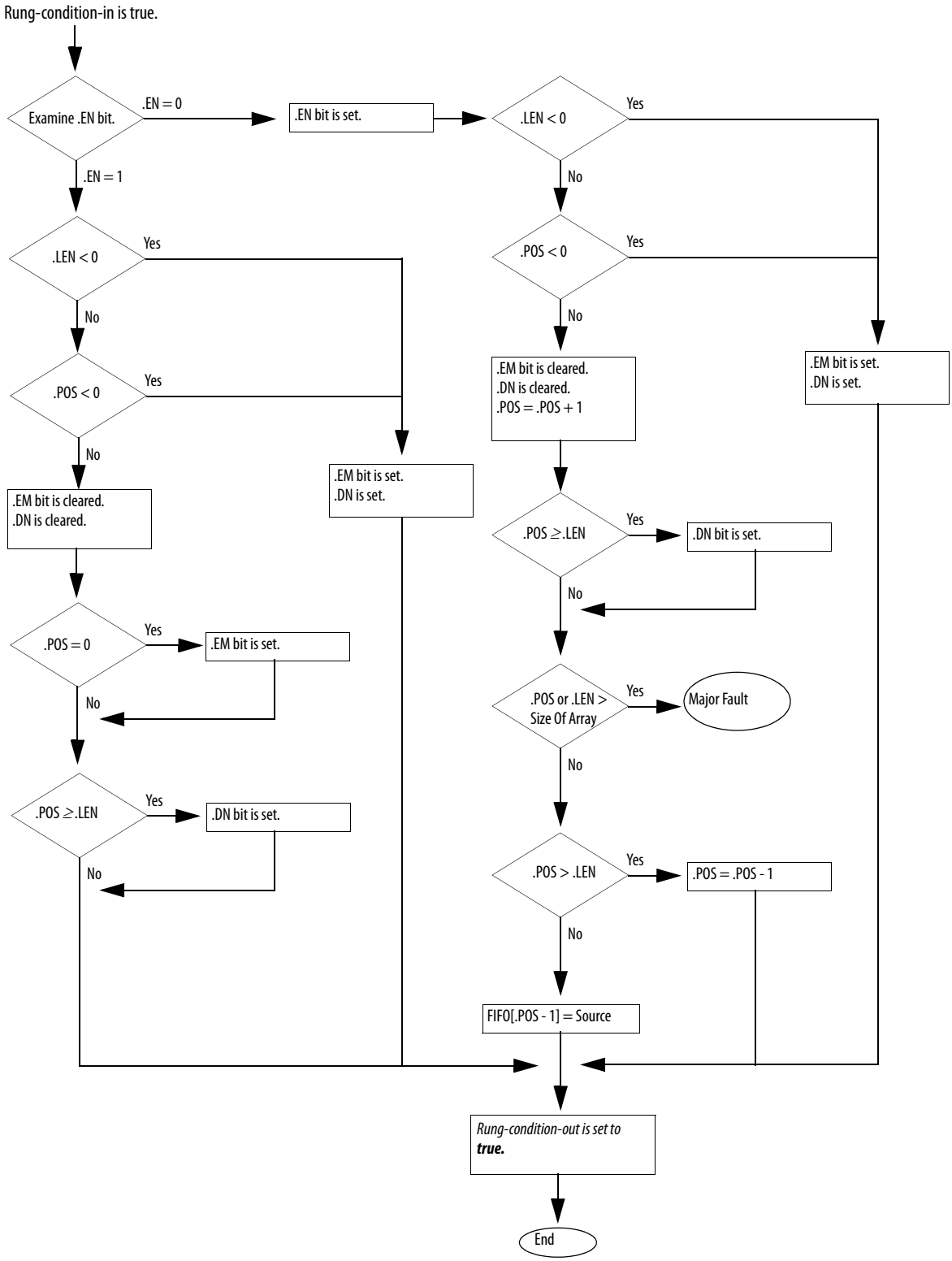
Condition	Relay Ladder Action



Condition	Relay Ladder Action

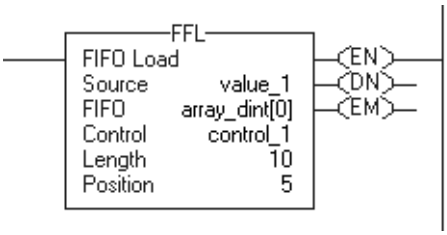


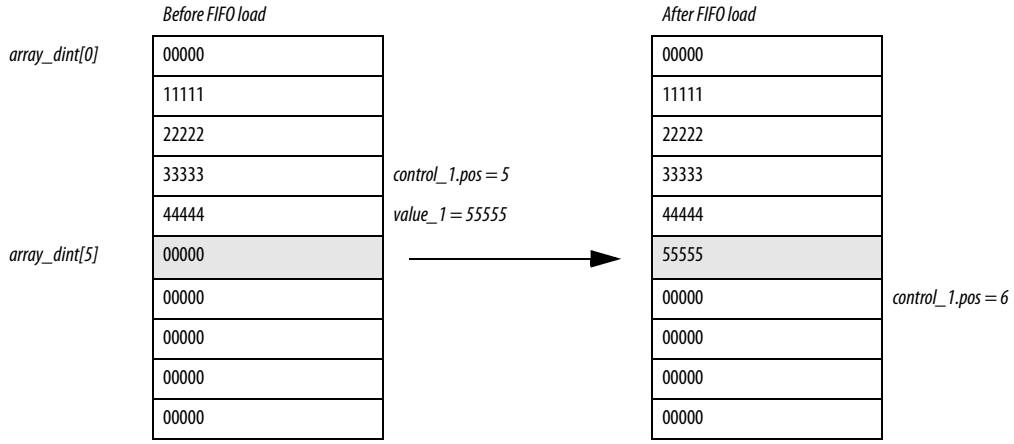
Condition	Relay Ladder Action



Postscan	The rung-condition-out is set to 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.

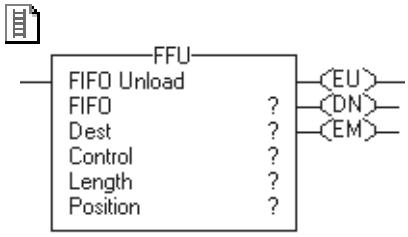




FIFO Unload (FFU)

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.

IMPORTANT You must test and confirm that the instruction doesn't change data that you don't want it to change.

The FFU instruction operates on contiguous memory. In some cases, the instruction unloads data from other members of the tag. This happens if the length is too big and the tag is a user-defined data type.

Arithmetic Status Flags:

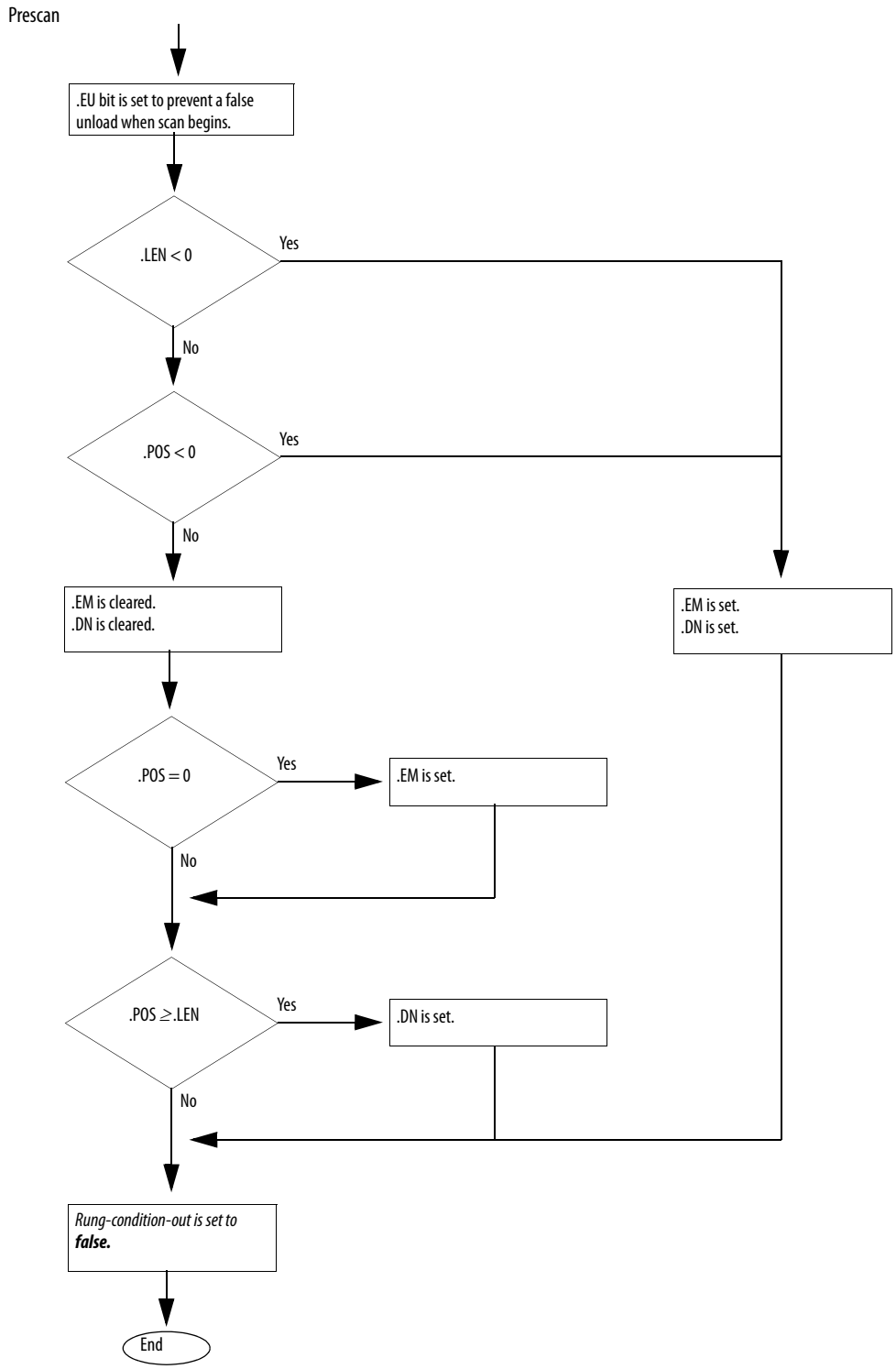
Not affected

Fault Conditions:

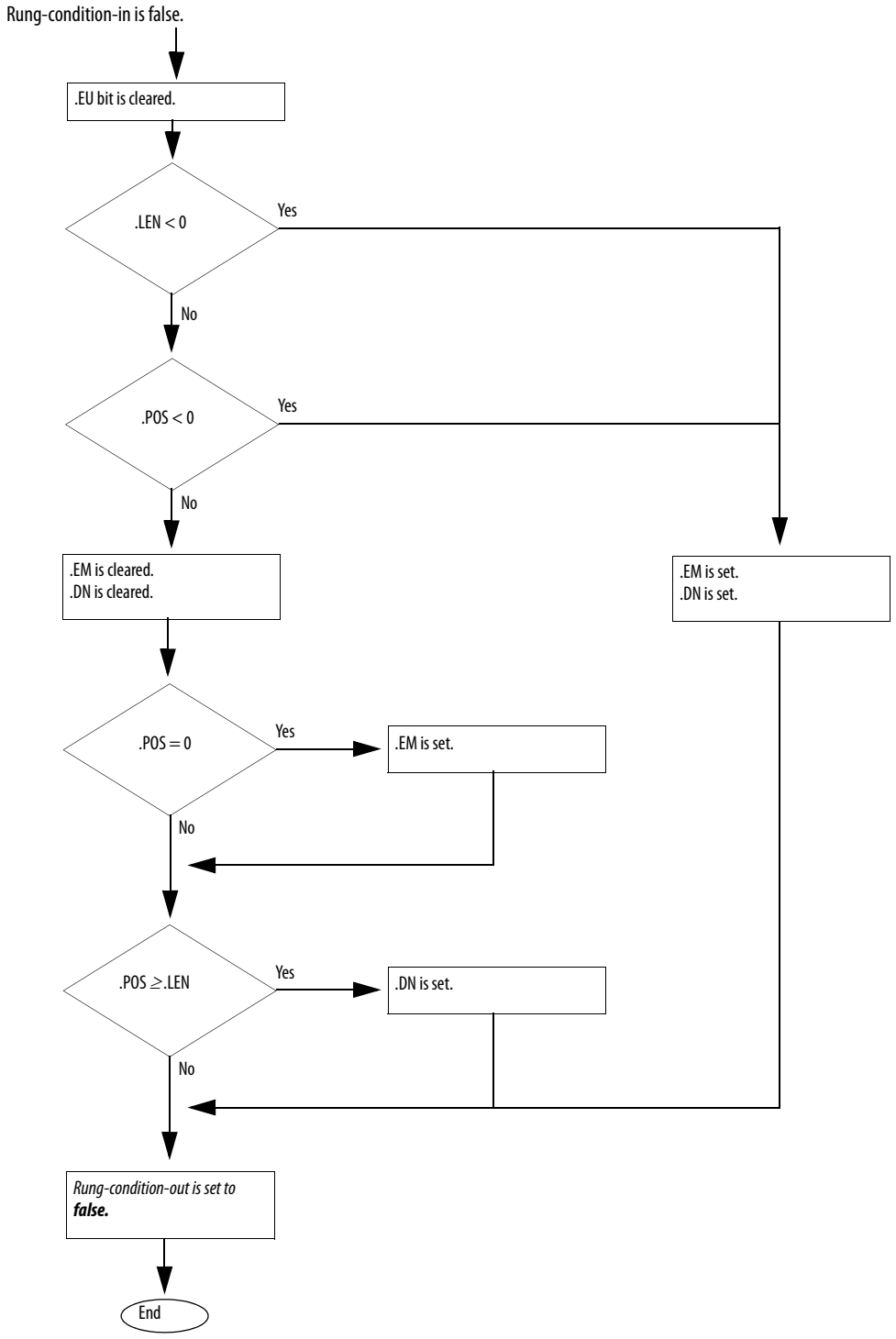
A major fault will occur if	Fault type	Fault code
Length > FIFO array size	4	20

Execution:

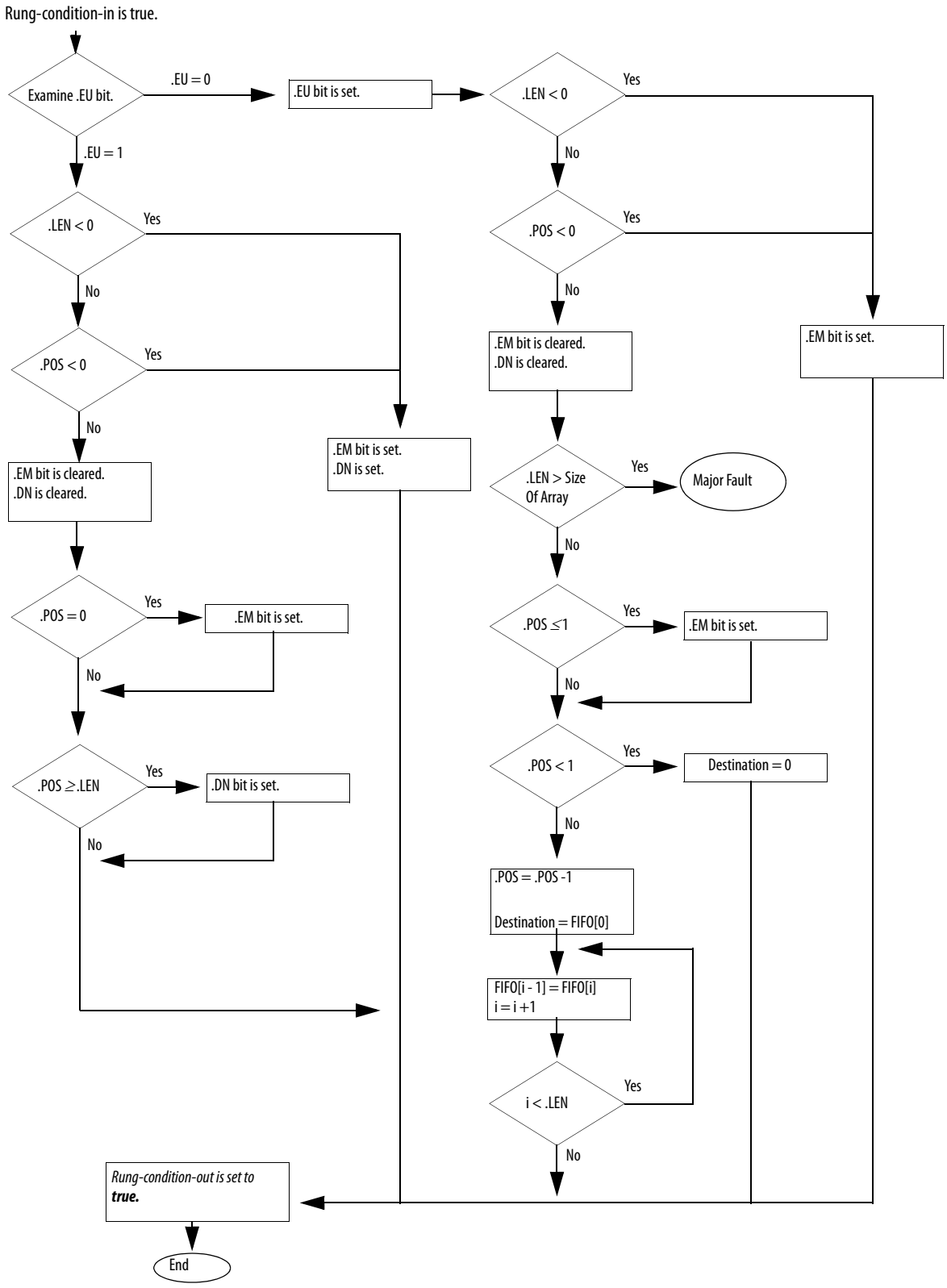
Condition	Relay Ladder Action



Condition	Relay Ladder Action

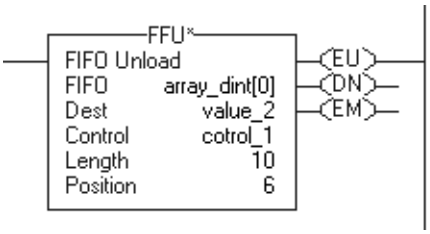


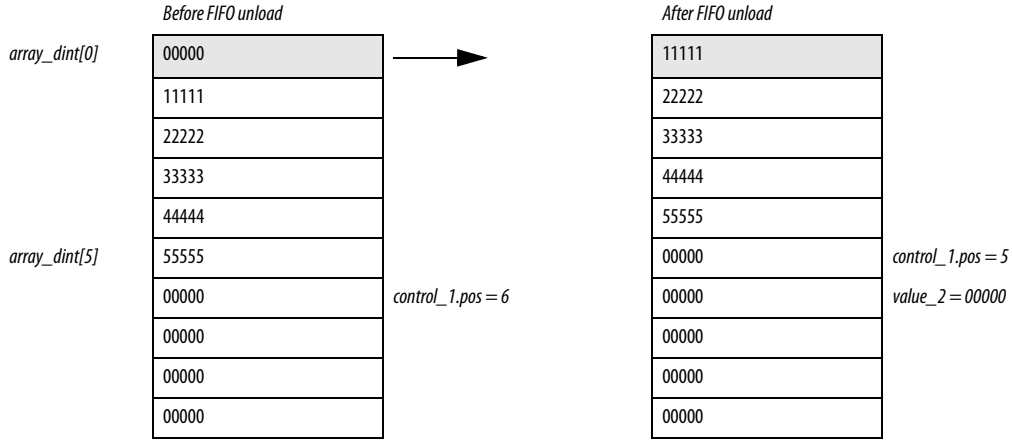
Condition	Relay Ladder Action



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*.

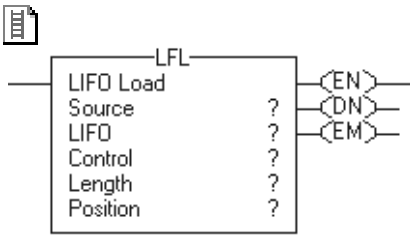




LIFO Load (LFL)

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.

IMPORTANT You must test and confirm that the instruction doesn't change data that you don't want it to change.

The LFL instruction operates on contiguous memory. In some cases, the instruction loads data past the array into other members of the tag. This happens if the length is too big and the tag is a user-defined data type.

Arithmetic Status Flags:

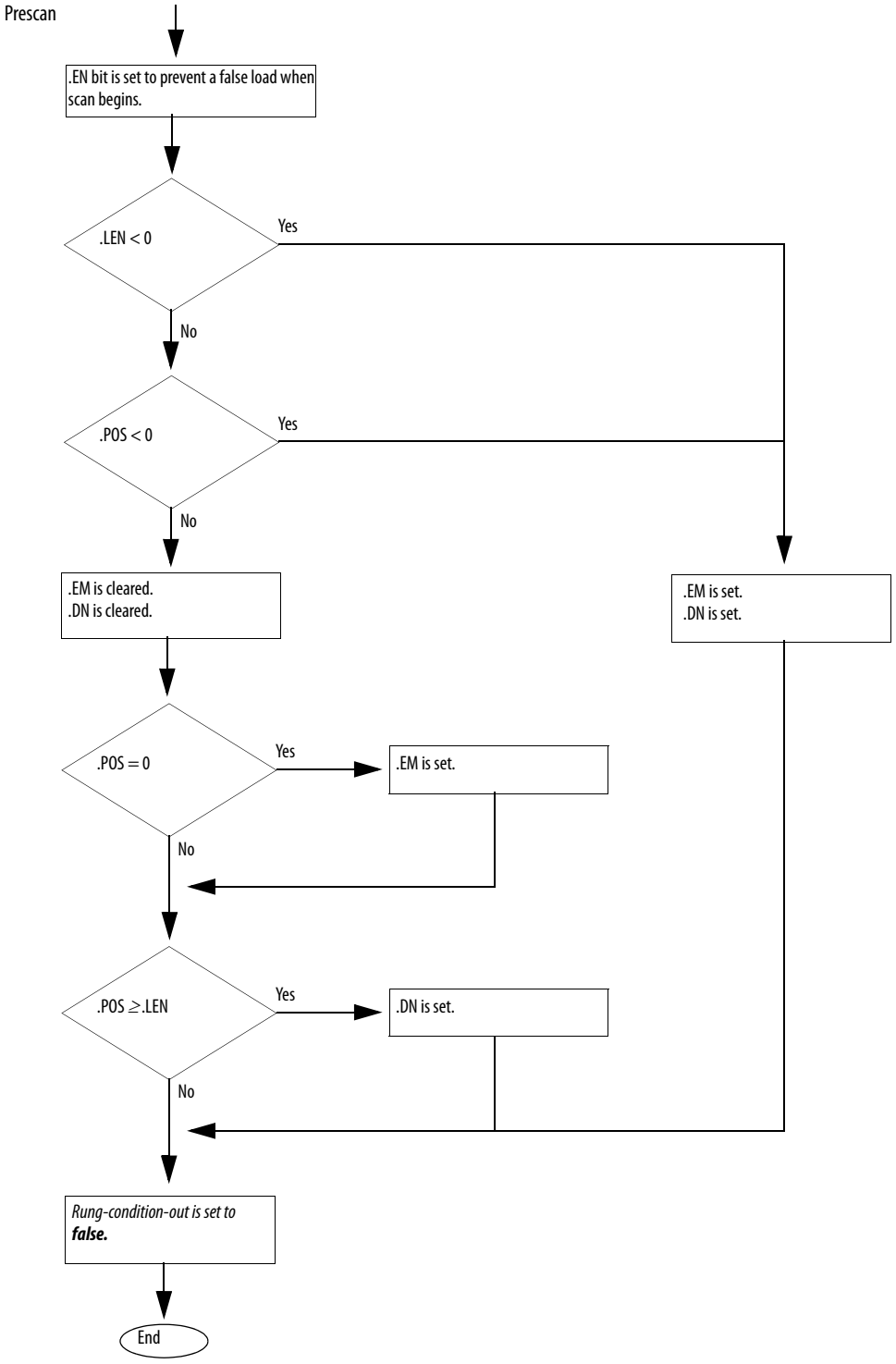
Not affected

Fault Conditions:

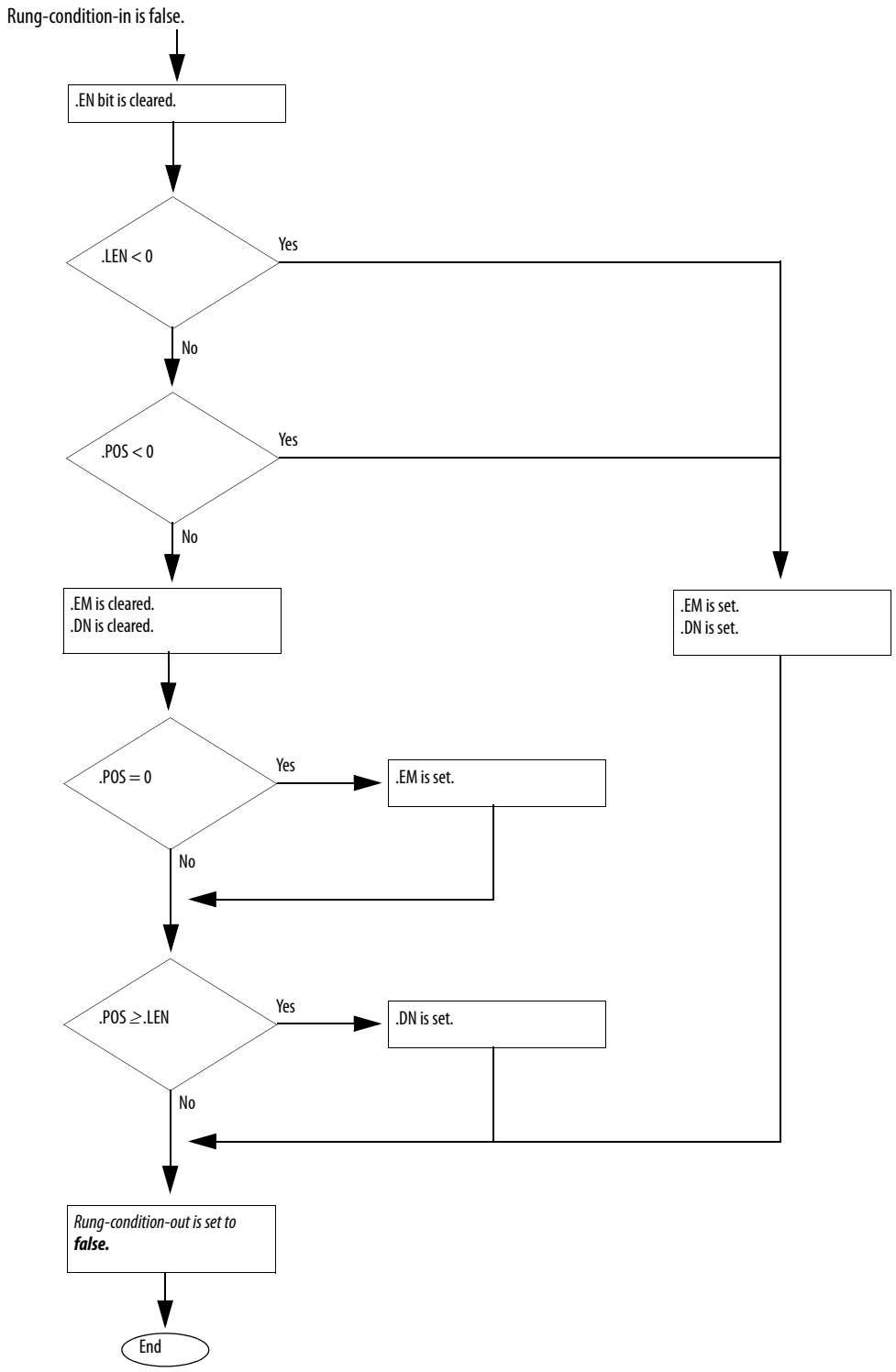
A major fault will occur if	Fault type	Fault code
(starting element + .POS) > LIFO array size	4	20

Execution:

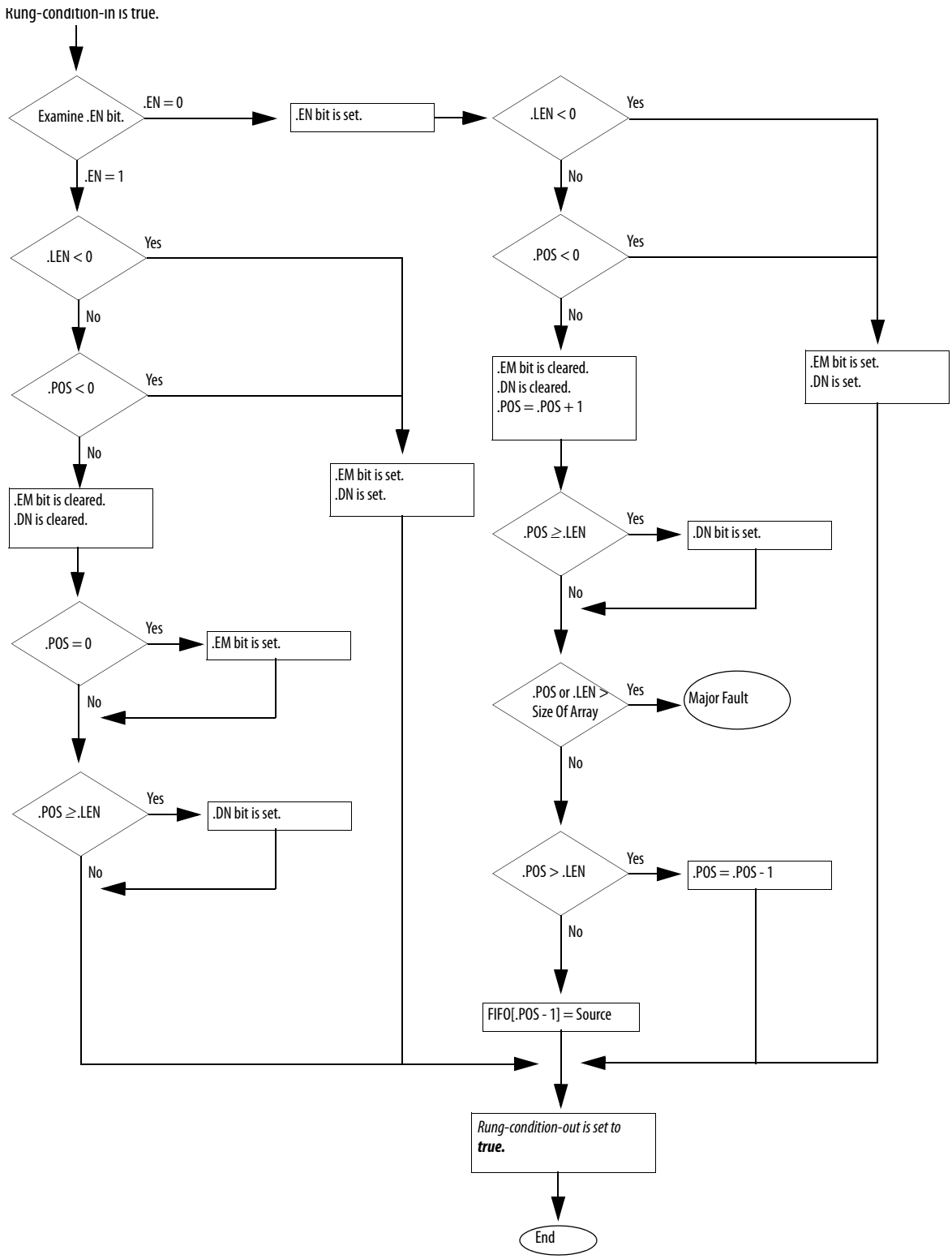
Condition	Relay Ladder Action



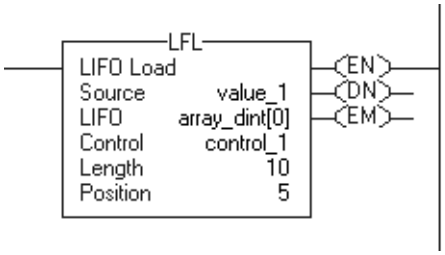
Condition	Relay Ladder Action

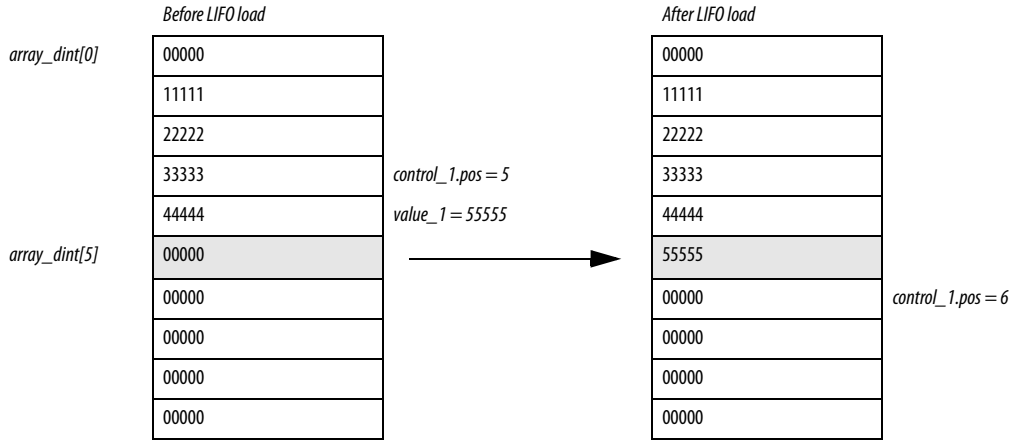


Condition	Relay Ladder Action



Example: When enabled, the LFL instruction loads *value_1* into the next position in the LIFO, which is *array_dint[5]* in this example.

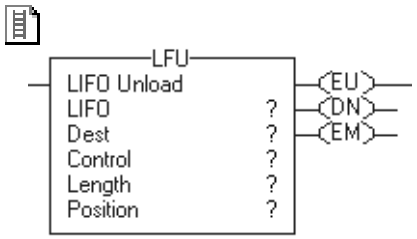




LIFO Unload (LFU)

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.

IMPORTANT You must test and confirm that the instruction doesn't change data that you don't want it to change.

The LFU instruction operates on contiguous memory. In some cases, the instruction unloads data from other members of the tag. This happens if the length is too big and the tag is a user-defined data type.

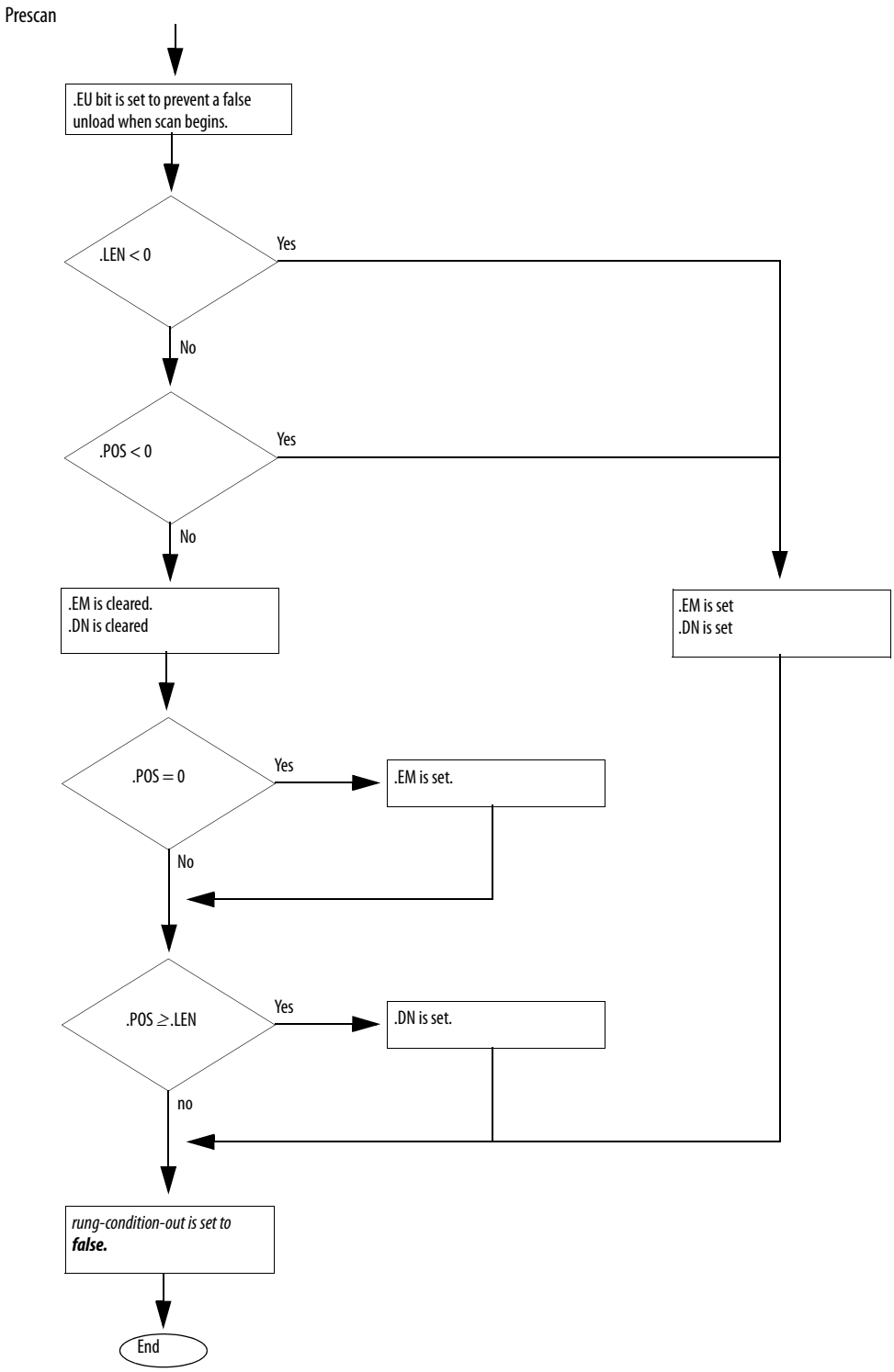
Arithmetic Status Flags: Not affected

Fault Conditions:

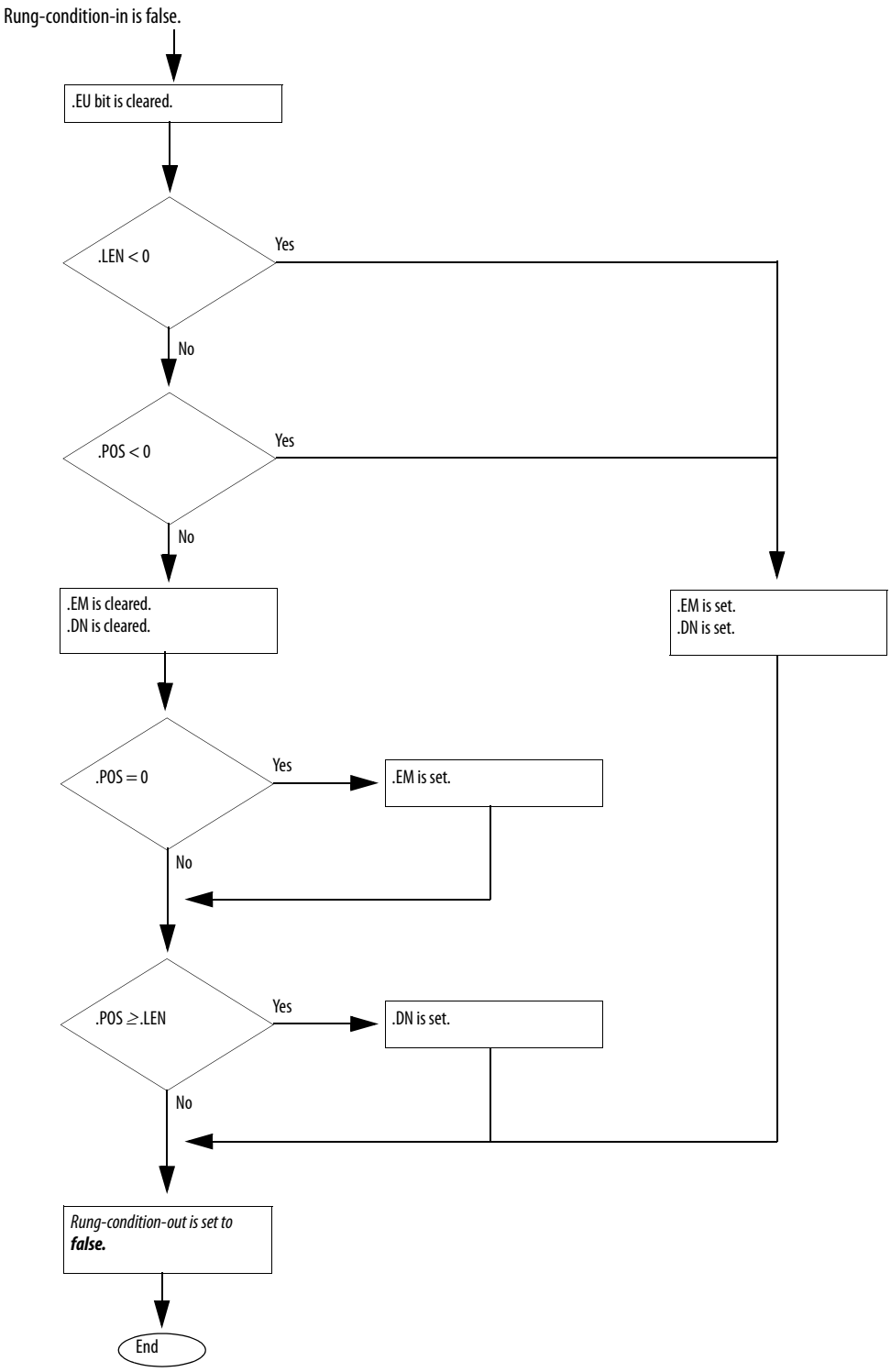
A major fault will occur if	Fault type	Fault code
Length > LIFO array size	4	20

Execution:

Condition	Relay Ladder Action

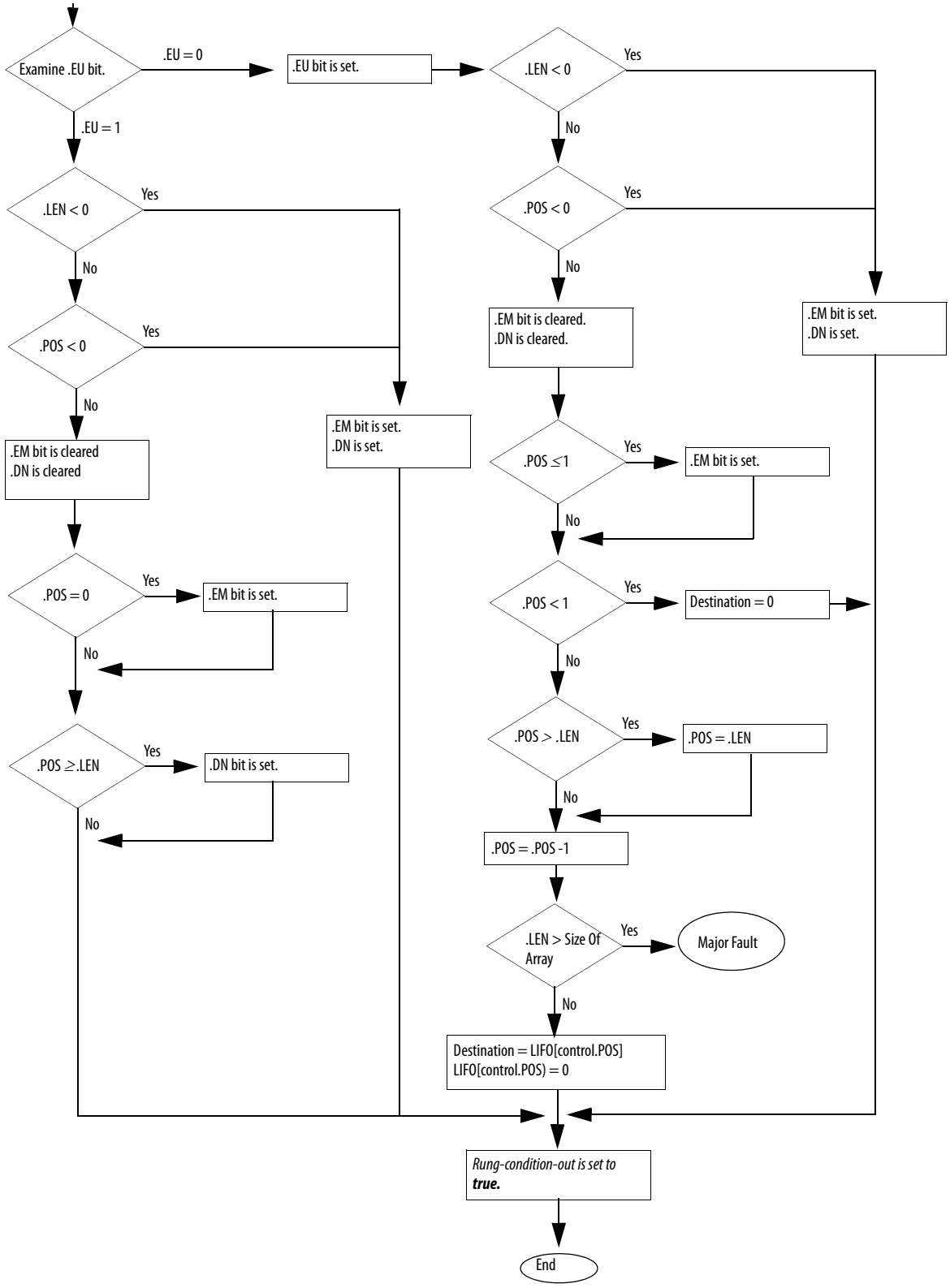


Condition	Relay Ladder Action



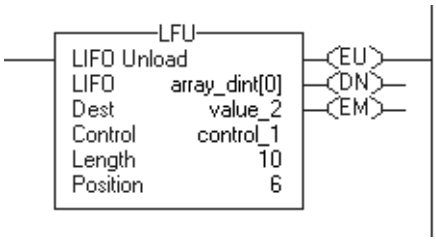
Condition	Relay Ladder Action

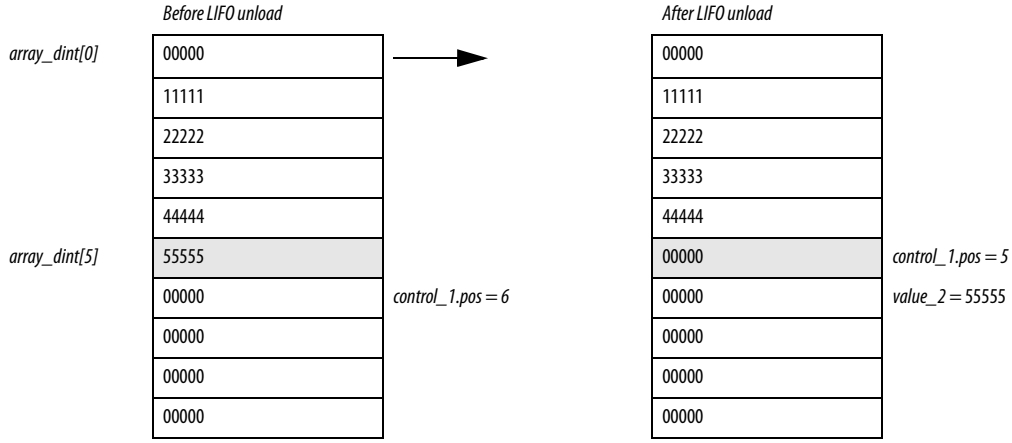
Rung-condition-in is true.



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)

No action taken. Sequencer instructions monitor consistent and repeatable operations.

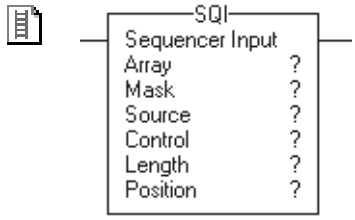
If you want to	Use this instruction	Available in these languages	Page
Detect when a step is complete	SQI	Relay ladder	432
Set output conditions for the next step	SQO	Relay ladder	436
Load reference conditions into sequencer arrays	SQL	Relay ladder	440

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 ≤ 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.

Enter an Immediate Mask Value

When you enter a mask, the programming software defaults to decimal values. If you want to enter a mask by 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

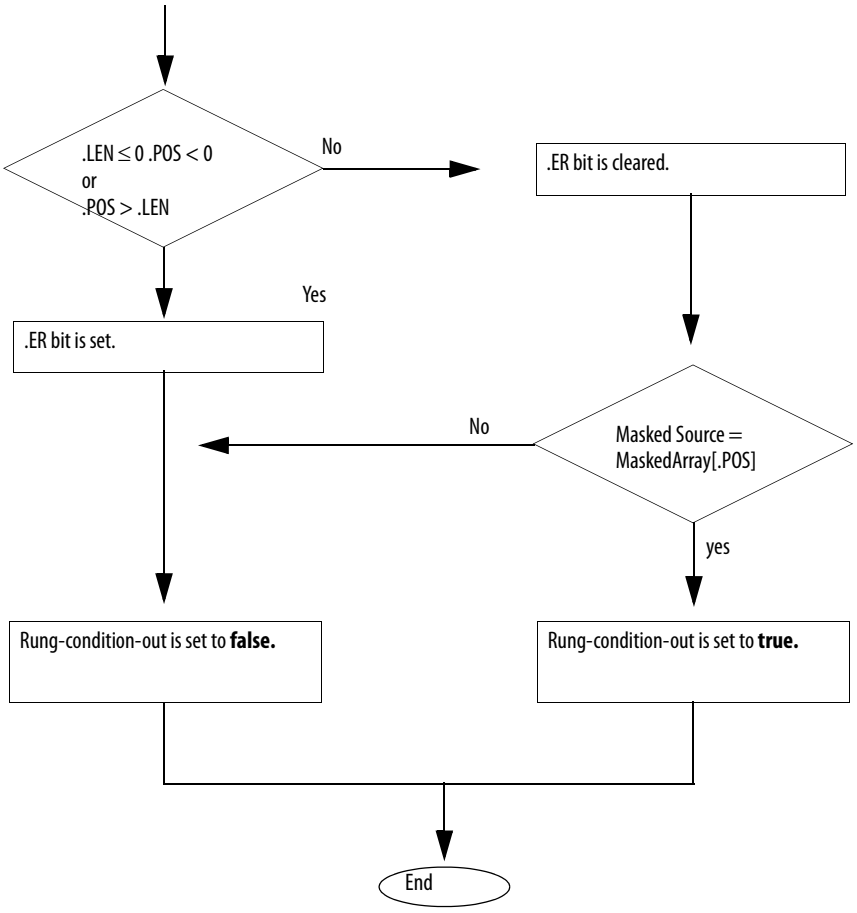
Arithmetic Status Flags: Not affected

Fault Conditions: None

Execution:

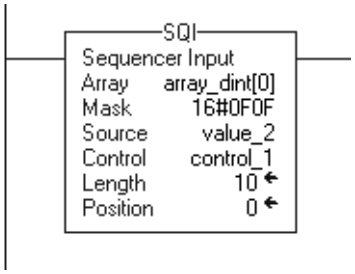
Condition:	Relay Ladder Action
Prescan	The rung-condition-out is set to false.
Rung-condition-in is false	The rung-condition-out is set to false.

Rung-condition-in is true.



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.



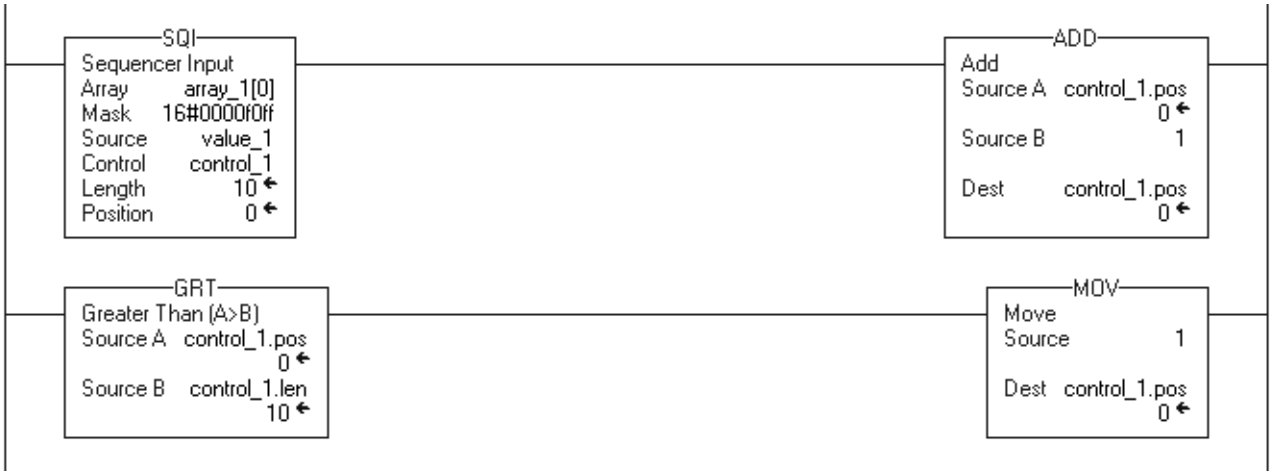
SQI Operand	Example Values (DINTs Displayed In Binary)
Source	`xxxxxxxx xxxxxxxx xxxx0101 xxxx1010`
Mask	`00000000 00000000 00001111 00001111`
Array	`xxxxxxxx xxxxxxxx xxxx0101 xxxx1010`

A 0 in the mask means the bit is not compared (designated by sextets in this example).

Use SQI without SQO

If you use the SQI instruction without a paired SQO instruction, you have to externally increment the sequencer array.

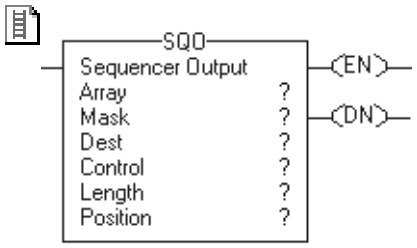
The SQI instruction compares the source value. The ADD instruction increments the sequencer array. The GRT determines whether another value is available to check in the sequencer array. The MOV instruction resets the position value after completely stepping through the sequencer array one time.



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.

Enter an Immediate Mask Value

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

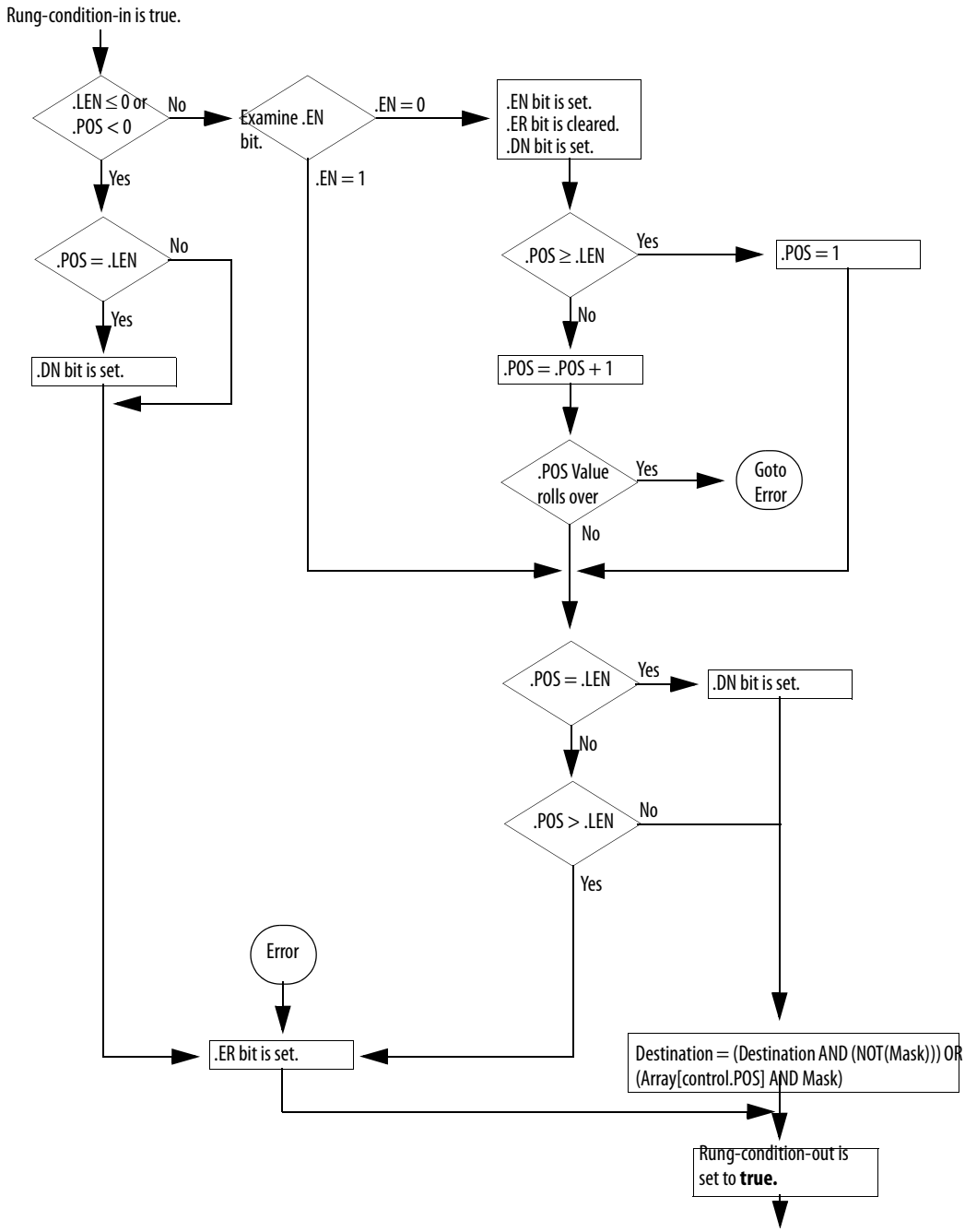
Prefix	Description
16#	Hexadecimal For example; 16#0F0F
8#	Octal For example; 8#16
2#	Binary For example; 2#00110011

Arithmetic Status Flags Not affected

Fault Conditions: None

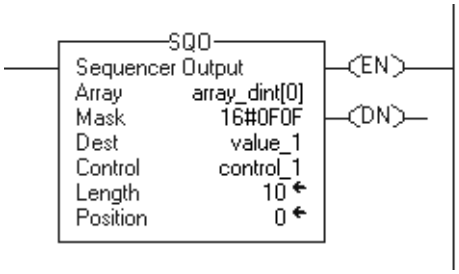
Execution:

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.



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.

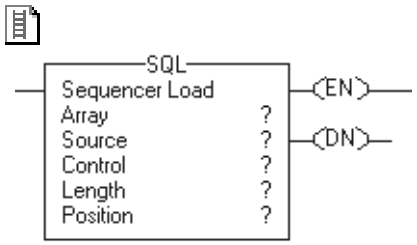
Resetting the Position of SQO

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 reset 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.

IMPORTANT You must test and confirm that the instruction doesn't change data that you don't want it to change.

The SQL instruction operates on contiguous memory. In some cases, the instruction loads data past the array into other members of the tag. This happens if the length is too big and the tag is a user-defined data type.

Arithmetic Status Flags:

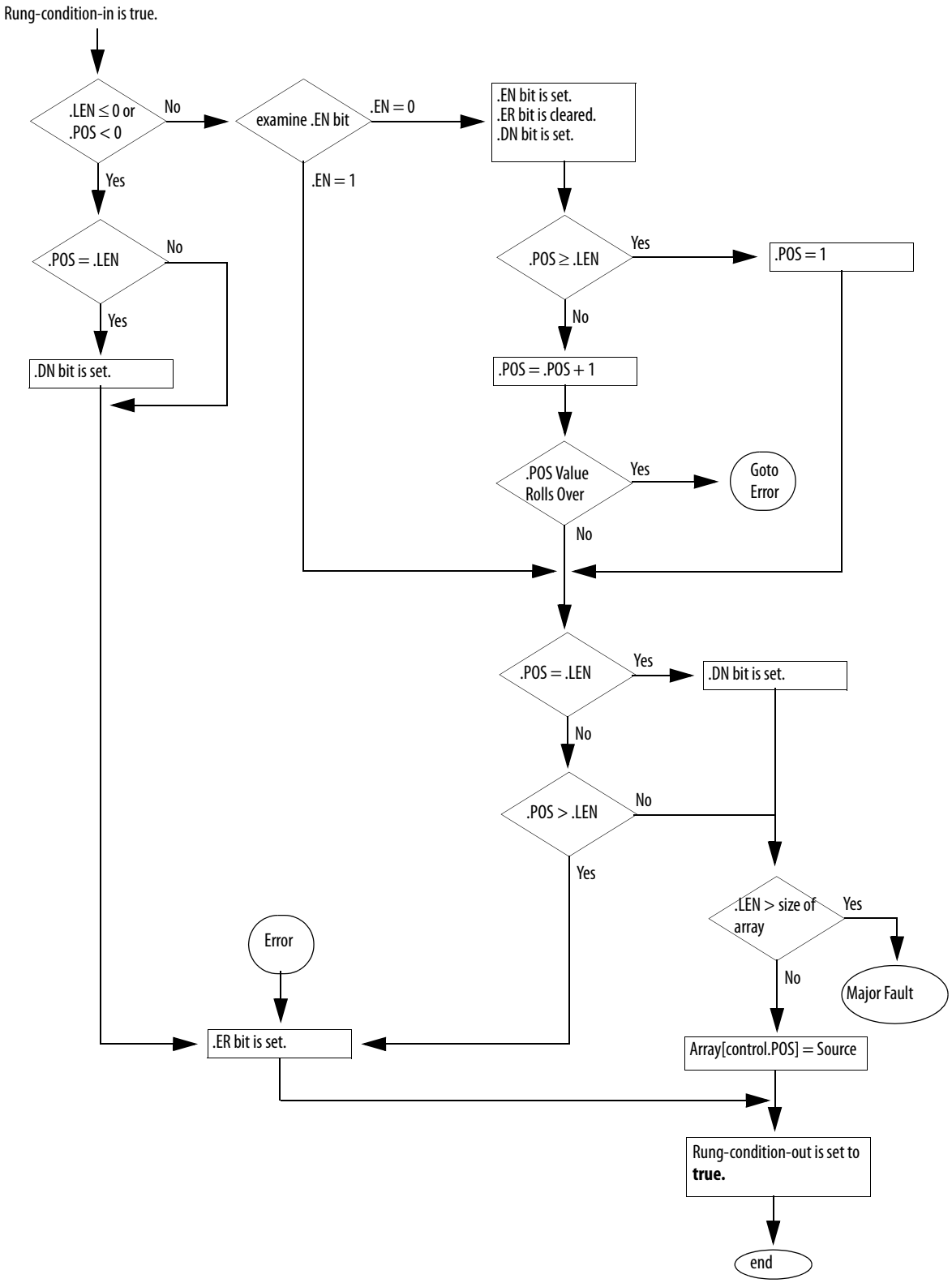
Not affected

Fault Conditions:

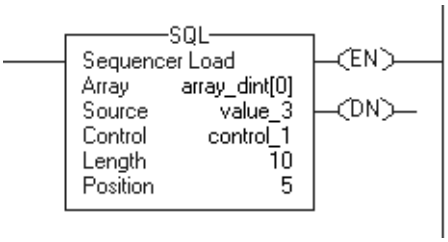
A major fault will occur if	Fault type	Fault code
Length > size of Array	4	20

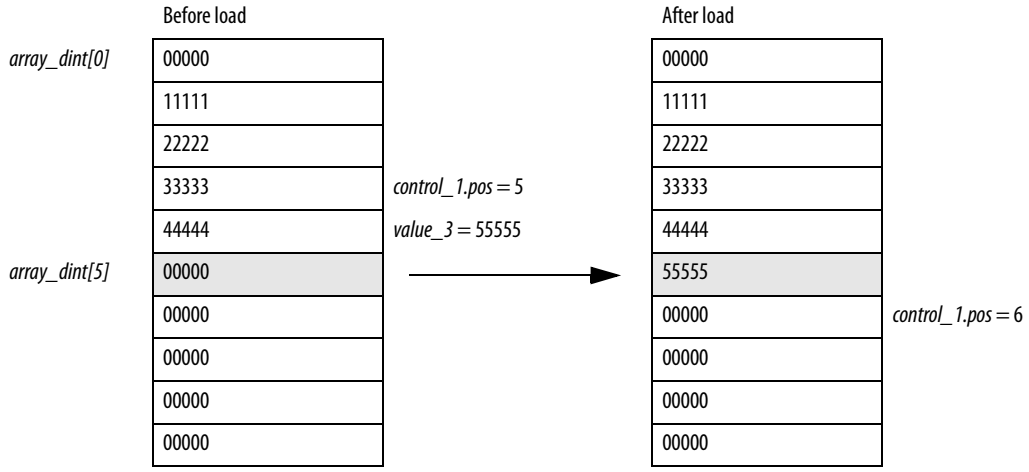
Execution:

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)

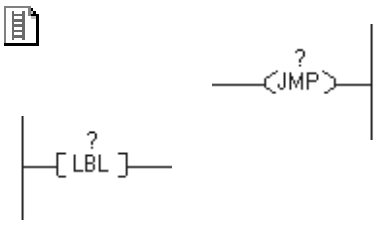
Use the program control instructions to change the flow of logic.

If you want to	Use this instruction	Available in these languages	Page
Jump over a section of logic that does not always need to be executed	JMP LBL	Relay ladder	447
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	436
Jump to an external routine (SoftLogix5800 controller only)	JXR	Relay ladder	459
Mark a temporary end that halts routine execution	TND	Relay ladder Structured text	462
Disable all the rungs in a section of logic	MCR	Relay ladder	464
Disable user tasks	UID	Relay ladder Structured text	466
Enable user tasks	UIE	Relay ladder Structured text	466
Disable a rung	AFI	Relay ladder	468
Insert a placeholder in the logic	NOP	Relay ladder	469
End a transition for a sequential function chart	EOT	Relay ladder Structured text	470
Pause a sequential function chart	SFP	Relay ladder Structured text	472
Reset a sequential function chart	SFR	Relay ladder Structured text	474
Trigger the execution of an event task	EVENT	Relay ladder Structured text	476

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 include the following:

- Have as many as 40 characters
- Contain letters, numbers, and underscores (_)

Arithmetic Status Flags:

Not affected

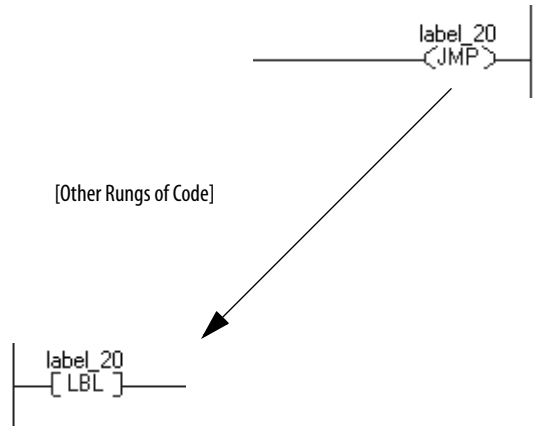
Fault Conditions:

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.

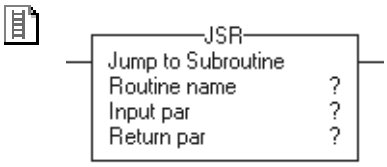
Example: When the JMP instruction is enabled, execution jumps over successive rungs of logic until it reaches the rung that contains the LBL instruction with *label_20*.



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:



Relay Ladder

Operand	Type	Format	Description
Routine name	ROUTINE	Name	Routine to execute (that is, 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 • 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 • Return parameters are optional. • Enter multiple return parameters, if needed.

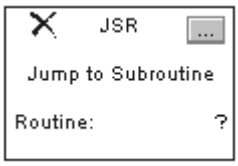
JSR Operands Continued:

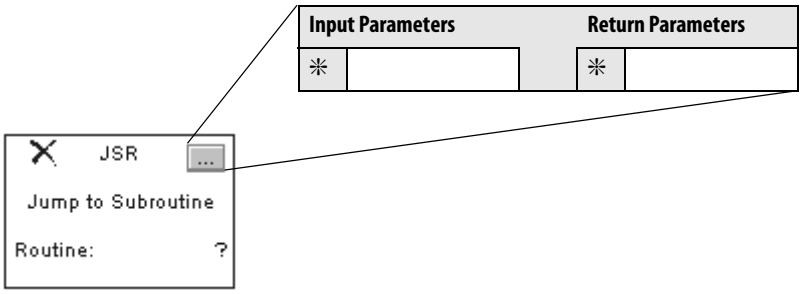
Structured Text

JSR(RoutineName,InputCount, InputPar,ReturnPar);

Operand	Type	Format	Description
Routine name	ROUTINE	Name	Routine to execute (that is, 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 • 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 • Return parameters are optional. • Enter multiple return parameters, if needed.

Function Block



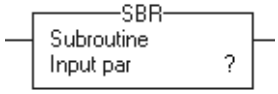


The operands are the same as those for the relay ladder JSR instruction.

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.



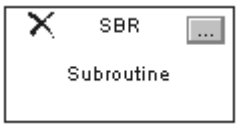
Relay Ladder

Operand	Type	Format	Description
Input parameter	BOOL SINT INT DINT REAL Structure	Tag Array tag	Tag in this routine into which you want to copy the corresponding input parameter from the JSR instruction

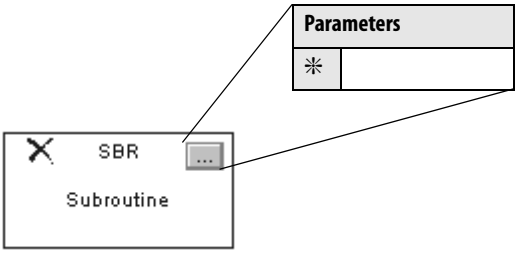
SBR(InputPar);

Structured Text

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

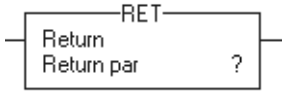


Function Block



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

RET Operands:



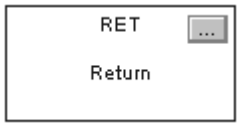
Relay Ladder

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

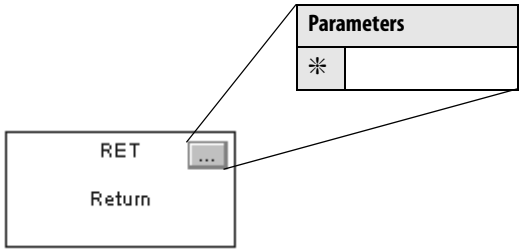
RET(ReturnPar);

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.

Use these guidelines to program a jump to a subroutine.

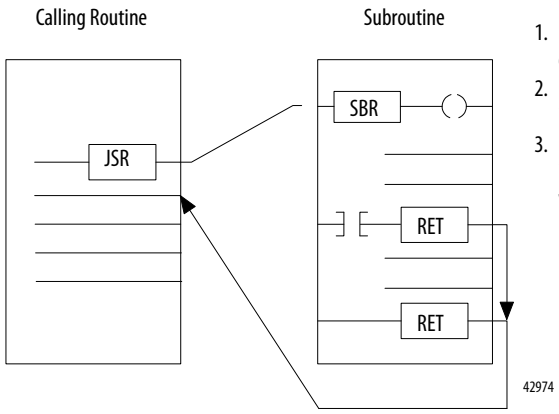
IMPORTANT Do not use a JSR instruction to call (execute) the main routine.

- 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.



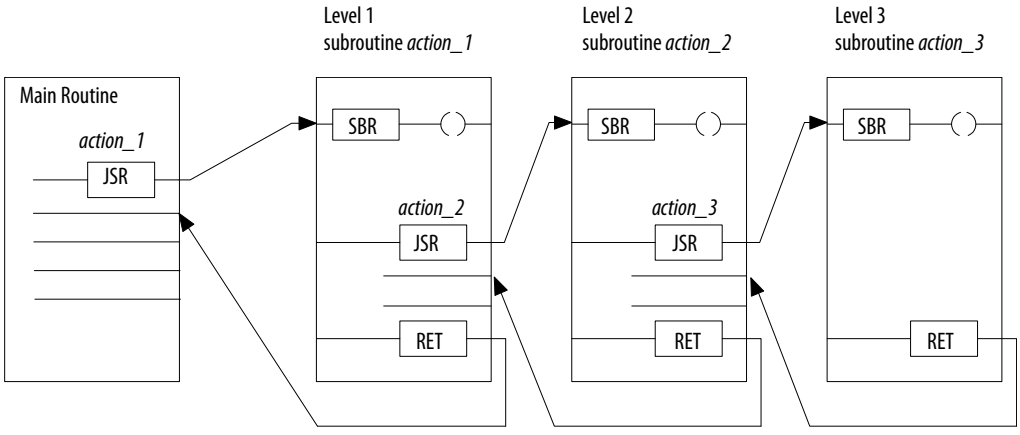
SBR

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.

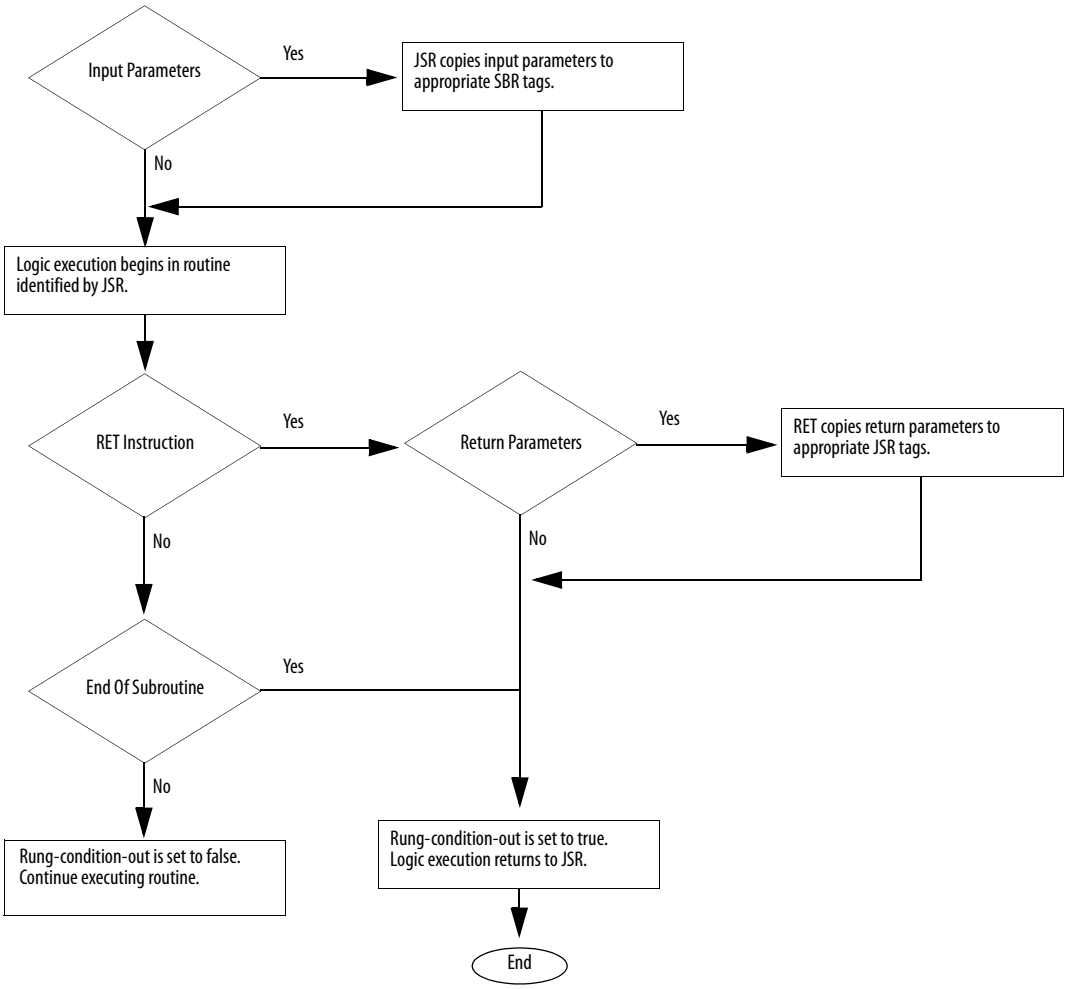
Fault Conditions:

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	The controller executes all subroutines regardless of rung condition. To ensure that all rungs in the subroutine are prescanned, the controller ignores RET instructions. (that is, RET instructions do not exit the subroutine.) • Release 6.*x* and earlier, input and return parameters are passed. • Release 7.*x* and later, input and return parameters are not passed. 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).	
Rung-condition-in is false to the JSR instruction	The subroutine **does not** execute. Outputs in the subroutine remain in their last state. The rung-condition-out is set to false.	N/A
Rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	N/A
EnableIn is set	N/A	EnableIn is always set. The instruction executes.

Condition	Relay Ladder Action	Structured Text Action
Instruction execution.		
Postscan	Same action as prescan described above.	Same action as prescan described above.

Function Block

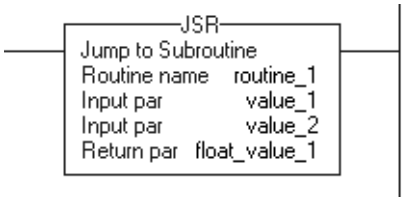
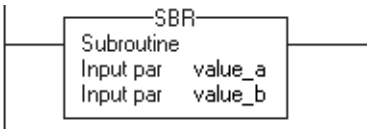
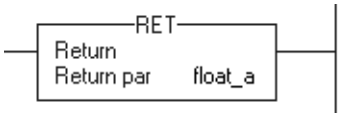
Condition	Action
Prescan	No action taken.
Instruction first scan	No action taken.
Instruction first run	No action taken.
Normal execution	1. If the routine contains an SBR instruction, the controller first executes the SBR instruction. 2. The controller latches all data values in IREFs. 3. The controller 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 in 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

Routine	Program
Main routine	
Subroutine	 *[Other rungs of code]* 

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);

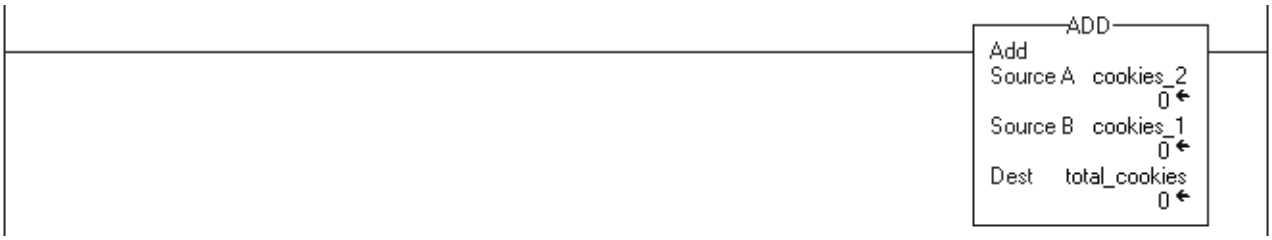
Example 2:

Relay Ladder

MainRoutine

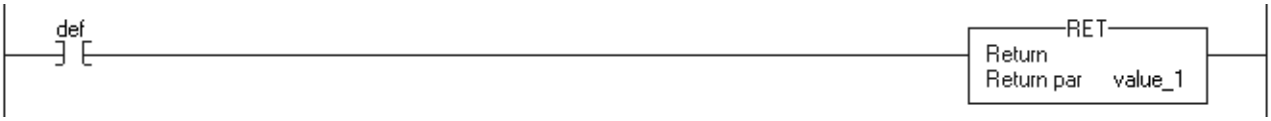
When *abc* is on, *subroutine_1* executes, calculates the number of cookies, and places a value in *cookies_1*.

Adds the value in *cookies_1* to *cookies_2* and stores the result in *total_cookies*.

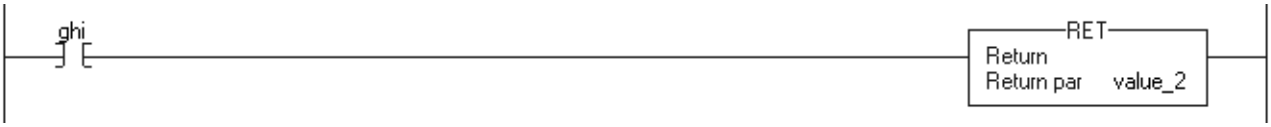


Subroutine_1

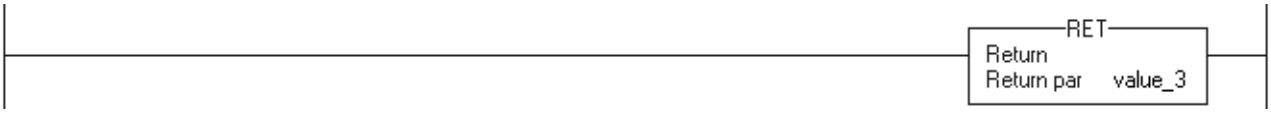
When *def* is on, the RET instruction returns *value_1* to the JSR *cookies_1* parameter and the rest of the subroutine is not scanned.



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.

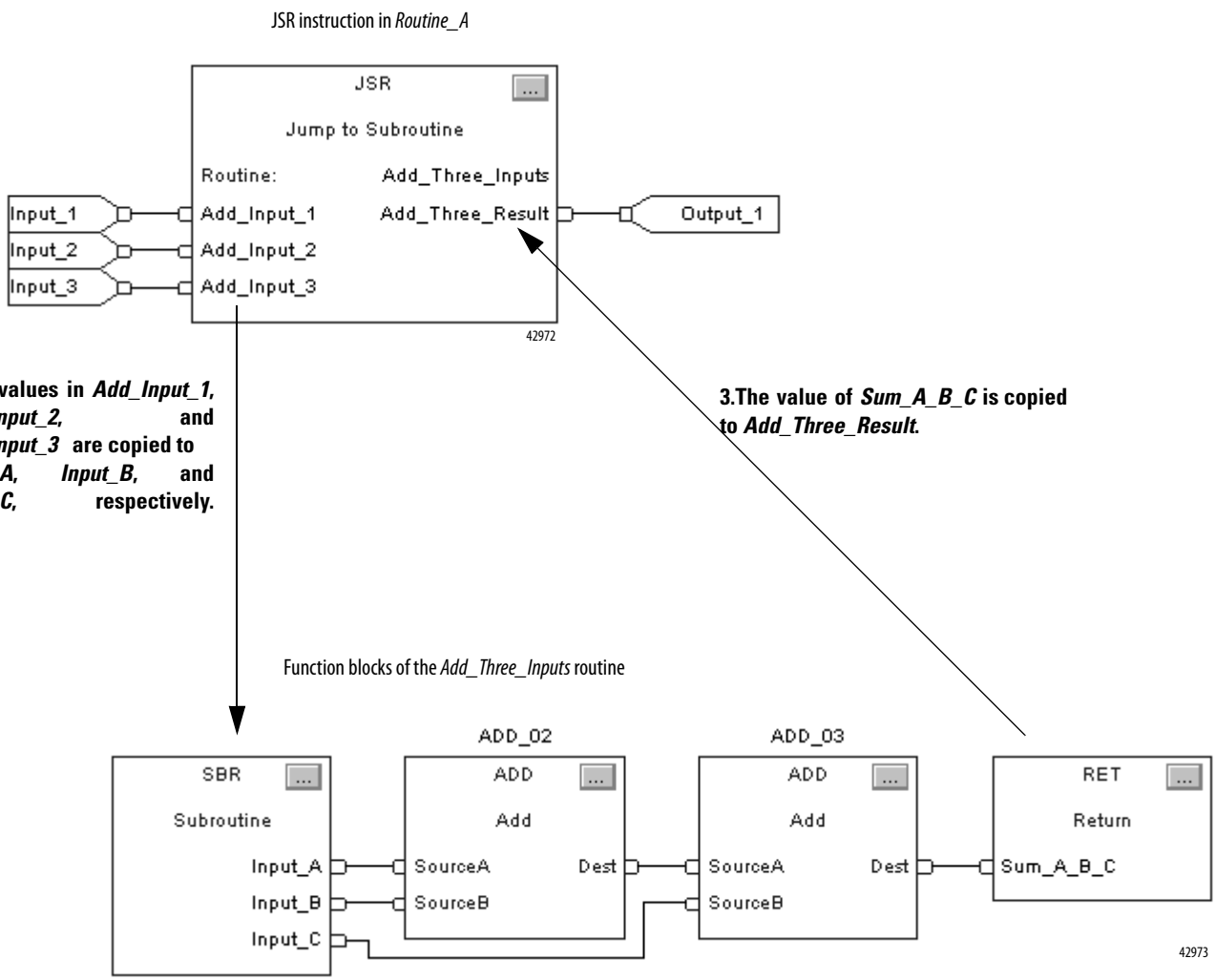


When both *def* and *ghi* are off (previous rungs), the RET instruction returns *value_3* to the JSR *cookies_1* parameter.



Example 3:

Function Block

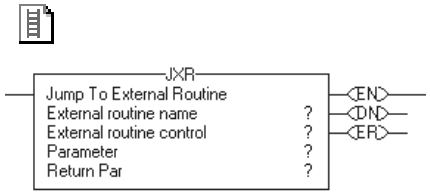


Jump to External Routine (JXR)

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
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 • 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 • 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_ROUTIN_ 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:
• 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:

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:

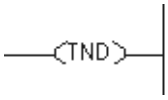
Condition	Relay Ladder Action	Structured Text Action
Prescan	The rung-condition-out is set to false.	No action taken.
Rung-condition-in is false	The rung-condition-out is set to false.	N/A
Rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	N/A
EnableIn is set	N/A	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:

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 the following:

- 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.

ATTENTION: 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.

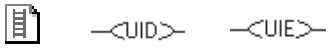
Example: When the first MCR instruction is enabled (*input_1*, *input_2*, and *input_3* are set), the controller executes the rungs in the MCR zone (between the two MCR instructions) and sets or clears outputs, depending on input conditions.

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:



Relay Ladder

None

UID();

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.

Follow these steps 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:

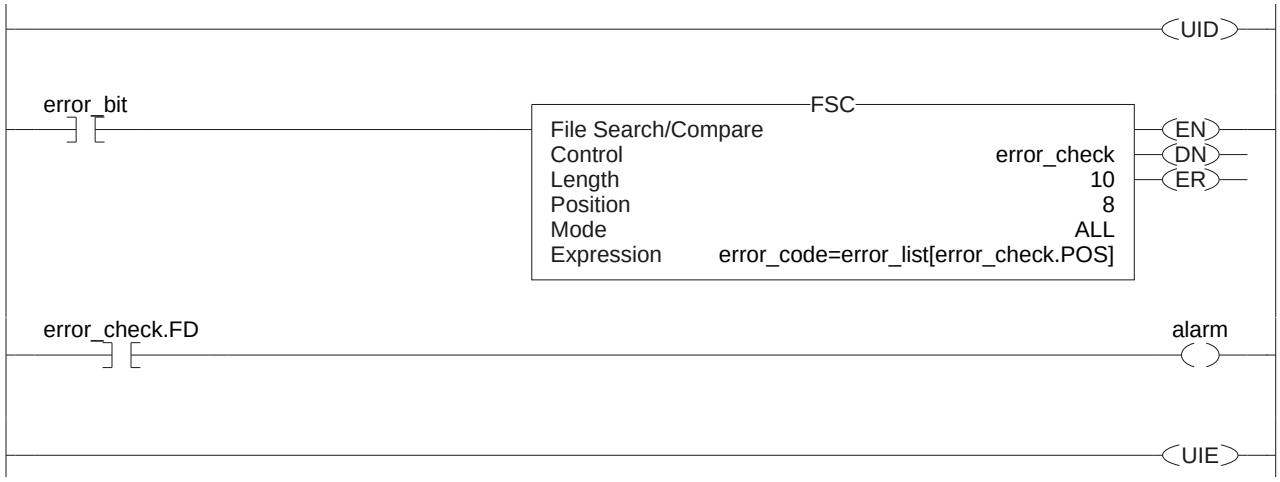
None

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.	N/A
Rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	N/A
EnableIn is set	N/A	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:

Relay Ladder

None

Description:

The AFI instruction sets its rung-condition-out to false.

Arithmetic Status Flags:

Not affected

Fault Conditions:

None

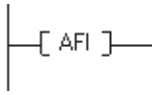
Execution:

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:

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 462).

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

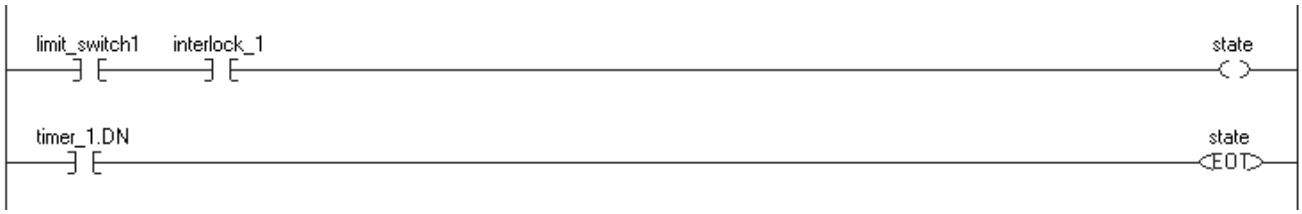
Execution:

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.	N/A
Rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	N/A
EnableIn is set	N/A	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

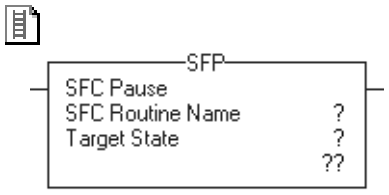
```
state := limit_switch1 AND interlock_1;

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)

Structured Text

```
SFP(SFCRoutineName,
TargetState);
```

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 474) to reset an SFC routine.

Arithmetic Status Flags:

Not affected

Fault Conditions:

A major fault will occur if	Fault type	Fault code
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.	N/A
Rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	N/A
EnableIn is set	N/A	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.

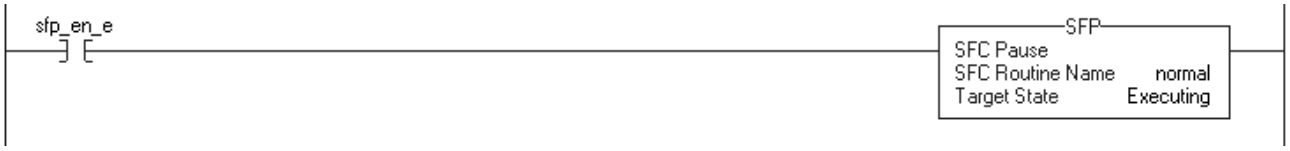
Example:

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;
```

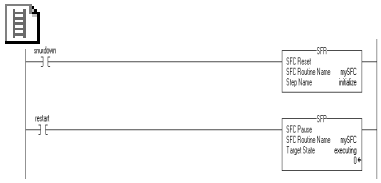
Resume executing the SFC routine:

```
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

Structured Text

```
SFR(SFCRoutineName,StepName);
```

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
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.	N/A
Rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	N/A
EnableIn is set	N/A	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

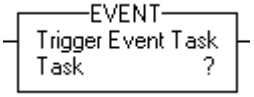
Structured Text

```
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 7 - 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.	
			To determine if	**Examine this bit**
			An EVENT instruction triggered the task (event task only).	0
			A time-out 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.	N/A
Rung-condition-in is true	The instruction executes.	N/A
	The rung-condition-out is set to true.	
EnableIn is set	N/A	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.

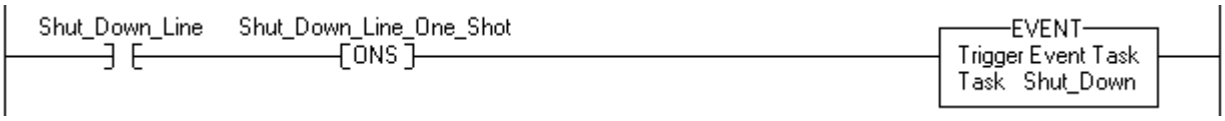
Example 1: A controller uses multiple programs but a common shut down procedure. Each program uses a program-scoped tag named *Shut_Down_Line* that turns on if the program detects a condition that requires a shut down. The logic in each program executes as follows:

If *Shut_Down_Line* = on (conditions require a shut down) then

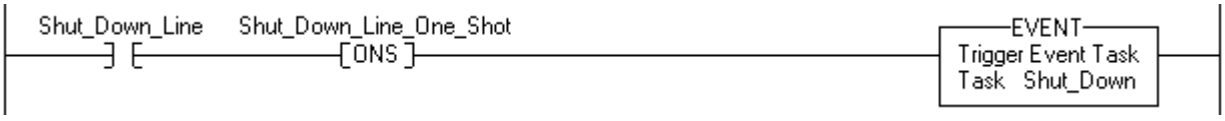
Execute the *Shut_Down* task one time

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;
```

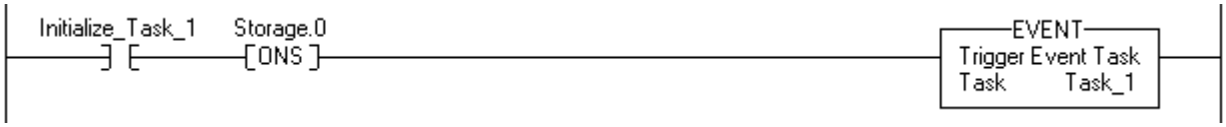
Example 2: The following example uses an EVENT instruction to initialize an event task. (Another type of event normally triggers the event task.)

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).

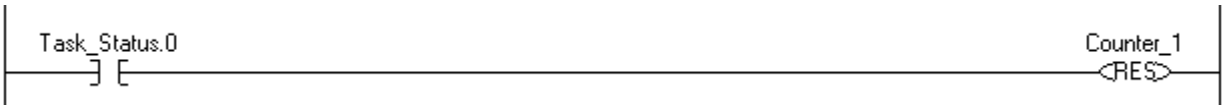


Task_1 (event task)

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 (that is, *Task_1*).

If *Task_Status*.0 = 1 then an EVENT instruction triggered the event task (that is, when the continuous task executes its EVENT instruction to initialize the event task).

The RES instruction resets a counter that the event task uses.

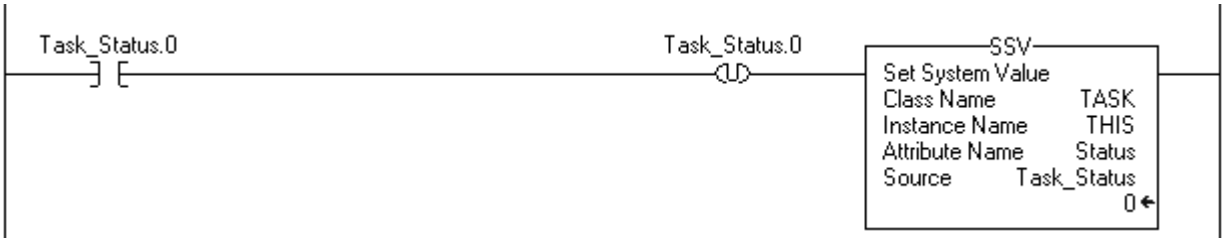


The controller does not clear the bits of the Status attribute once they are set. To use a bit for new status information, you must manually clear the bit.

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.



Notes:

For/Break Instructions

(FOR, FOR...DO, BRK, EXIT, RET)

Use the FOR instruction to repeatedly call a subroutine. Use the BRK instruction to interrupt the execution of a subroutine.

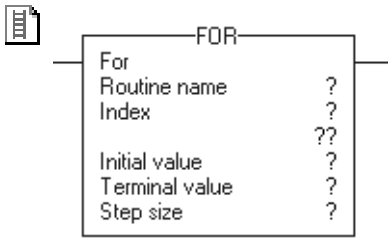
If you want to	Use this instruction	Available in these languages	Page
Repeatedly execute a routine	FOR FOR...DO(1)	Relay ladder Structured text	482
Terminate the repeated execution of a routine	BRK EXIT(1)	Relay ladder Structured text	485
Return to the FOR instruction	RET	Relay ladder	486

(1) Structured text only.

For (FOR)

The FOR instruction executes a routine repeatedly.

Operands:



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

Structured Text

FOR *count*:= *initial_value* TO *final_value* BY *increment* DO

<*statement*>;

Use the FOR...DO construct. See Structured Text Programming 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:

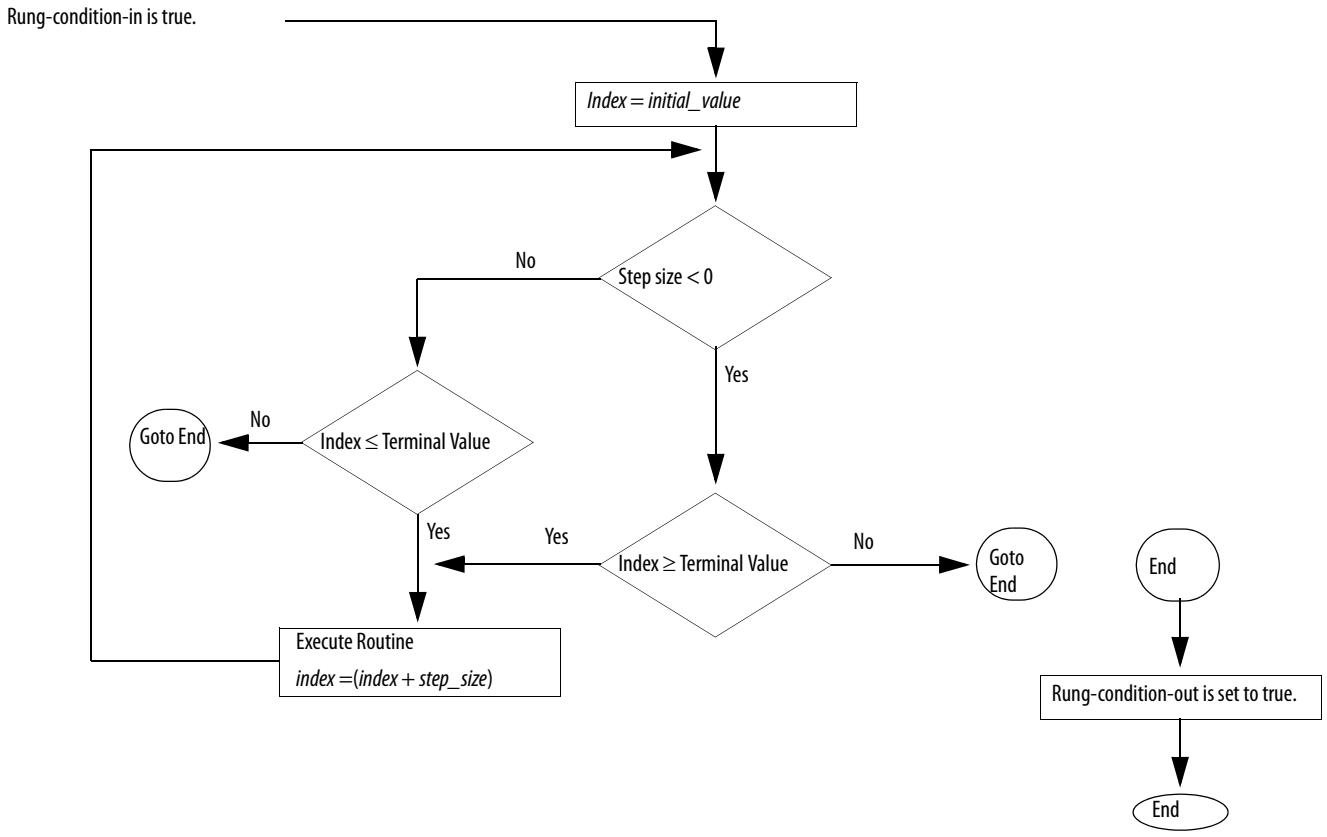
Not affected

Fault Conditions:

A major fault will occur if	Fault Type	Fault Code
Main routine contains a RET instruction	4	31

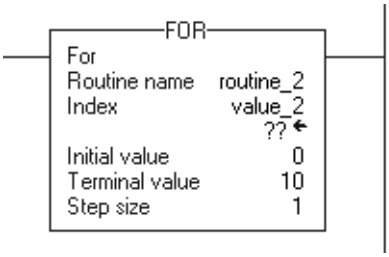
Execution:

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.



Postscan	The rung-condition-out is set to false.

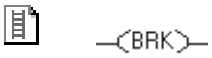
Example: When enabled, the FOR instruction repeatedly executes *routine_2* and increments *value_2* by 1 each time. When *value_2* is > 10 or a BRK instruction is enabled, the FOR instruction no longer executes *routine_2*.



Break (BRK)

The BRK instruction interrupts the execution of a routine that was called by a FOR instruction.

Operands:



Relay Ladder

None

EXIT;

Structured Text

Use the EXIT statement in a loop construct. See Appendix B 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:

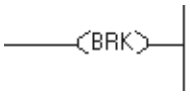
None

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.

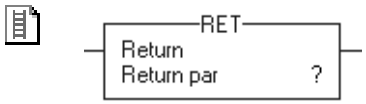


Return (RET)

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:

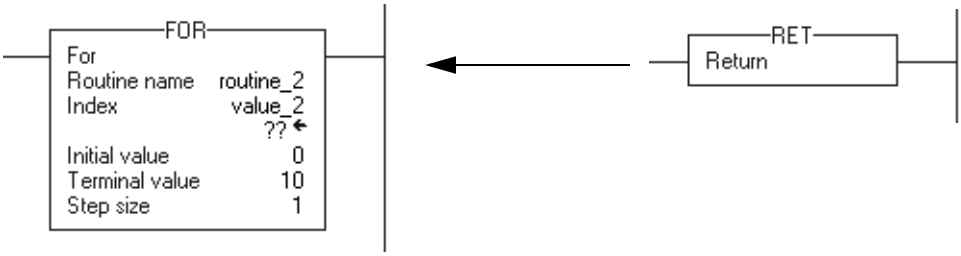
Condition:	Relay Ladder Action
Prescan	The rung-condition-out is set to false.
Rung-condition-in is false	The rung-condition-out is set to false.
Rung-condition-in is true	Returns the specified parameters to the calling routine. The rung-condition-out is set to true.
Postscan	The rung-condition-out is 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.

Calling Routine **Subroutine**



Notes:

Special Instructions
(FBC, DDT, DTR, PID)

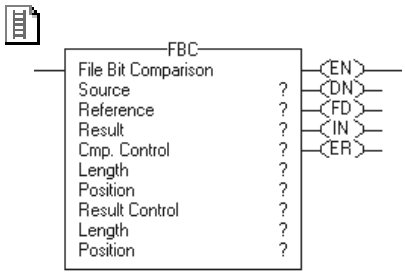
The special instructions perform application-specific operations.

If you want to	Use this instruction	Available in these languages	Page
Compare data against a known, good reference and record any mismatches	FBC	Relay ladder	490
Compare data against a known, good reference, record any mismatches, and update the reference to match the source	DDT	Relay ladder	497
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	504
Control a PID loop	PID	Relay ladder Structured text	507

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.

IMPORTANT You must test and confirm that the instruction doesn't change data that you don't want it to change.

The FBC instruction operates on contiguous memory. In some cases, the instruction searches or writes past the array into other members of the tag. This happens if a length is too big and the tag is a user-defined data type.

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.

Select 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.

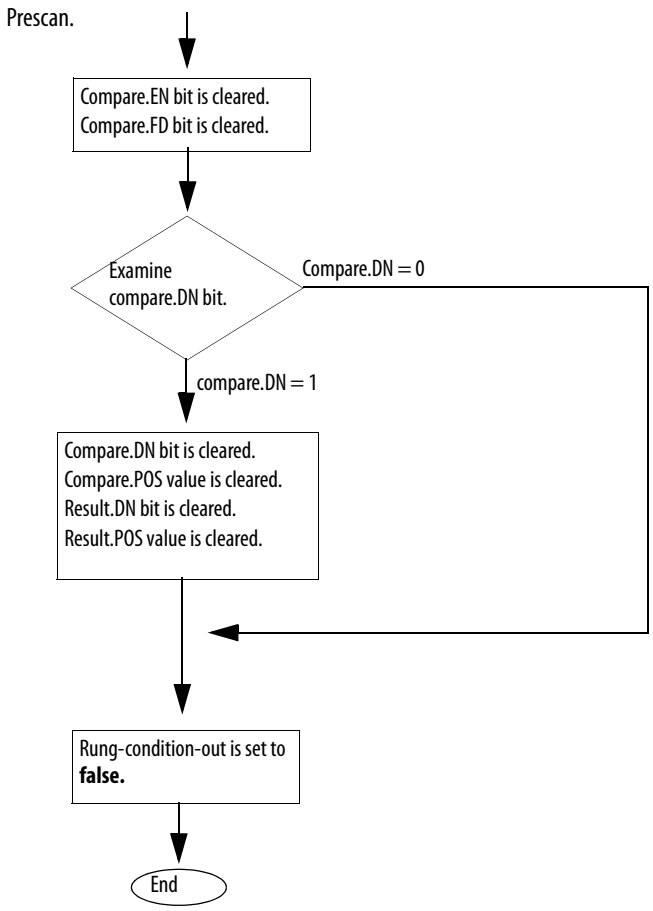
Arithmetic Status Flags: Not affected

Fault Conditions:

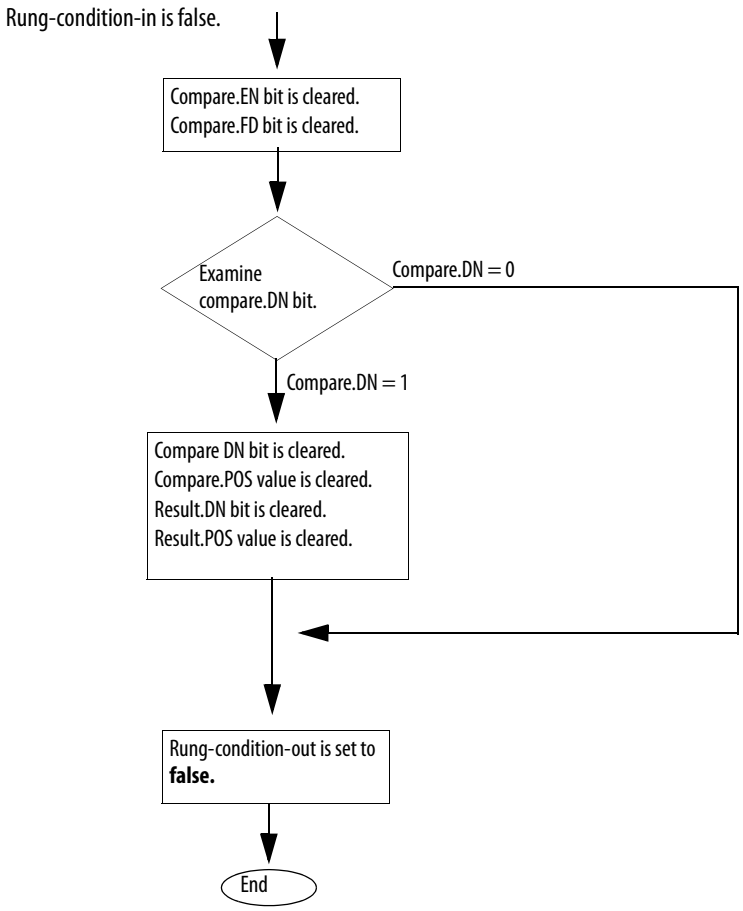
A major fault will occur if	Fault type	Fault code
Result.POS > Size of Result array	4	20

Execution:

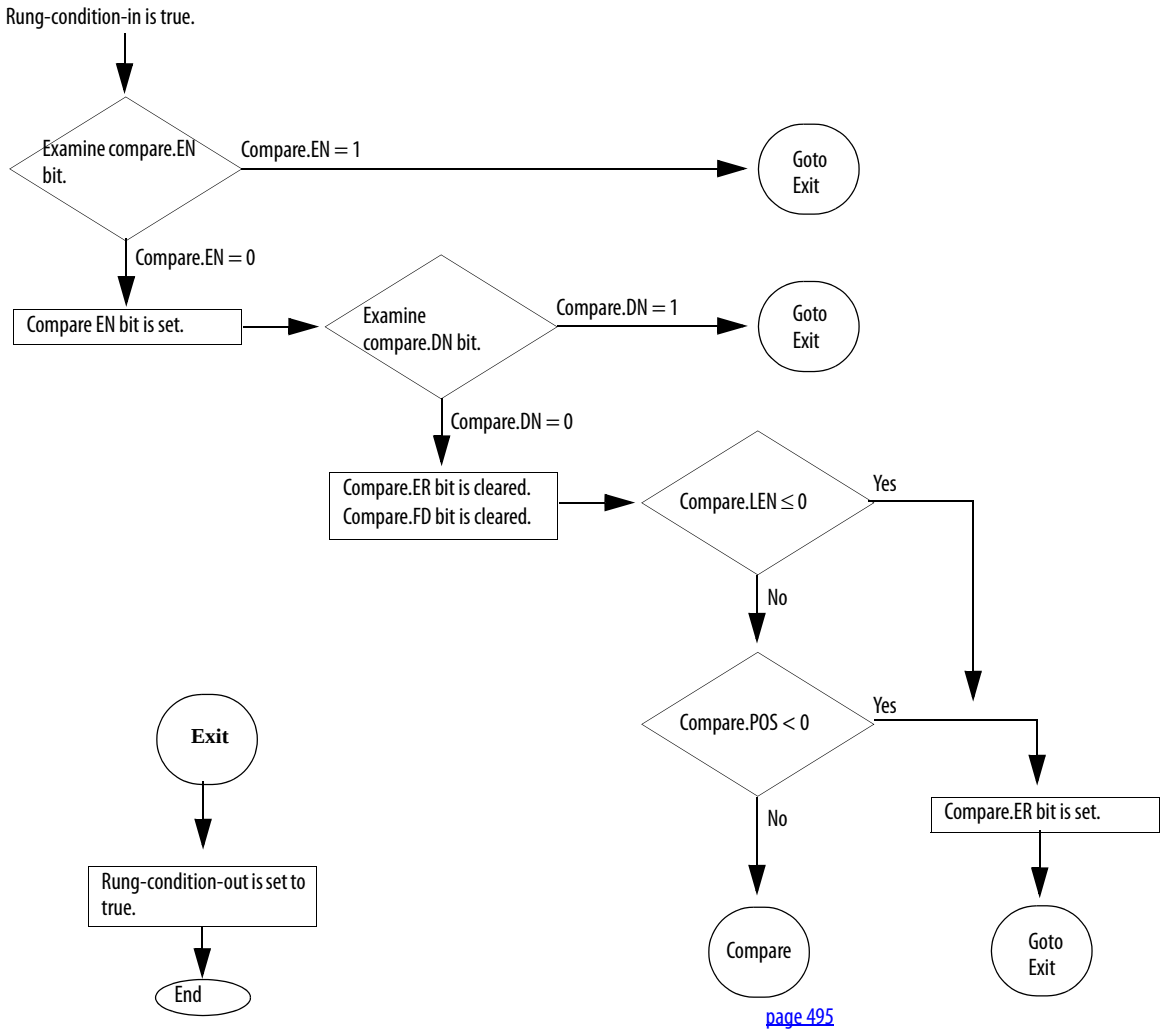
Condition	Relay Ladder Action



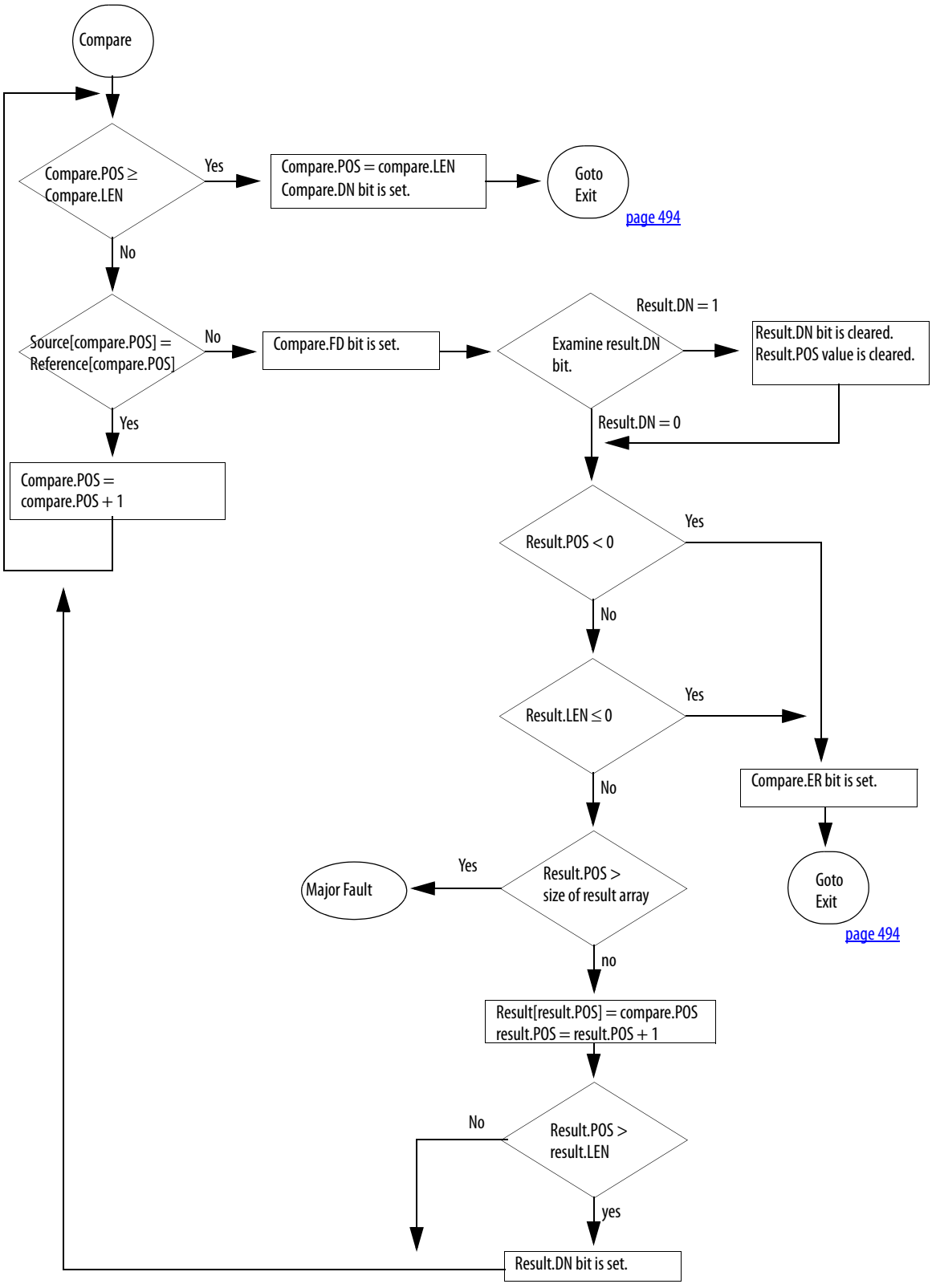
Condition	Relay Ladder Action



Condition	Relay Ladder Action

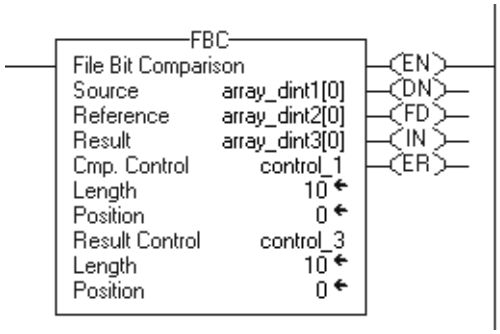


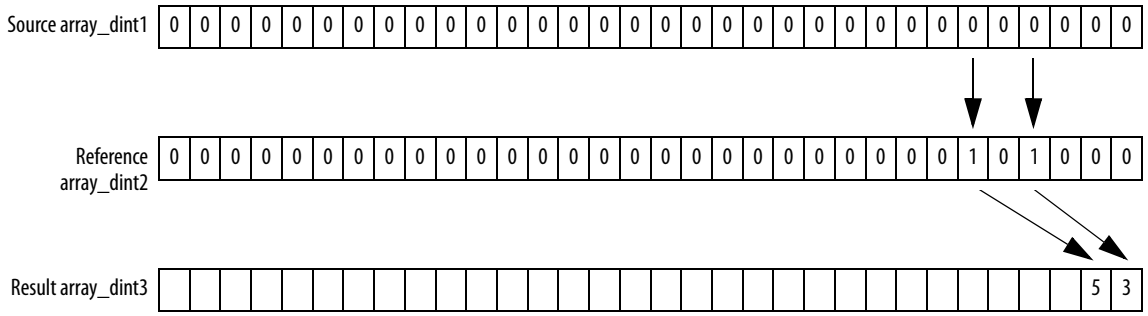
Condition	Relay Ladder Action



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*.

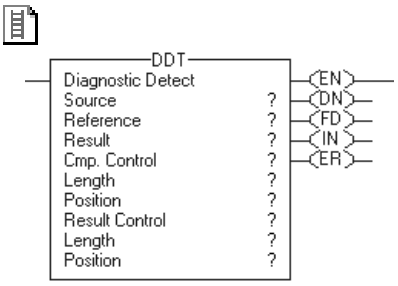




Diagnostic Detect (DDT)

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.

IMPORTANT You must test and confirm that the instruction doesn't change data that you don't want it to change.

The DDT instruction operates on contiguous memory. In some cases, the instruction searches or writes past the array into other members of the tag. This happens if a length is too big and the tag is a user-defined data type.

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.

Select 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.

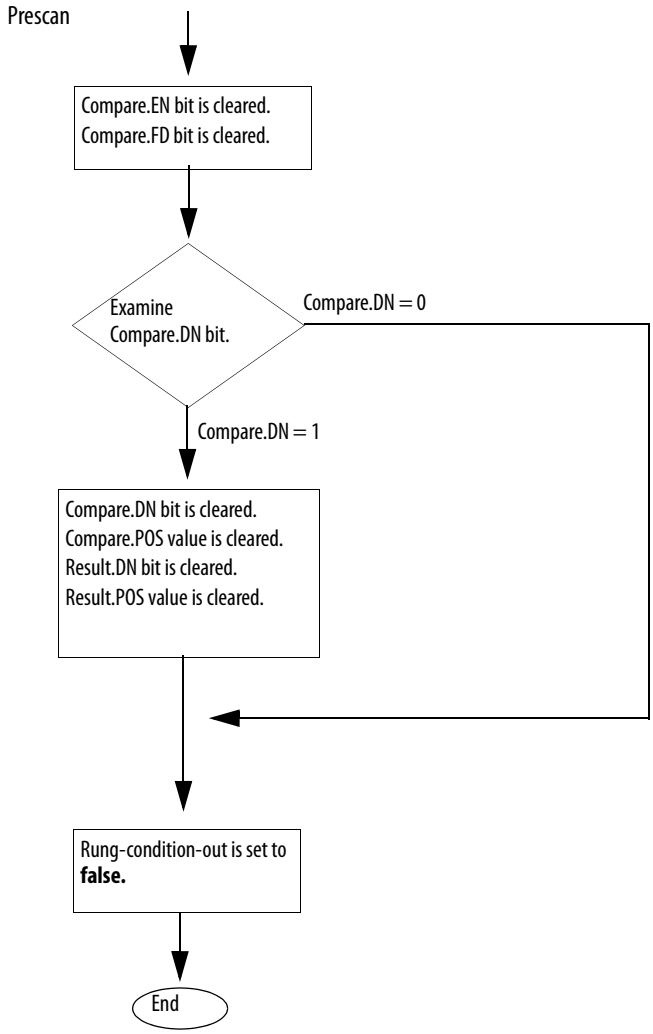
Arithmetic Status Flags: Not affected

Fault Conditions:

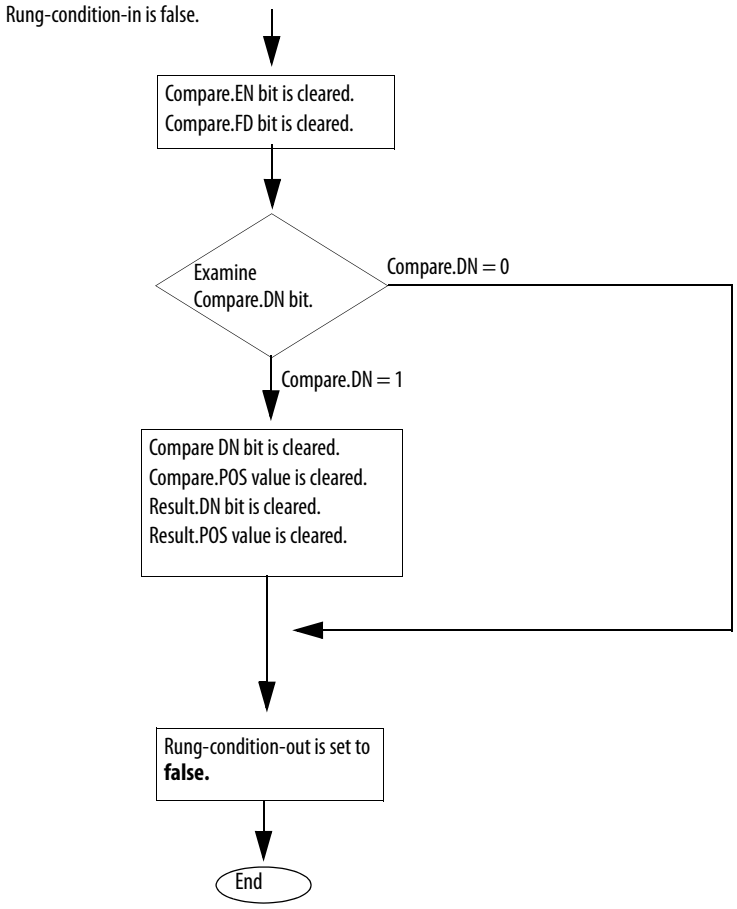
A major fault will occur if	Fault type	Fault code
Result.POS > size of Result array	4	20

Execution:

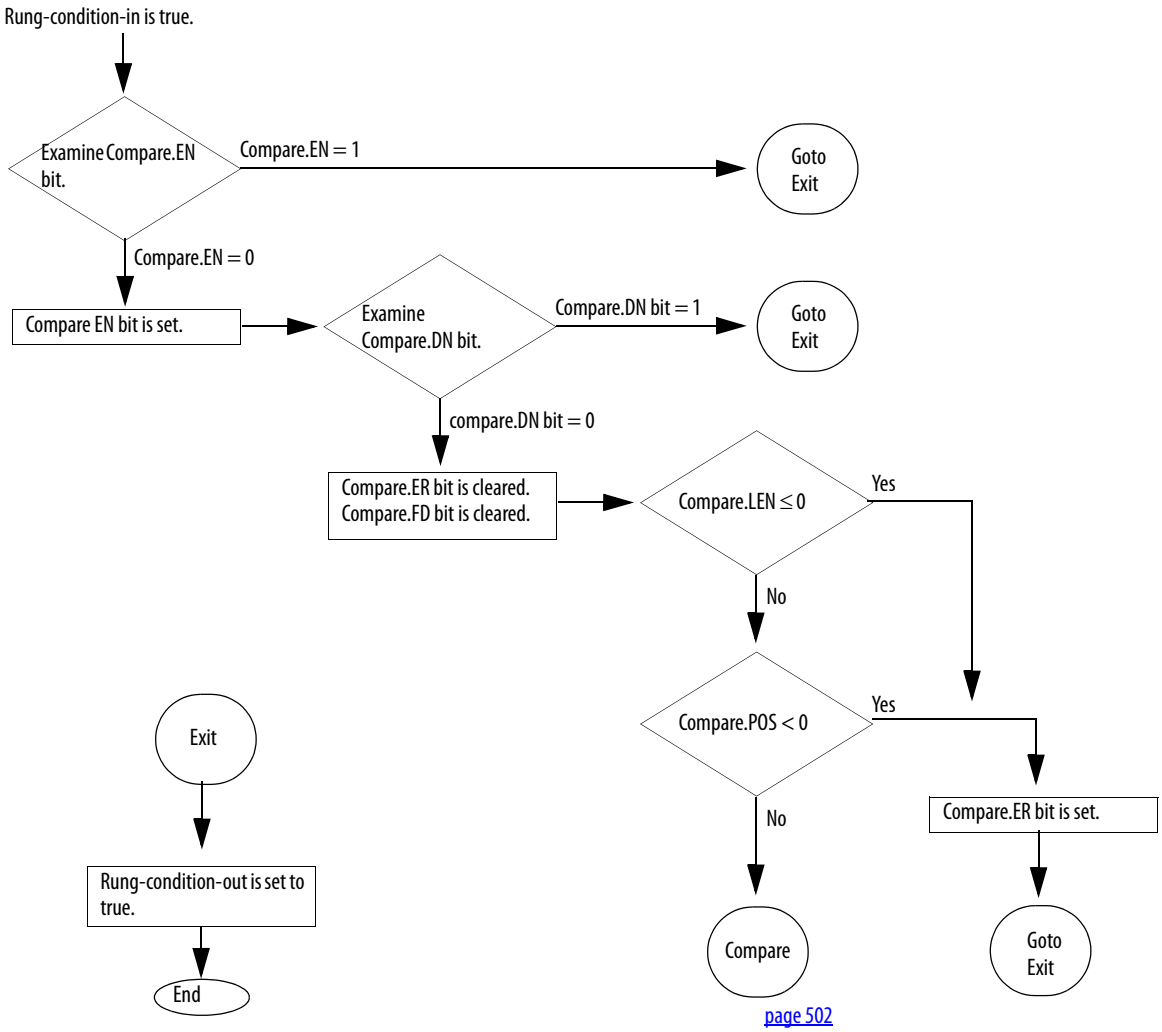
Condition:	Relay Ladder Action



Condition:	Relay Ladder Action

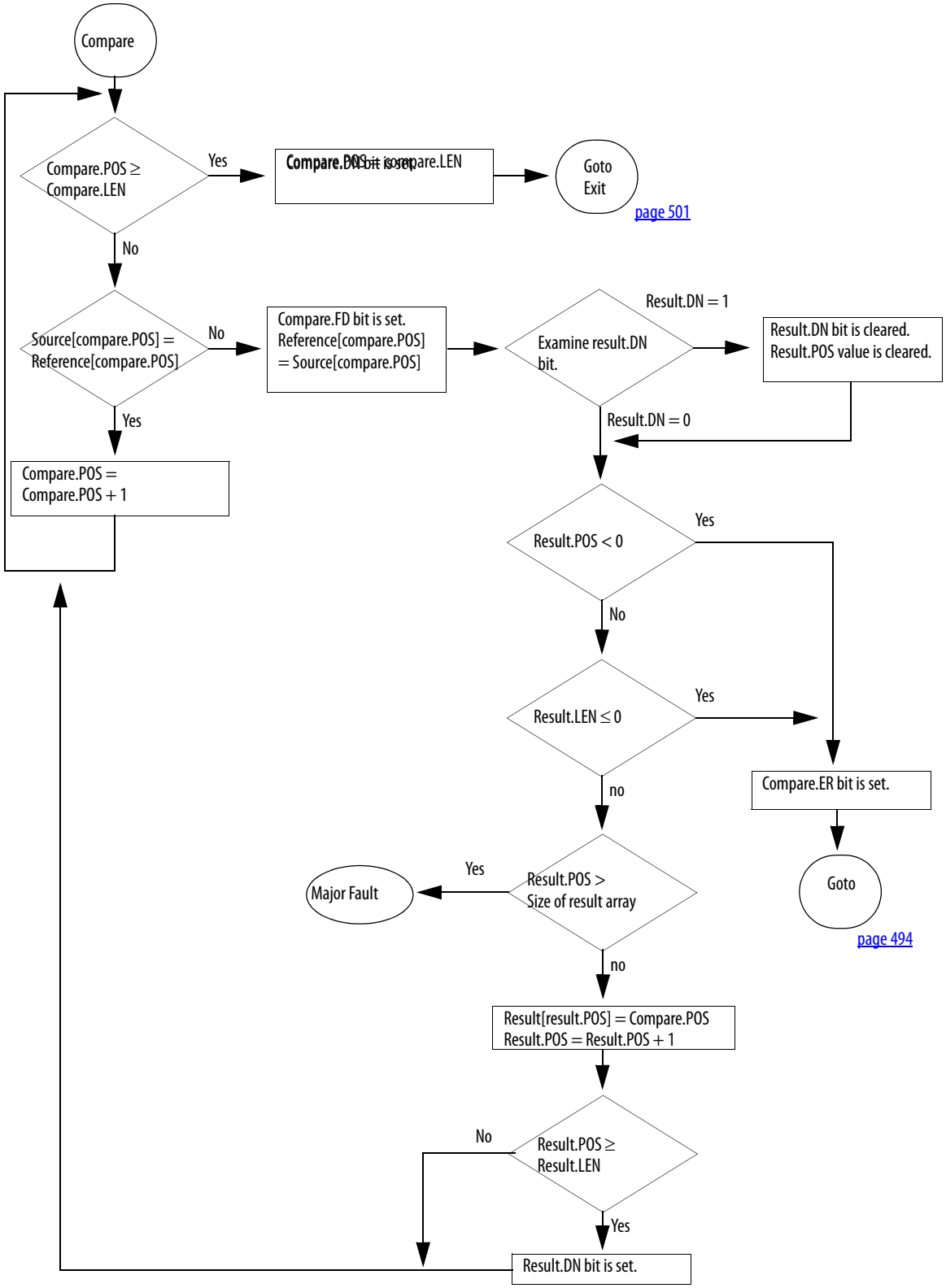


Condition:	Relay Ladder Action



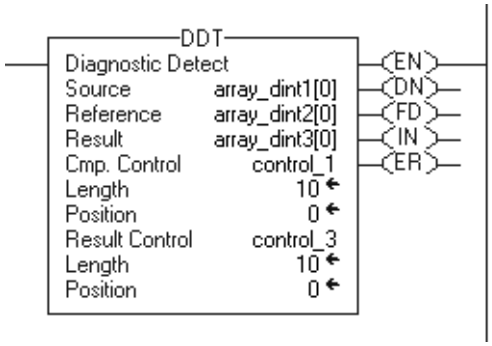
page 502

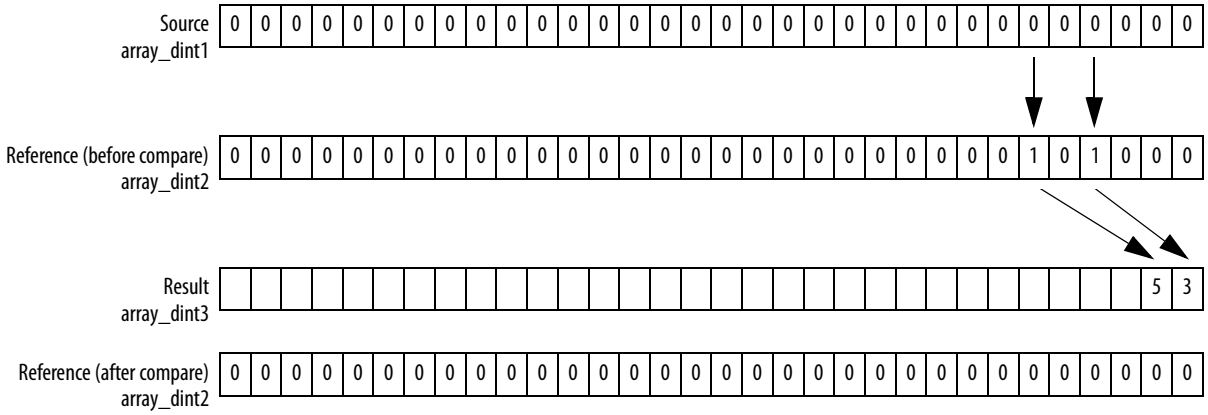
Condition:	Relay Ladder Action



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*.

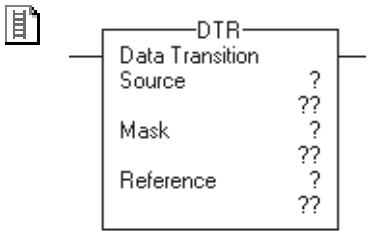




Data Transitional (DTR)

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.

Enter 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

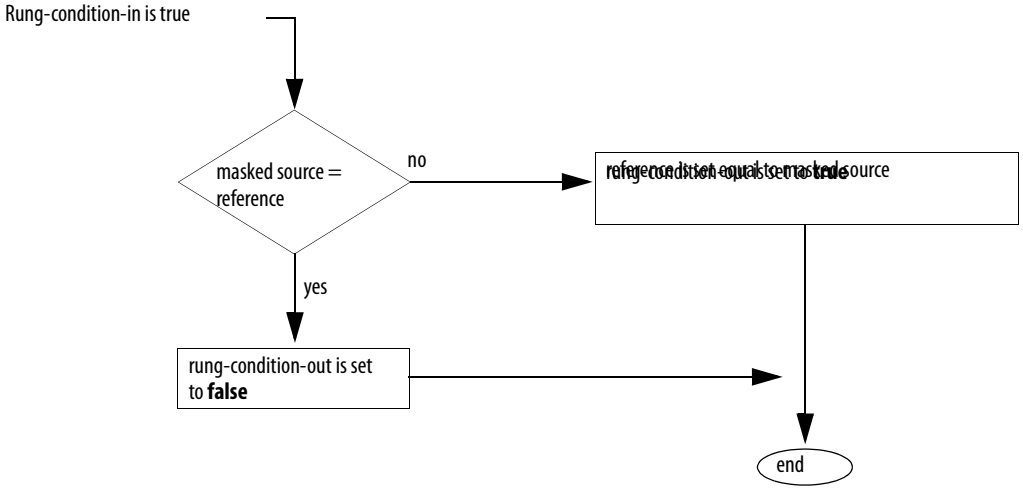
Arithmetic Status Flags:

Not affected

Fault Conditions: None

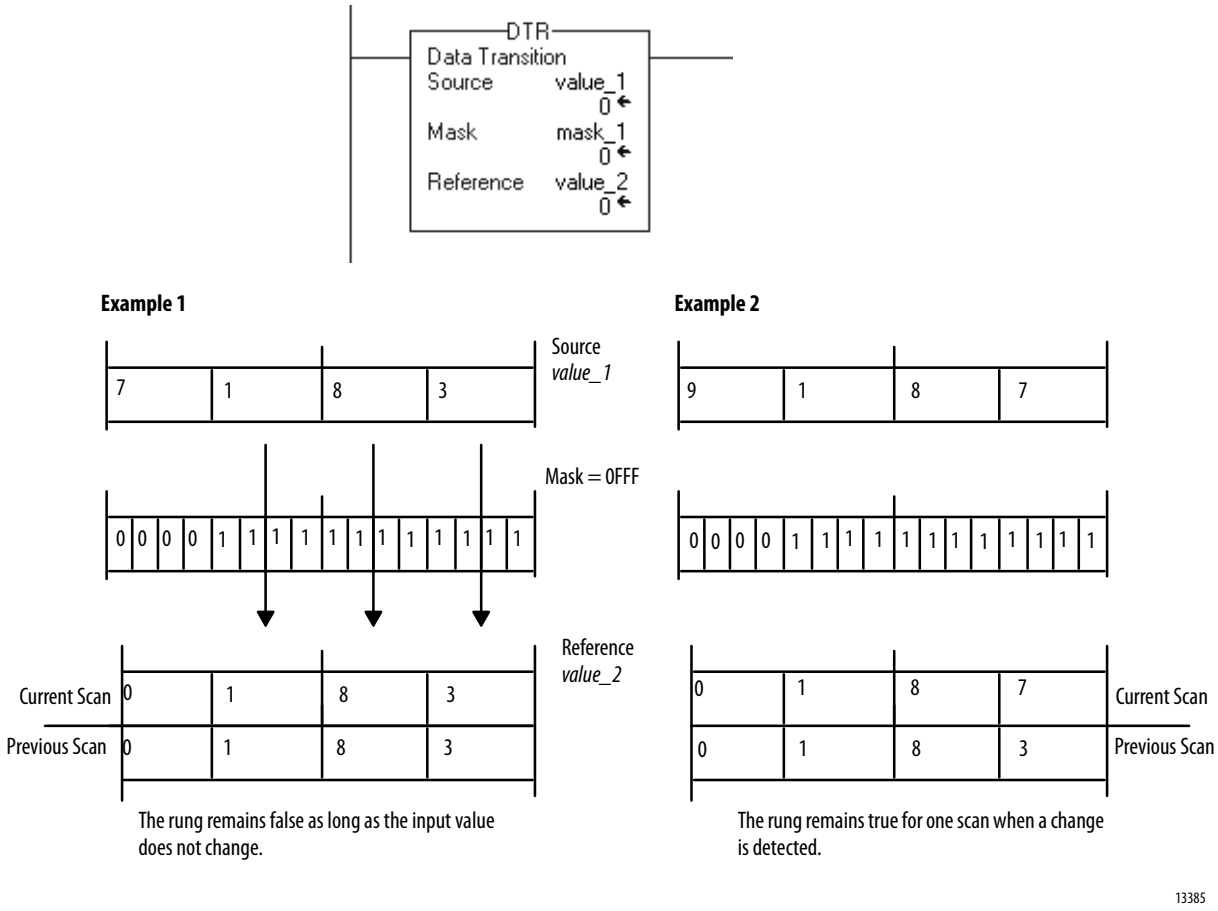
Execution:

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.



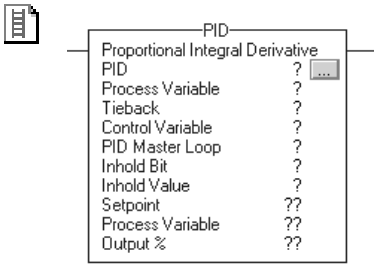
A 0 in the mask leaves the bit unchanged.

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	(Optional) Output of a hardware hand/auto station that 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 that goes to the final control device (valve, damper, and so forth) 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	(Optional) 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	(Optional) 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	(Optional) 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 percent 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	Setpoint	
.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 %	

Mnemonic:	Data Type	Description
.MAXS	REAL	Maximum engineering unit scaling value
.MINS	REAL	Minimum engineering unit scaling value
.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%)

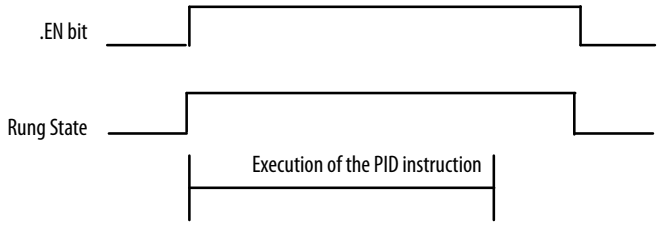
Mnemonic:	Data Type	Description	
.DATA	REAL[17]	The .DATA member stores:	
		Element	**Description**
		.DATA[0]	Integral accumulation
		.DATA[1]	Derivative smoothing temporary value
		.DATA[2]	Previous .PV value
		.DATA[3]	Previous .ERR value
		.DATA[4]	Previous valid .SP value
		.DATA[5]	Percent scaling 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)	
.EWD	BOOL	Error is within deadband (0=no; 1=yes)	
.DVNA	BOOL	Deviation is alarmed low (0=no; 1=yes)	

Mnemonic:	Data Type	Description
.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.

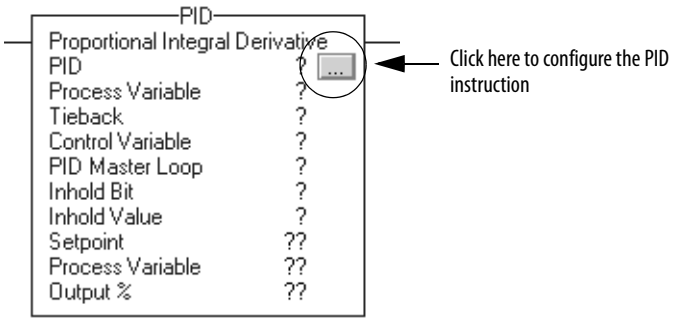
A minor fault will occur if	Fault type	Fault code
.UPD ≤ 0	4	35
Setpoint out of range	4	36

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.	N/A
Rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	N/A
EnableIn is set	N/A	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.

Configure 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.



Specify Tuning

Select the Tuning tab. Changes take affect 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.

Specify 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).(1)
CV low limit	Enter a low limit for the control variable (.MINO).(1)
Deadband value	Enter a deadband value (.DB).
No derivative smoothing	Enable or disable this selection (.NDF).
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).

(1) When using the ladder-based PID instruction, if you set MAXO = MINO, the PID instruction will reset these values to default. MAXO = 100.0 and MINO = 0.0

Specify 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).

Specify Scaling

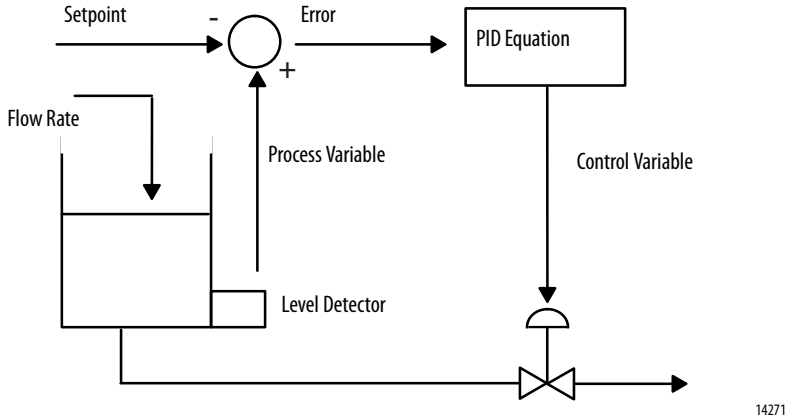
Select the Scaling tab. You must click OK or Apply for any changes to take effect.

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)(1)
PV engineering units minimum	Enter the minimum engineering units corresponding to .MINI (.MINS)(1)
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).

(1) When using the ladder-based PID instruction, if you set MAXO = MINO, the PID instruction will reset these values to default. MAXO = 100.0 and MINO = 0.0

Use PID Instructions

PID closed-loop control holds a process variable at a desired set point. The illustration shows an example of a flow-rate/fluid level.



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 affect only their specific proportional, integral, or derivative terms respectively. When using dependent gains, the proportional gain is replaced with a controller gain that

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)	$$CV = K_C\left[E + \frac{1}{T_i}\int_0^t Edt + T_d\frac{dE}{dt}\right] + BIAS$$
	Process variable (PV)	E = SP - PV $$CV = K_C\left[E + \frac{1}{T_i}\int_0^t Edt - T_d\frac{dPV}{dt}\right] + BIAS$$ E = PV - SP $$CV = K_C\left[E + \frac{1}{T_i}\int_0^t Edt + T_d\frac{dPV}{dt}\right] + BIAS$$
Independent gains	Error (E)	$$CV = K_PE + K_i\int_0^t Edt + K_d\frac{dE}{dt} + BIAS$$
	Process variable (PV)	E = SP - PV $$CV = K_PE + K_i\int_0^t Edt - K_d\frac{dPV}{dt} + BIAS$$ E = PV - SP $$CV = K_PE + K_i\int_0^t Edt + K_d\frac{dPV}{dt} + BIAS$$

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 that is bypassing the output from the controller. **Important:** 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 (that is $K_i = 0$). 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.

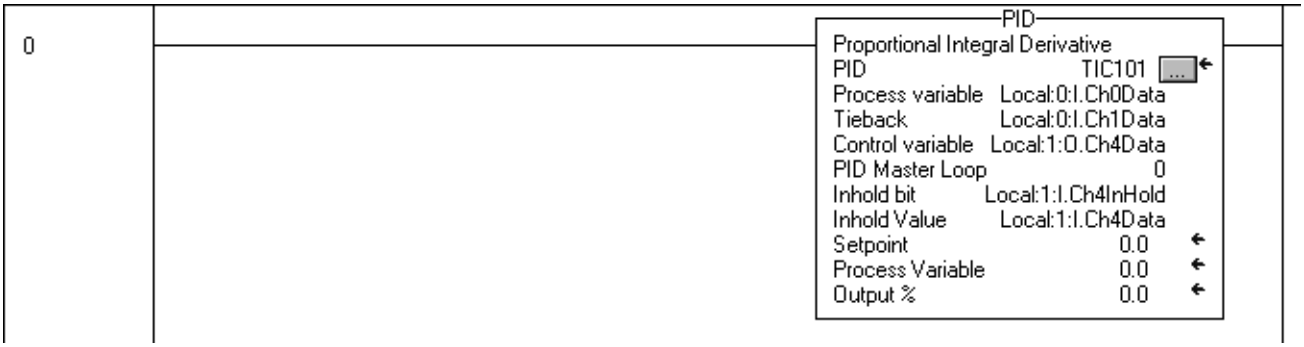
PID Instruction Timing

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 ms. Only rare cases, such as tension control on an unwinder spool, require loop updates as fast as every 10 ms 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



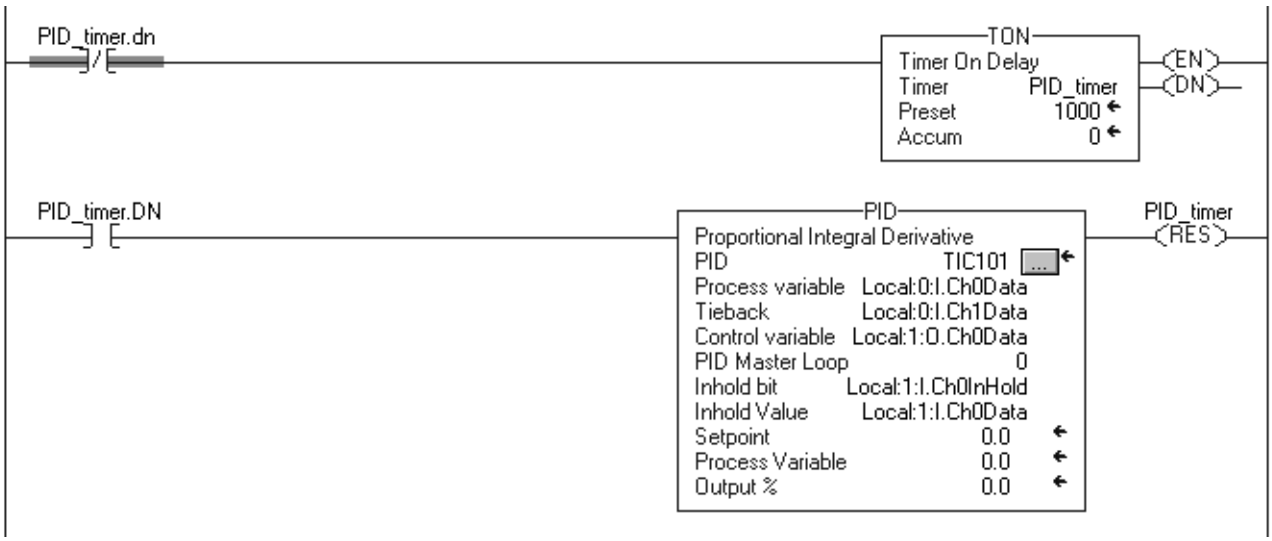
Structured Text

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...10 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 ms periodic task, use a loop update time of 250 ms (.UPD = .25), and configure the analog input module to produce data at least about every 25...50 ms.

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.

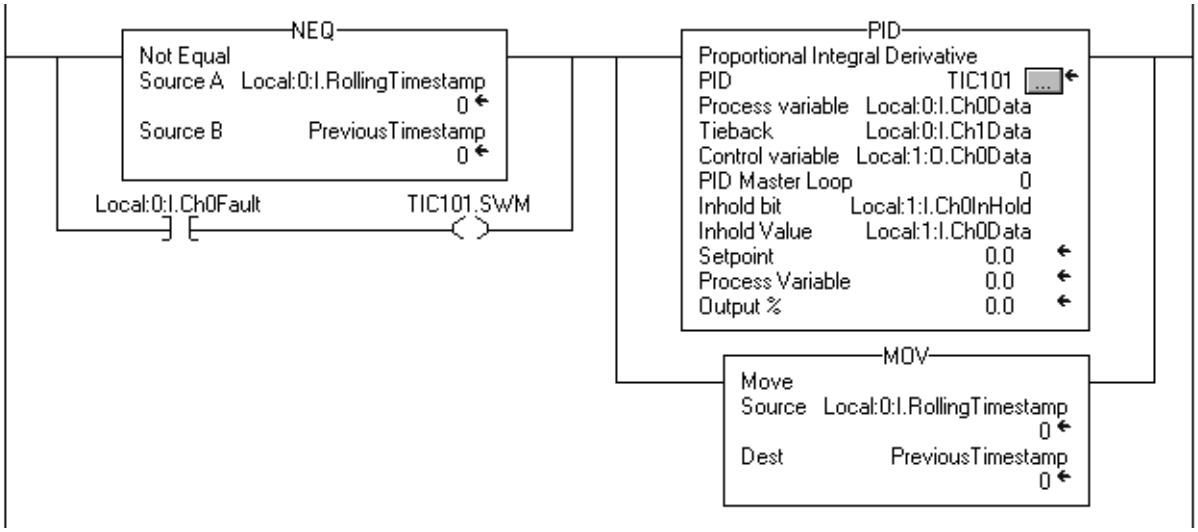
The most accurate way to execute a PID instruction is to use the real time sampling (RTS) feature of the 1756 analog input modules. The analog input module samples its inputs at the real time sampling rate you configure when you set up the module. When the module's real time sample period expires, it 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...32,767 ms. 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 ms, you could put the PID logic in a periodic task that runs every 100 ms 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 ms.

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 the 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.

Instructions for setting a bumpless restart.

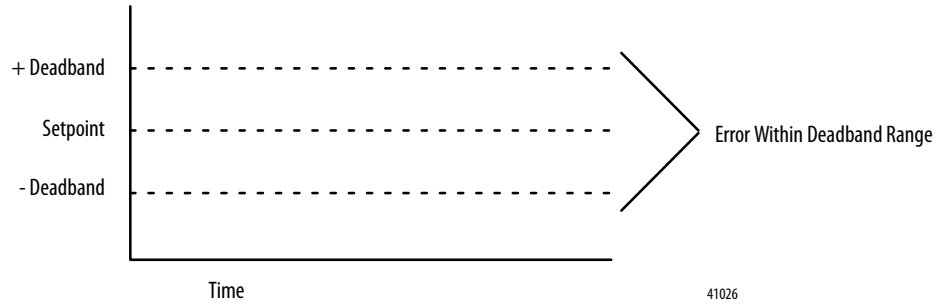
Do this	Details
Configure the 1756 analog output module's channel that receives the control variable from the PID instruction	Select the "hold for initialization" checkbox on the properties page for the specific channel of the module. This tells the analog output module that when the controller returns 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.
Enter the Inhold bit tag and Inhold Value tag in the PID instruction	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. 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. When he 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.

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.

Set 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 zero when the error is within the deadband.

Use 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.

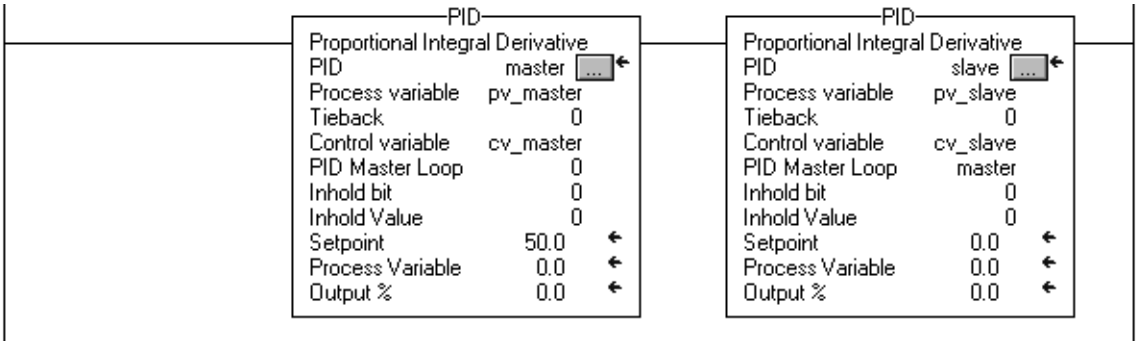
The feedforward value represents a disturbance fed into the PID instruction before the disturbance has a chance to change the process variable. Feedforward is often used to control processes with a transportation lag. For example, a feedforward value representing 'cold water poured into a warm mix' could 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.

Cascade 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);

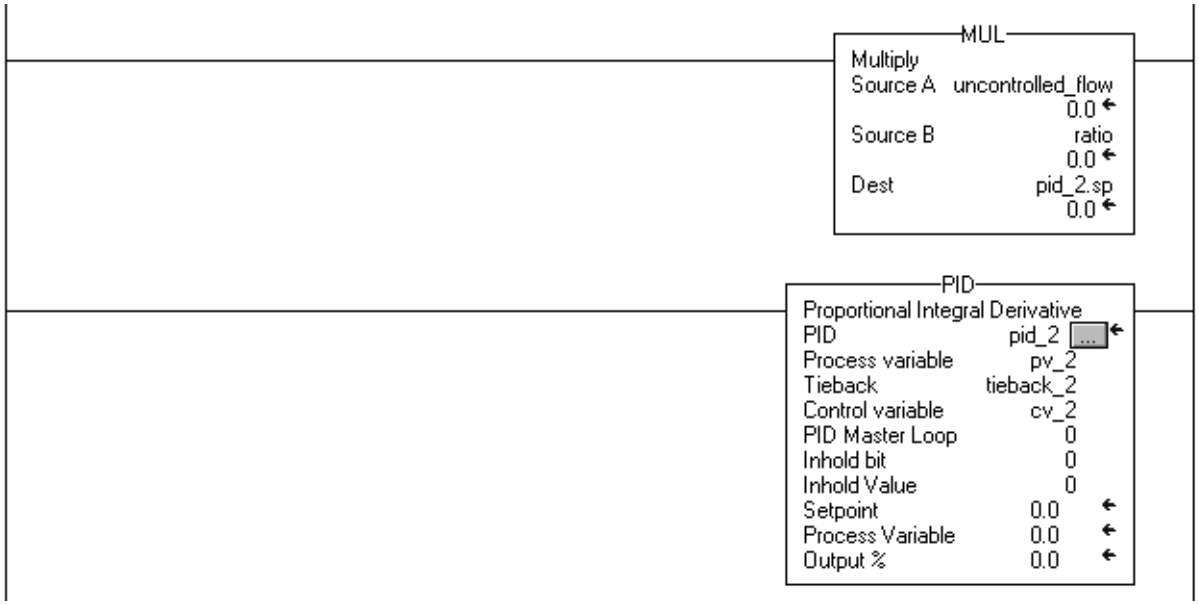
PID (slave,pv_slave,0,cv_slave,master,0,0);

Control 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

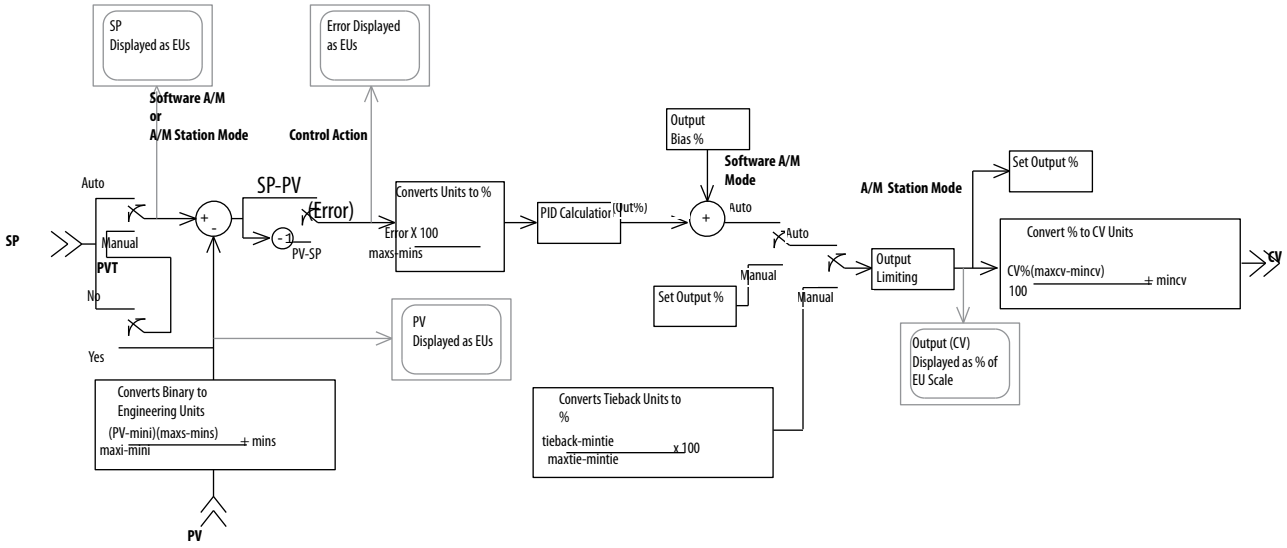
```
pid_2.sp := uncontrolled_flow * ratio

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

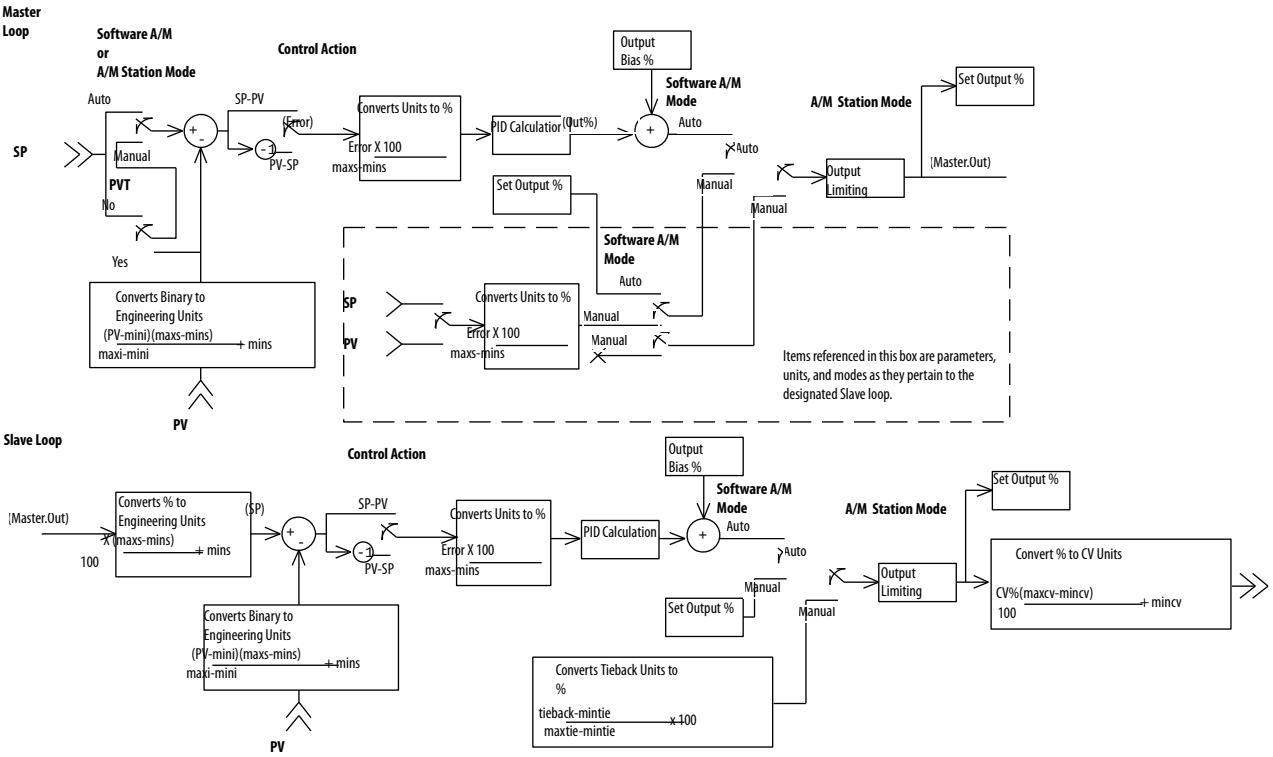
PID Theory

The following figures show the process flow for PID instructions.

PID Process



PID Process with Master/Slave Loops



Trigonometric Instructions
(SIN, COS, TAN, ASN, ASIN, ACS, ACOS, ATN, ATAN)

The trigonometric instructions evaluate arithmetic operations by using trigonometric operations.

If you want to	Use this instruction	Available in these languages	Page
Take the sine of a value	SIN	Relay ladder Structured text Function block	528
Take the cosine of a value	COS	Relay ladder Structured text Function block	531
Take the tangent of a value	TAN	Relay ladder Structured text Function block	534
Take the arc sine of a value	ASN ASIN(1)	Relay ladder Structured text Function block	537
Take the arc cosine of a value	ACS ACOS(1)	Relay ladder Structured text Function block	540
Take the arc tangent of a value	ATN ATAN(1)	Relay ladder Structured text Function block	543

(1) Structured text only.

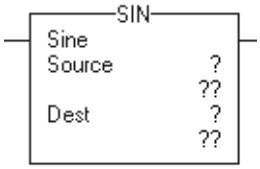
You can mix data types, but loss of accuracy and rounding error might occur and the instruction takes more time to execute. Check the overflow status bit (S:V) to see whether the result was truncated.

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

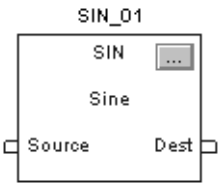
Operand	Type	Format	Description
Source	SINT INT DINT REAL	Immediate Tag	Find the sine of this value
Destination	SINT INT DINT REAL	Tag	Tag to store the result

dest := SIN(source);

Structured Text

Use SIN as a function. This function computes the sine of *source* and stores the result in *dest*.

See Structured Text Programming 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 ($-2\pi x2^{15}$) and less than or equal to 205887.4 ($2\pi x2^{15}$). 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

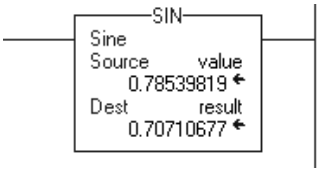
Condition	Action
Prescan	The rung-condition-out is set to false.
Rung-condition-in is false	The rung-condition-out is set to false.
Rung-condition-in is true	The controller calculates the sine of the Source and places the result in the Destination. The rung-condition-out is set to true.
Postscan	The rung-condition-out is set to false.

Function Block

Condition	Action
Prescan	No action taken.
Instruction first scan	No action taken.
Iinstruction 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);

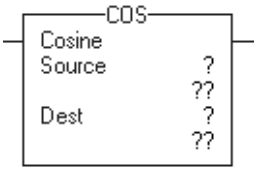
Function Block

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

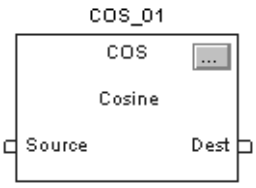
Structured Text

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

Use COS as a function. This function computes the cosine of *source* and stores the result in *dest*.

See Structured Text Programming 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 ($-2\pi x2^{15}$) and less than or equal to 205887.4 ($2\pi x2^{15}$). 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

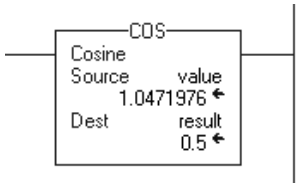
Condition	Action
Prescan	The rung-condition-out is set to false.
Rung-condition-in is false	The rung-condition-out is set to false.
Rung-condition-in is true	The controller calculates the cosine of the Source and places the result in the Destination. The rung-condition-out is set to true.
Postscan	The 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*.

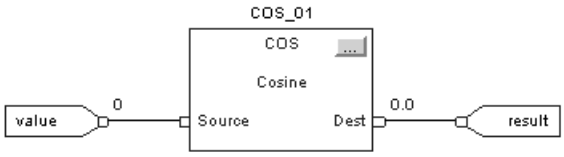
Relay Ladder



Structured Text

```
result := COS(value);
```

Function Block

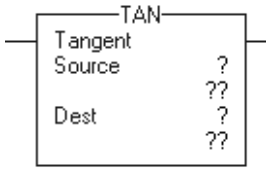


Tangent (TAN)

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

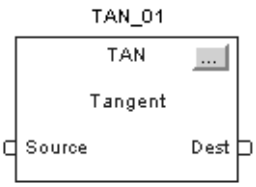
Structured Text

dest := TAN(source);

Use TAN as a function. This function computes the tangent of *source* and stores the result in *dest*.

See Structured Text Programming 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($-2\pi x2^{14}$) and less than or equal to 102943.7 ($2\pi x2^{14}$).

Arithmetic Status Flags: Arithmetic status flags are affected.

Fault Conditions: None

Execution:

Relay Ladder

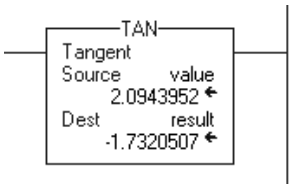
Condition	Action
Prescan	The rung-condition-out is set to false.
Rung-condition-in is false	The rung-condition-out is set to false.
Rung-condition-in is true	The controller calculates the tangent of the Source and places the result in the Destination. The rung-condition-out is set to true.
Postscan	The 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*.

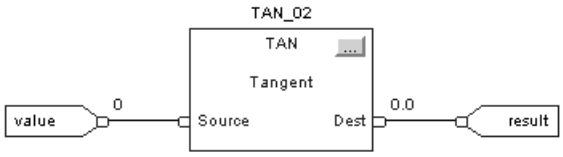
Relay Ladder



Structured Text

result := TAN(value);

Function Block

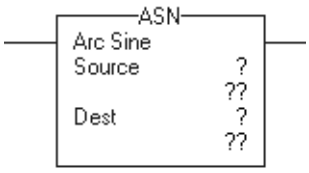


Arc Sine (ASN)

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

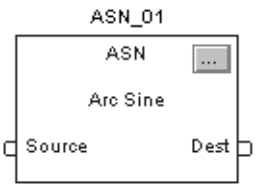
Structured Text

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

Use ASIN as a function. This function computes the arc sine of *source* and stores the result in *dest*.

See Structured Text Programming 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

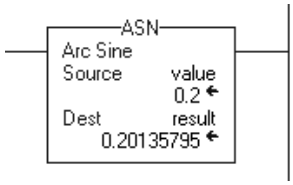
Condition	Action
Prescan	The rung-condition-out is set to false.
Rung-condition-in is false	The rung-condition-out is set to false.
Rung-condition-in is true	The controller calculates the arc sine of the Source and places the result in the Destination. The rung-condition-out is set to true.
Postscan	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*.

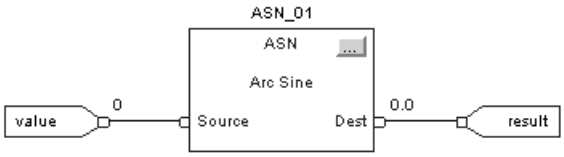
Relay Ladder



Structured Text

result := ASIN(value);

Function Block

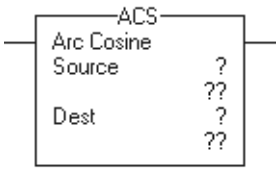


Arc Cosine (ACS)

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

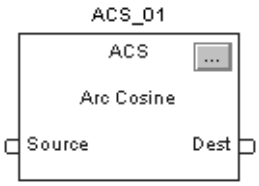
Structured Text

dest := ACOS(source);

Use ACOS as a function. This function computes the arc cosine of *source* and stores the result in *dest*.

See Structured Text Programming 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 π = 3.141593).

Arithmetic Status Flags: Arithmetic status flags are affected.

Fault Conditions: None

Execution:

Relay Ladder

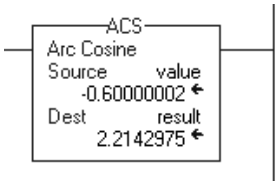
Condition	Action
Prescan	The rung-condition-out is set to false.
Rung-condition-in is false	The rung-condition-out is set to false.
Rung-condition-in is true	The controller calculates the arc cosine of the Source and places the result in the Destination. The rung-condition-out is set to true.
Postscan	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*.

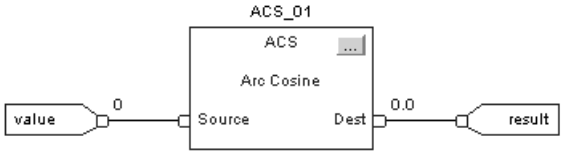
Relay Ladder



Structured Text

result := ACOS(value);

Function Block

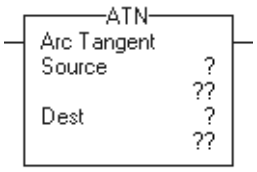


Arc Tangent (ATN)

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

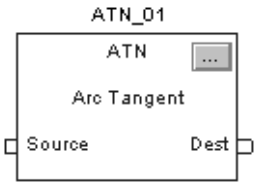
Structured Text

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

Use ATAN as a function. This function computes the arc tangent of *source* and stores the result in *dest*.

See Structured Text Programming 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 -π/2 and less than or equal to π/2 (where π = 3.141593).

Arithmetic Status Flags: Arithmetic status flags are affected.

Fault Conditions: None

Execution:

Relay Ladder

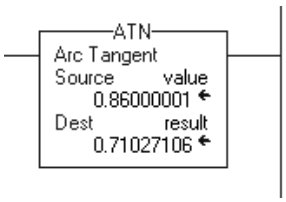
Condition	Action
Prescan	The rung-condition-out is set to false.
Rrung-condition-in is false	The rung-condition-out is set to false.
Rung-condition-in is true	The controller calculates the arc tangent of the Source and places the result in the Destination. The rung-condition-out is set to true.
Postscan	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*.

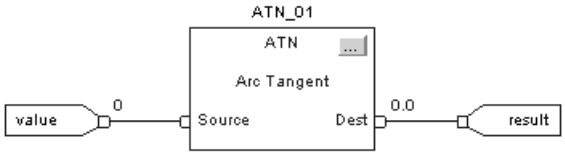
Relay Ladder



Structured Text

```
result := ATAN(value);
```

Function Block



Notes:

Advanced Math Instructions
(LN, LOG, XPY)

The advanced math instructions include these instructions.

If you want to	Use this instruction	Available in these languages	Page
Take the natural log of a value	LN	Relay ladder Structured text Function block	548
Take the log base 10 of a value	LOG	Relay ladder Structured text Function block	551
Raise a value to the power of another value	XPY	Relay ladder Structured text(1) Function block	554

(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.

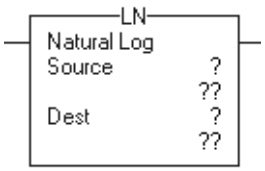
For relay ladder instructions, **bold** data types indicate optimal data types. An instruction executes faster and requires less memory if all the operands of the instruction use the same optimal data type, typically DINT or REAL.

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

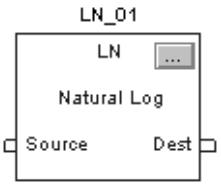
Structured Text

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

Use LN as a function. This function computes the natural log of *source* and stores the result in *dest*.

See Structured Text Programming 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

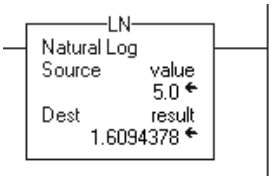
Condition	Action
Prescan	The rung-condition-out is set to false.
Rung-condition-in is false	The rung-condition-out is set to false.
Rung-condition-in is true	The controller calculates the natural log of the Source and places the result in the Destination. The rung-condition-out is set to true.
Postscan	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*.

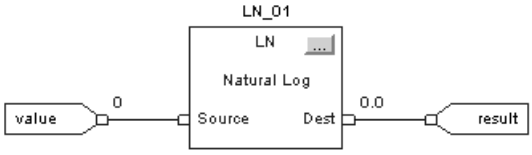
Relay Ladder Example



Structured Text

result := LN(value);

Function Block

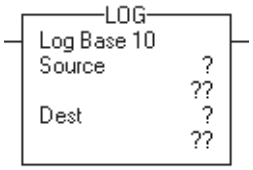


Log Base 10 (LOG)

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

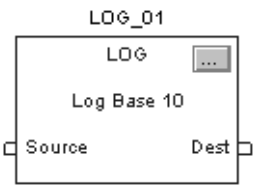
Structured Text

dest := LOG(source);

Use LOG as a function. This function computes the log of *source* and stores the result in *dest*.

See Structured Text Programming 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

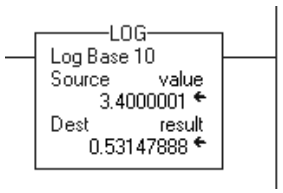
Condition	Action
Prescan	The rung-condition-out is set to false.
Rung-condition-in is false	The rung-condition-out is set to false.
Rung-condition-in is true	The controller calculates the log of the Source and places the result in the Destination. The rung-condition-out is set to true.
Postscan	The rung-condition-out 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*.

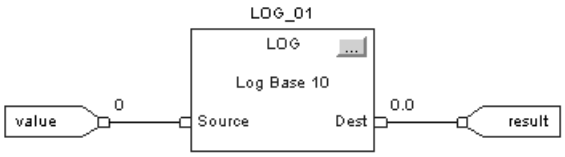
Relay Ladder



Structured Text

result := LOG(value);

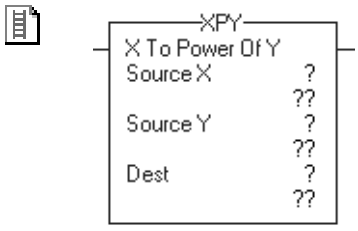
Function Block



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

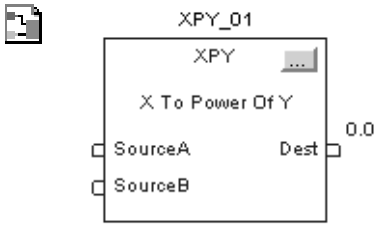
Operand	Type	Format	Description
Source X	SINT INT DINT **REAL**	Immediate Tag	Base value
Source Y	SINT INT DINT **REAL**	Immediate Tag	Exponent
Destination	SINT INT DINT **REAL**	Tag	Tag to store the result

Structured Text

dest := sourceX ** sourceY;

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 Structured Text Programming 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

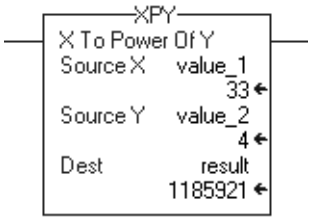
Condition	Action
Prescan	The rung-condition-out is set to false.
Rung-condition-in is false	The rung-condition-out is set to false.
Rung-condition-in is true	The controller takes Source X to the power of Source Y and places the result in the Destination. The rung-condition-out is set to true.
Postscan	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*.

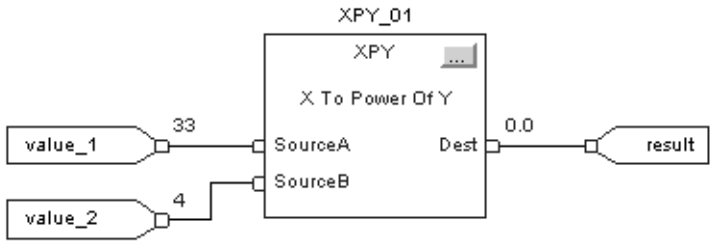
Relay Ladder



Structured Text

```
result := (value_1 ** value_2);
```

Function Block



Notes:

Math Conversion Instructions

(DEG, RAD, TOD, FRD, TRN, TRUNC)

The math conversion instructions convert values.

If you want to	Use this instruction	Available in these languages	Page
Convert radians to degrees	DEG	Relay ladder Structured text Function block	560
Convert degrees to radians	RAD	Relay ladder Structured text Function block	563
Convert an integer value to a BCD value	TOD	Relay ladder Function block	566
Convert a BCD value to an integer value	FRD	Relay ladder Function block	569
Remove the fractional part of a value	TRN TRUNC(1)	Relay ladder Structured text Function block	571

(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.

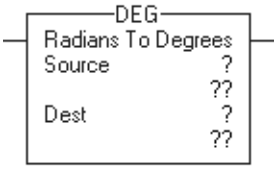
For relay ladder instructions, **bold** data types indicate optimal data types. An instruction executes faster and requires less memory if all the operands of the instruction use the same optimal data type, typically DINT or REAL.

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

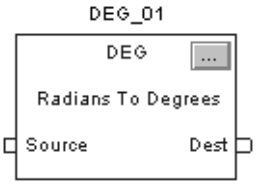
Structured Text

dest := DEG(source);

Use DEG as a function. This function converts *source* to degrees and stores the result in *dest*.

See Structured Text Programming 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:
Source*180/π (where π = 3.141593)

Arithmetic Status Flags: Arithmetic status flags are affected.

Fault Conditions: None

Execution:

Relay Ladder

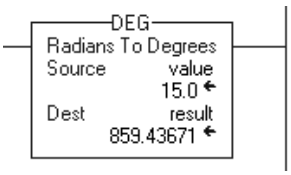
Condition	Action
Prescan	The rung-condition-out is set to false.
Rung-condition-in is false	The rung-condition-out is set to false.
Rung-condition-in is true	The controller converts the Source to degrees and places the result in the Destination. The rung-condition-out is set to true.
Postscan	The rung-condition-out 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*.

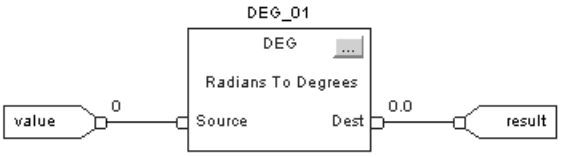
Relay Ladder



Structured Text

result := DEG(value);

Function Block

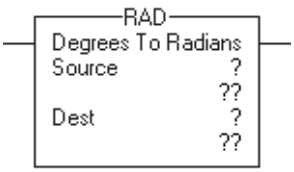


Radians (RAD)

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

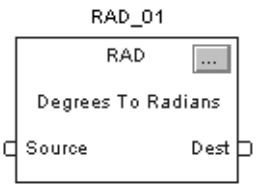
Structured Text

dest := RAD(source);

Use RAD as a function. This function converts *source* to radians and stores the result in *dest*.

See Structured Text Programming 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:
Source*π/180 (where π = 3.141593)

Arithmetic Status Flags: Arithmetic status flags are affected.

Fault Conditions: None

Execution:

Relay Ladder

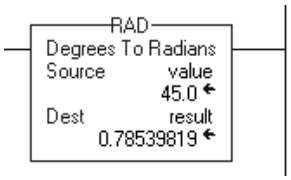
Condition	Action
Prescan	The rung-condition-out is set to false.
Rung-condition-in is false	The rung-condition-out is set to false.
Rung-condition-in is true	The controller converts the Source to radians and places the result in the Destination. The rung-condition-out is set to true.
Postscan	The rung-condition-out 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*.

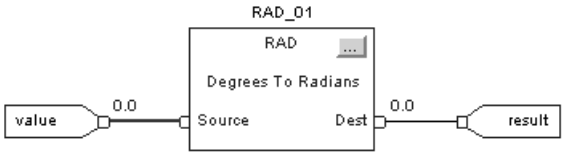
Relay Ladder



Structured Text

```
result := RAD(value);
```

Function Block

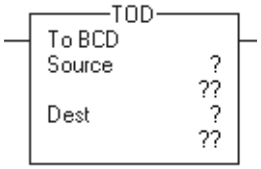


Convert to BCD (TOD)

The TOD instruction converts a decimal value (0 ≤ Source ≤ 99,999,999) to a BCD value and stores the result in the Destination.

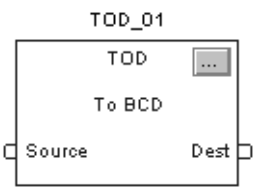
Operands:

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.

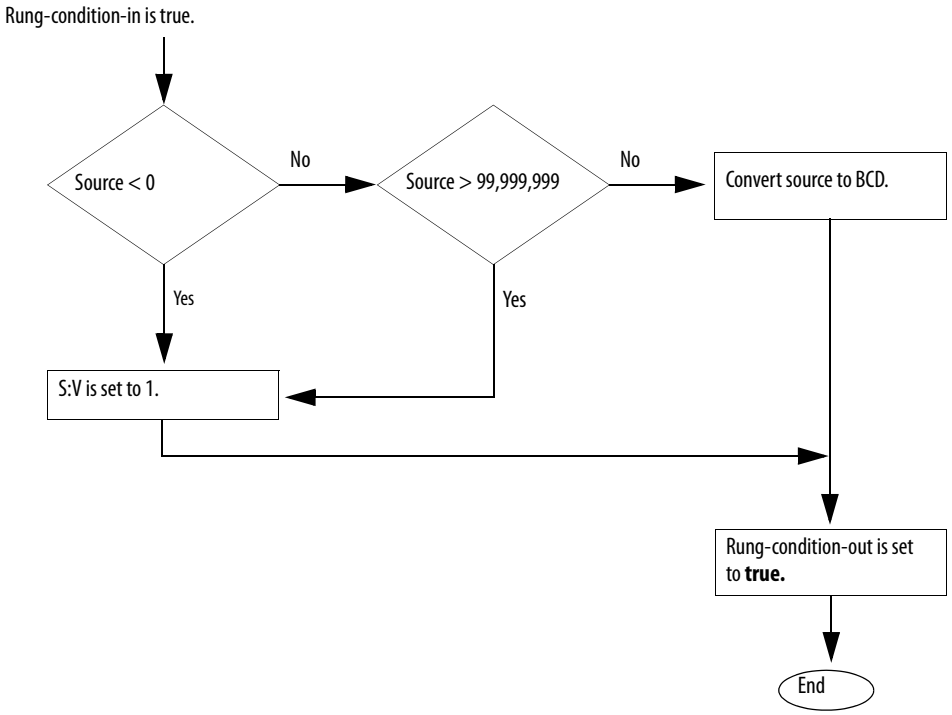
Fault Conditions:

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.



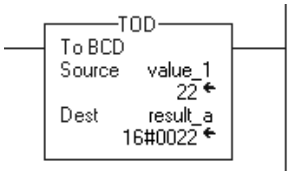
Rung-condition-in is true	The controller converts the Source to BCD and places the result in the Destination. The rung-condition-out is set to true.
Postscan	The rung-condition-out is set to false.

Function Block

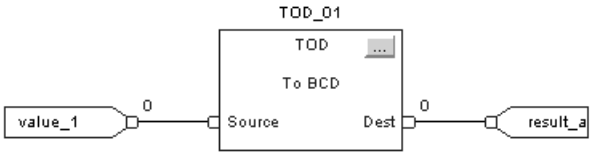
Condition	Action
Prescan	No action taken.
Instruction first scan	No action taken.
Instruction first run	No action taken.
EnableIn is cleared	EnableOut is cleared.
EnableIn is set	The instruction executes. EnableOut is set.
Postscan	No action taken.

Example: The TOD instruction converts *value_1* to a BCD value and places the result in *result_a*.

Relay Ladder



Function Block

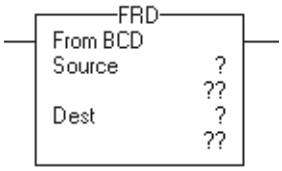


Convert to Integer (FRD)

The FRD instruction converts a BCD value (Source) to a decimal value and stores the result in the Destination.

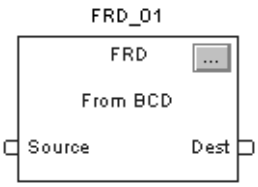
Operands:

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

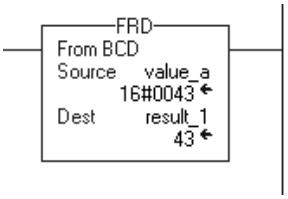
Condition	Action
Prescan	The rung-condition-out is set to false.
Rung-condition-in is false	The rung-condition-out is set to false.
Rung-condition-in is true	The controller converts the Source to a decimal value and places the result in the Destination. The rung-condition-out is set to true.
Postscan	The rung-condition-out is set to false.

Function Block

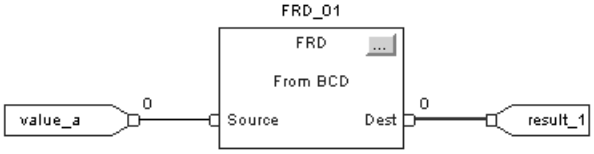
Condition	Action
Prescan	No action taken.
Instruction first scan	No action taken.
Instruction first run	No action taken.
EnableIn is cleared	EnableOut is cleared.
EnableIn is set	The instruction executes. EnableOut is set.
Postscan	No action taken.

Example: The FRD instruction converts *value_a* to a decimal value and places the result in *result_1*.

Relay Ladder



Function Block

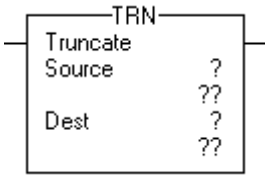


Truncate (TRN)

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

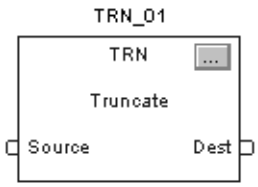
Structured Text

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

Use TRUNC as a function. This function truncates *source* and stores the result in *dest*.

See Structured Text Programming 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

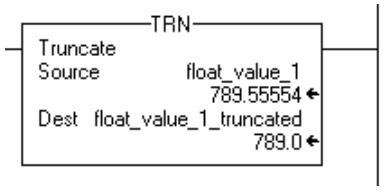
Condition	Action
Prescan	The rung-condition-out is set to false.
Rung-condition-in is false	The rung-condition-out is set to false.
Rung-condition-in is true	The controller removes the fractional part of the Source and places the result in the Destination. The rung-condition-out is set to true.
Postscan	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*.

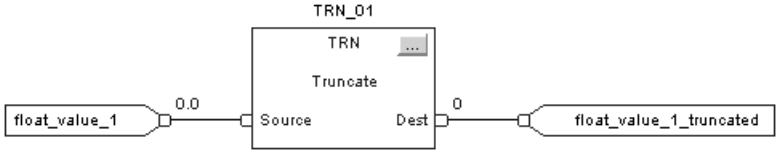
Relay Ladder



Structured Text

```
float_value_1_truncated := TRUNC(float_value_1);
```

Function Block



Notes:

ASCII Serial Port Instructions
(ABL, ACB, ACL, AHL, ARD, ARL, AWA, AWT)

Use the ASCII serial port instructions to read and write ASCII characters.

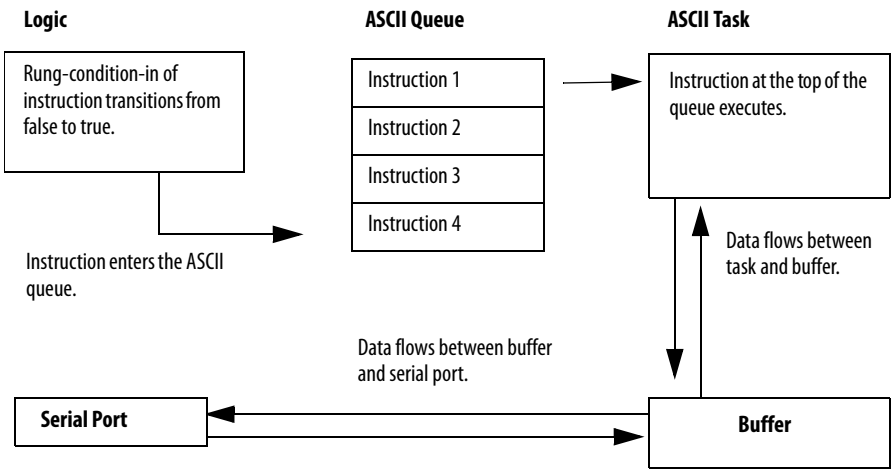
IMPORTANT To use the ASCII serial port instructions, you must configure the serial port of the controller. For procedures, see the Logix5000 Controllers Common Procedures Programming Manual, publication 1756-PM001.

The 1756-L7*x* controllers do not have a serial port and do not use the ASCII Read/Write instructions. In addition, you cannot redirect an ASCII Read/Write instruction to the USB port.

If you want to	For example	Use this instruction	Available in these languages	Page
Determine when the buffer contains termination characters	Check for data that contains termination characters	ABL	Relay ladder Structured text	579
Count the characters in the buffer	Check for the required number of characters before reading the buffer	ACB	Relay ladder Structured text	582
Clear the buffer	• Remove old data from the buffer at start-up. • Synchronize the buffer with a device.	ACL	Relay ladder Structured text	584
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	586
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	590
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	594
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	598
Send characters	Send messages that use a variety of termination characters	AWT	Relay ladder Structured text	602

Instruction Execution

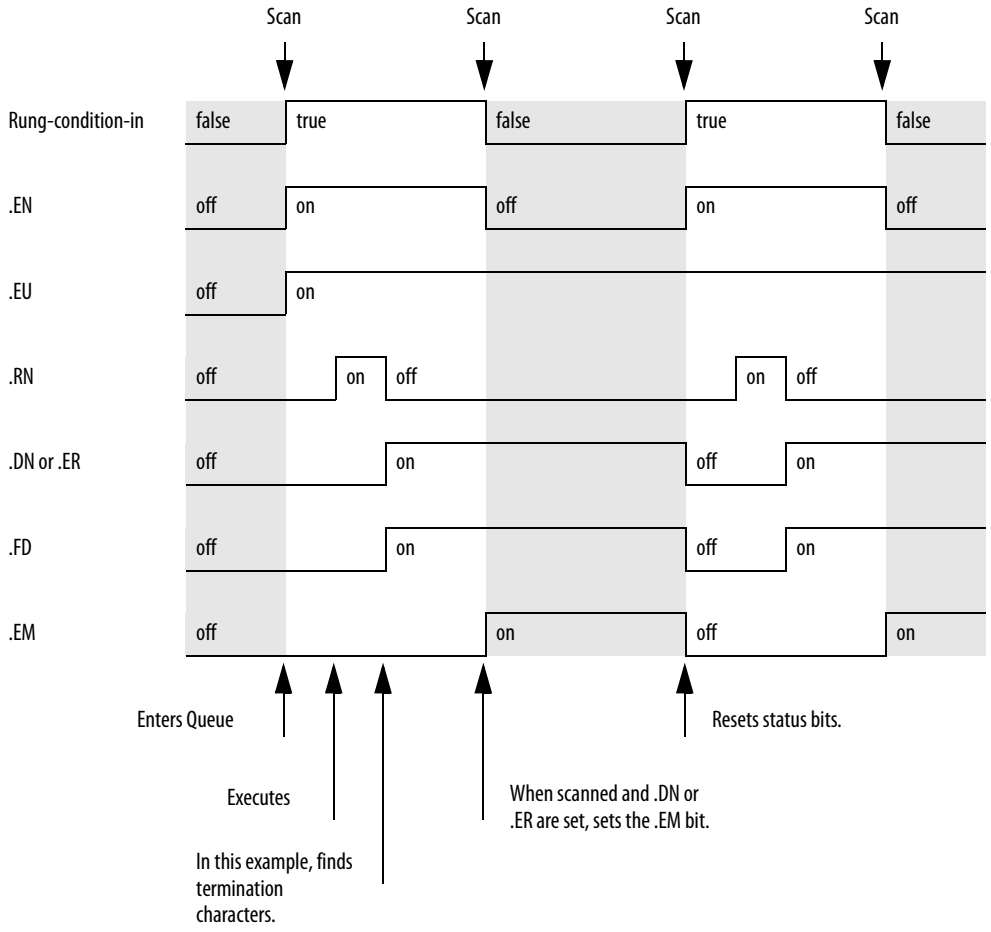
ASCII serial port instructions execute asynchronous to the scan of the logic.



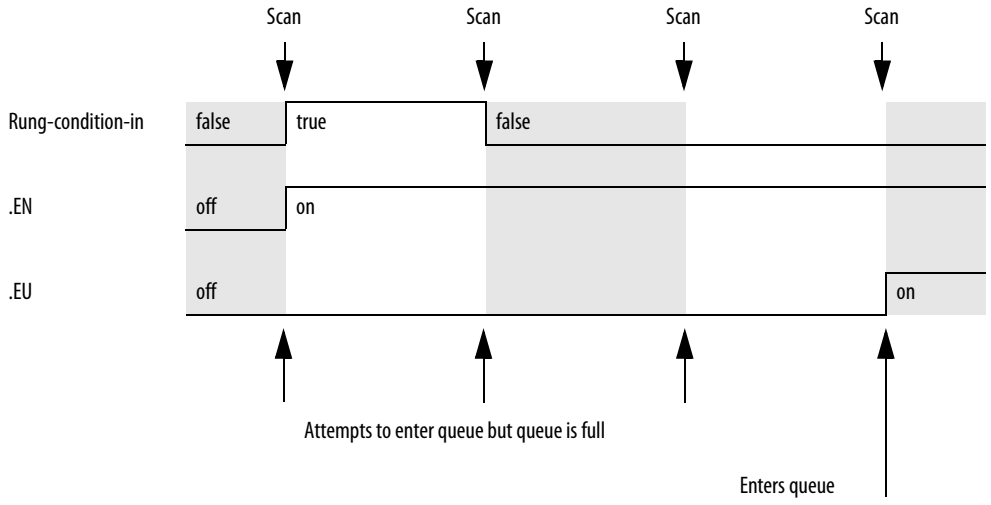
Each ASCII serial port instruction (except ACL) uses a SERIAL_PORT_CONTROL structure to perform these functions:

- 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.

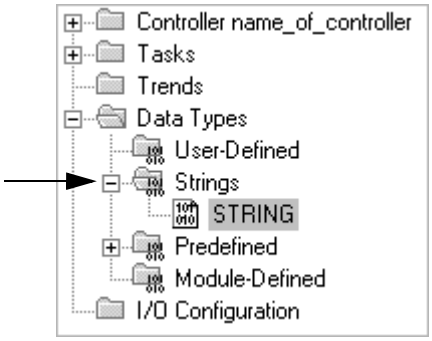


ASCII Error Codes

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 the Logix5000 Controllers Common Procedures Programming Manual, publication 1756-PM001.

Each string data type contains the following members.

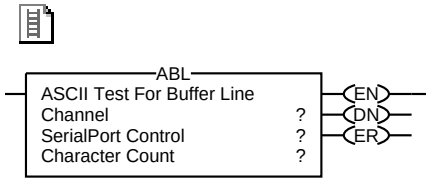
Name	Data Type	Description	Notes
LEN	DINT	Number of characters in the string	The LEN automatically updates to the new count of characters whenever you: • See the String Browser dialog box to enter characters. • See instructions that read, convert, or manipulate a string. The LEN shows the length of the current string. The DATA member may contain additional, old characters, which are not included in the LEN count.
DATA	SINT array	ASCII characters of the string	• To access the characters of the string, address the name of the tag. For example, to access the characters of the *string_1* tag, enter *string_1*. • 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.

Structured Text

```
ABL(Channel
SerialPortControl);
```

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 does the following:

- 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.

Follow these guidelines to program the ABL instruction.

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 executes only on a transition.

Arithmetic Status Flags: Not affected

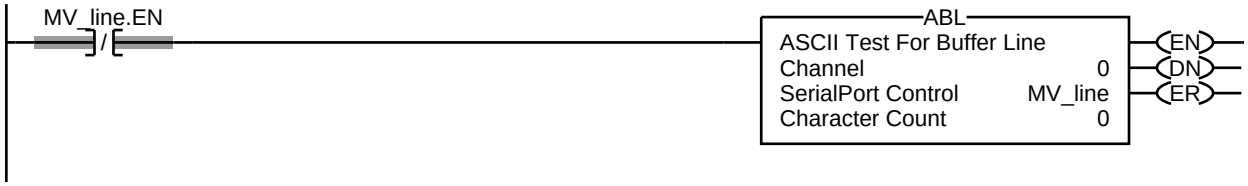
Fault Conditions: None

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.	N/A
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.	N/A
EnableIn is set	N/A	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



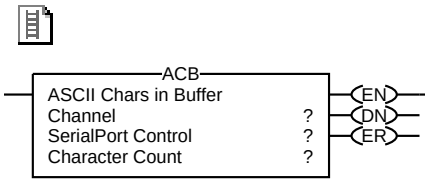
Structured Text

```
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.

Structured Text

ACB(Channel SerialPortControl);

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.

Follow these guidelines to program the ACB instruction.

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 executes only on a transition.

Arithmetic Status Flags: Not affected

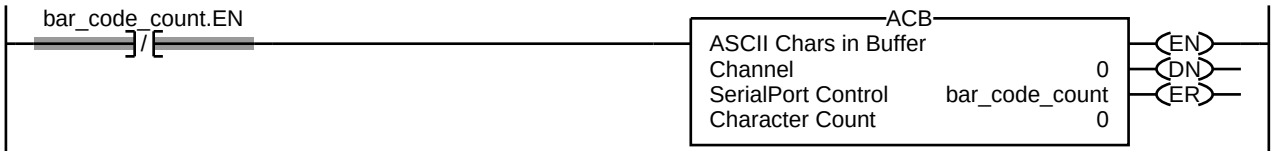
Fault Conditions: None

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.	N/A
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.	N/A
EnableIn is set	N/A	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

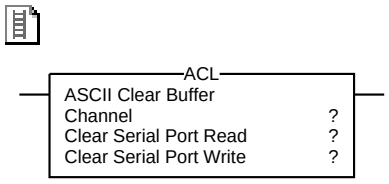
```
ACB(0,bar_code_count);
```

ASCII Clear Buffer (ACL)

The ACL instruction immediately clears the buffer and ASCII queue.

Operands:

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.

Structured Text

```
ACL(Channel,
ClearSerialPortRead,
ClearSerialPortWrite);
```

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

Follow these guidelines to program the ACL instruction.

1. Configure the serial port of the controller:

If Your Application	Then
Uses ARD or ARL instructions	Select User mode
Does not 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.	N/A
Rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	N/A
EnableIn is set	N/A	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

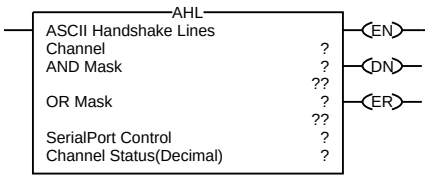
```
osri_1.InputBit := S:FS;

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 do the following:

- 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)

Follow these guidelines to program the AHL instruction.

1. Configure the serial port of the controller.

If your application	Then
Uses ARD or ARL instructions	Select User mode
Does not 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 executes only on a transition.

Arithmetic Status Flags: Not affected

Fault Conditions:

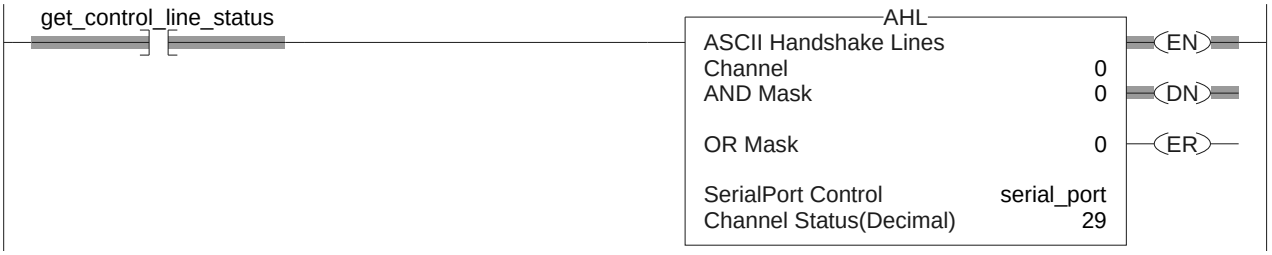
Type	Code	Cause	Recovery Method
4	57	The AHL instruction failed to execute because the serial port is set to no handshaking.	Either: • Change the Control Line setting of the serial port. • Delete the AHL instruction.

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.	N/A
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.	N/A
EnableIn is set	N/A	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

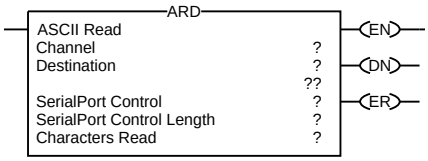
```
osri_1.InputBit := get_control_line_status;

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): • 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.	• If you want to compare, convert, or manipulate the characters, use a string data type. • String data types are: – 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 move to the destination (read)	• 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.

Follow these guidelines to program the ARD instruction.

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 executes only on a transition.
4. To trigger a subsequent action when the instruction is done, examine the EM bit.

Arithmetic Status Flags: Not affected

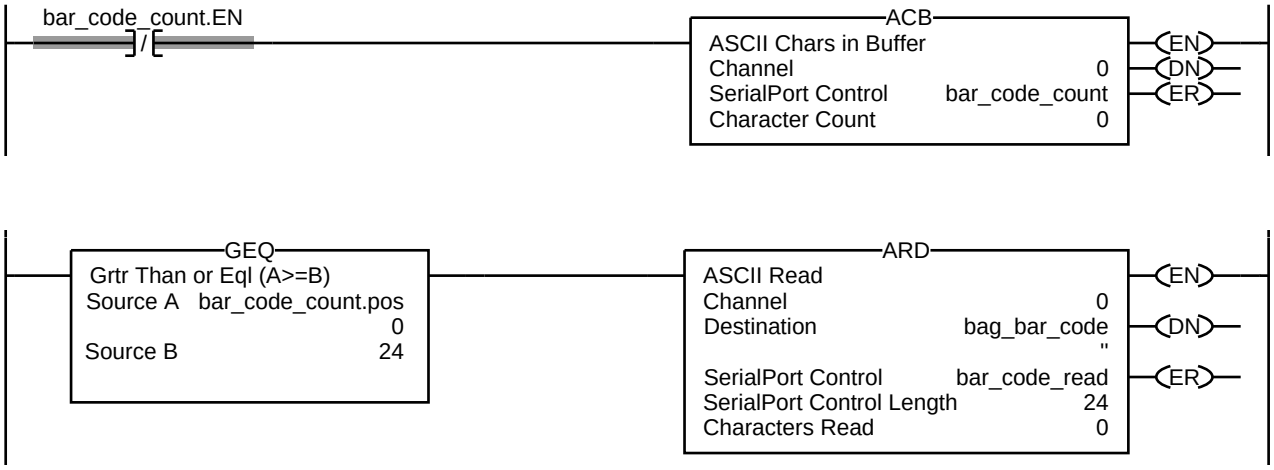
Fault Conditions: None

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.	N/A
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.	N/A
EnableIn is set	N/A	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

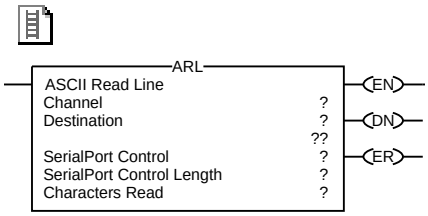
```
ACB(0,bar_code_count);

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): • 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.	• If you want to compare, convert, or manipulate the characters, use a string data type. • String data types are: – 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	• Enter the maximum number of characters that any message will contain (that is, when to stop reading if no termination characters are found). For example, if messages range from 3 to 6 characters in length, enter 6. • 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.

ARL(Channel,Destination,
SerialPortControl);

Structured Text

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 (that is, 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.

- 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.

Follow these guidelines to program the ARL instruction.

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 executes only on a transition.

4. To trigger a subsequent action when the instruction is done, examine the EM bit.

Arithmetic Status Flags: Not affected

Fault Conditions: None

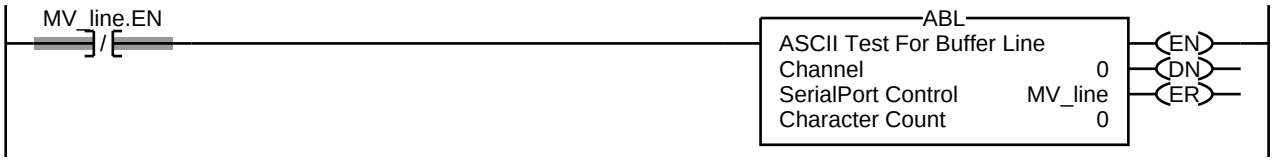
Execution:

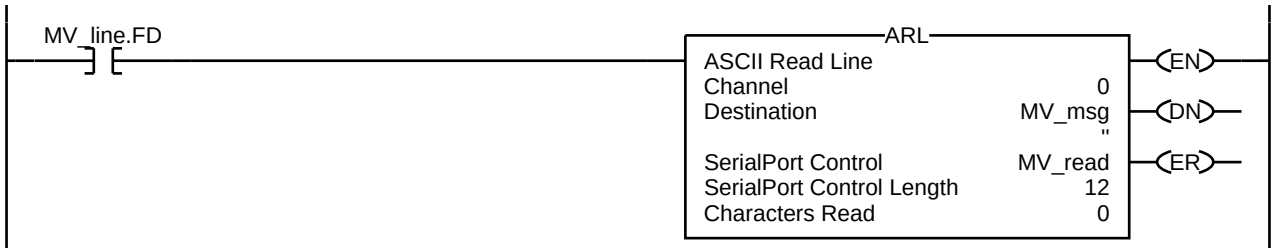
Condition	Relay Ladder Action	Structured Text Action
Prescan	The rung-condition-out is set to false.	No action taken.
Rung-condition-in is false	The rung-condition-out is set to false.	N/A
Rung-condition-in is true	The instruction executes when rung-condition-in toggles from cleared to set. The rung-condition-out is set to true.	N/A
EnableIn is set	N/A	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, its 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

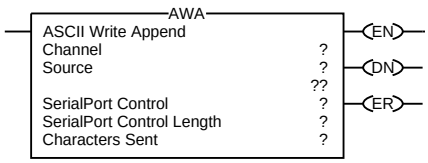
```
ABL(0,MV_line);


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: • 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.	• If you want to compare, convert, or manipulate the characters, use a string data type. • String data types are: – 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	• 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 does the following:

- 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

Follow these guidelines to program the AWA instruction.

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 executes only on a transition.
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

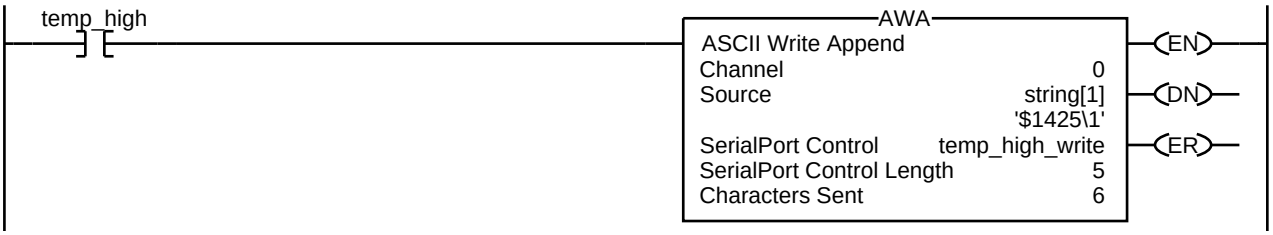
Execution:

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.	N/A
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.	N/A
EnableIn is set	N/A	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

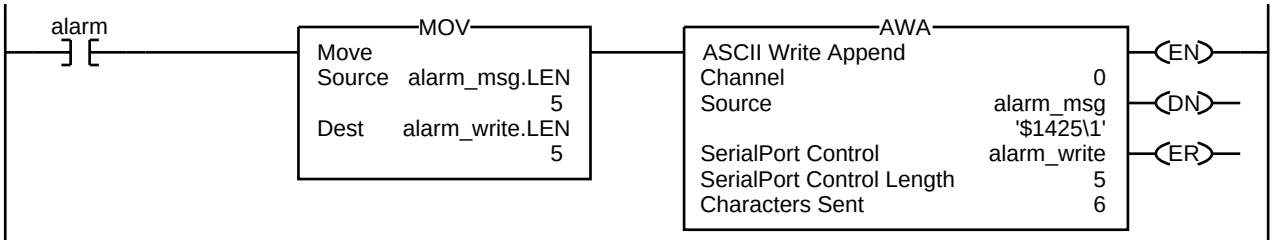


Structured Text

```
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

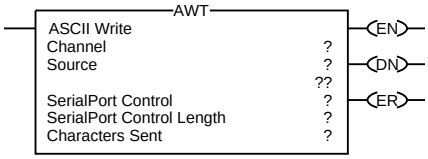
```
osri_1.InputBit := alarm;

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: • 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.	• If you want to compare, convert, or manipulate the characters, use a string data type. • String data types are: – 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	• 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.

Follow these guidelines to program the AWT instruction.

1. Configure the serial port of the controller.

If your application	Then
Uses ARD or ARL instructions	Select User mode
Does not 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 executes only on a transition.
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

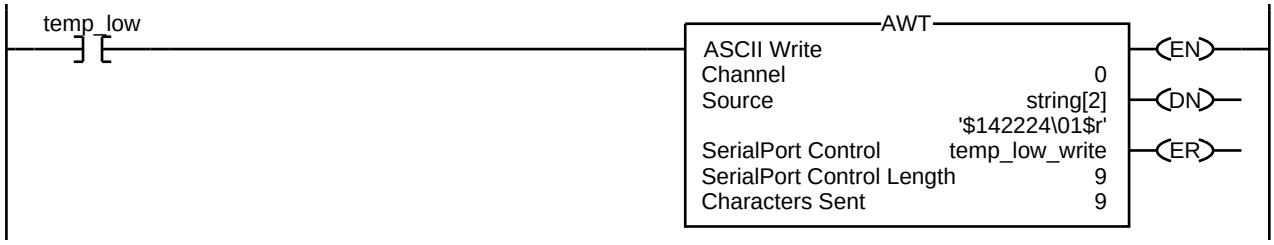
Fault Conditions: None

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.	N/A
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.	N/A
EnableIn is set	N/A	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

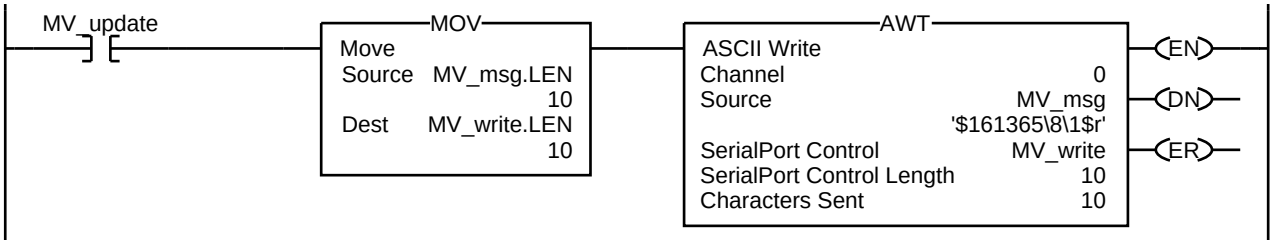


Structured Text

```
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)

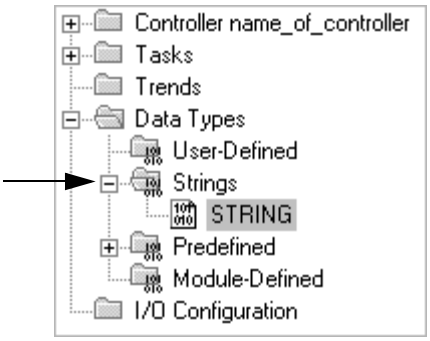
Use the ASCII string instructions to modify and create strings of ASCII characters.

If you want to	For example	Use this instruction	Available in these languages	Page
Add characters to the end of a string	Add termination characters or delimiters to a string	CONCAT	Relay ladder Structured text	610
Delete characters from a string	Remove header or control characters from a string	DELETE	Relay ladder Structured text	612
Determine the starting character of a sub-string	Locate a group of characters within a string	FIND	Relay ladder Structured text	614
Insert characters into a string	Create a string that uses variables	INSERT	Relay ladder Structured text	616
Extract characters from a string	Extract information from a bar code	MID	Relay ladder Structured text	618

You can also use the following instructions to compare or convert ASCII characters.

If you want to	Use this instruction	Page
Compare a string to another string	CMP	219
See if the characters are equal to specific characters	EQU	220
See if the characters are not equal to specific characters	NEQ	251
See if the characters are equal to or greater than specific characters	GEQ	224
See if the characters are greater than specific characters	GRT	228
See if the characters are equal to or less than specific characters	LEQ	232
See if the characters are less than specific characters	LES	236
Rearrange the bytes of a INT, DINT, or REAL tag	SWPB	308
Find a string in an array of strings	FSC	357
Convert characters to a SINT, INT, DINT, or REAL value	STOD	623
Convert characters to a REAL value	STOR	625
Convert a SINT, INT, DINT, or REAL value to a string of ASCII characters	DTOS	627
Convert REAL value to a string of ASCII characters	RTOS	629

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 the Logix5000 Controllers Common Procedures Programming Manual, publication 1756-PM001.

Each string data type contains the following members.

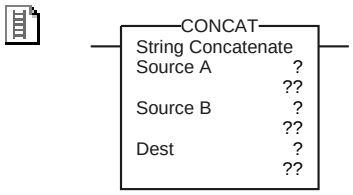
Name	Data Type	Description	Notes
LEN	DINT	Number of characters in the string	The LEN automatically updates to the new count of characters whenever you: •Use the String Browser dialog box to enter characters. •Use instructions that read, convert, or manipulate a string. The LEN shows the length of the current string. The DATA member may contain additional, old characters, which are not included in the LEN count.
DATA	SINT array	ASCII characters of the string	•To access the characters of the string, address the name of the tag. For example, to access the characters of the *string_1* tag, enter *string_1*. •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: •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	

Structured Text

CONCAT(SourceA,SourceB, Dest);

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.	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.	N/A
Rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	N/A
EnableIn is set	N/A	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

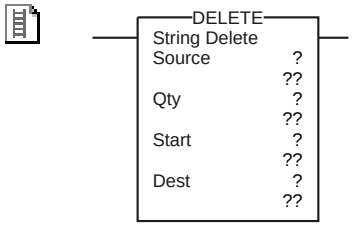
Structured Text

```
IF add_node THEN

CONCAT(string_1,node_num_ascii,msg);

add_node := 0;

END_IF;
```

String Delete (DELETE)

The DELETE instruction does not automatically remove all ASCII characters from a string. An algorithm determines which characters in the string are removed depending on the starting position, quantity, and size of the Source.

Operands:



Relay Ladder

The DELETE instruction does the following:

- Copies the string from the Source to the Destination, ignoring deleted characters and updating the Destination string with the number of characters copied
- Updates the length of the Destination string by the position of characters in the Source string and the number of characters being deleted
- Leaves the Source unchanged unless the Source and the Destination are the same tag

Operand	Type	Format	Enter	Notes
Source	string	Tag	Tag that contains the string from which you want to delete characters	String data types are: •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,

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

Arithmetic Status Flags:

Not affected

Fault Conditions:

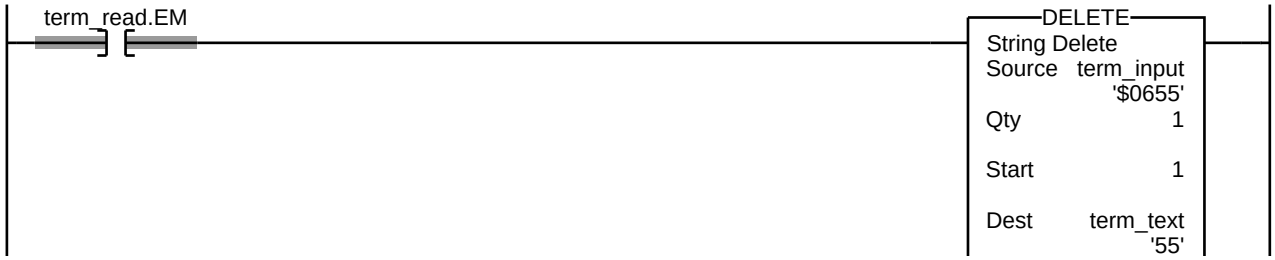
Type	Code	Cause	Recovery Method
4	51	The LEN value of the string tag is greater than the DATA size of the string tag.	1. Check that no instruction is writing to the LEN member of the string tag. 2. In the LEN value, enter the number of characters that the string contains.
4	56	The Start or Quantity value 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.	N/A
Rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	N/A
EnableIn is set	N/A	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

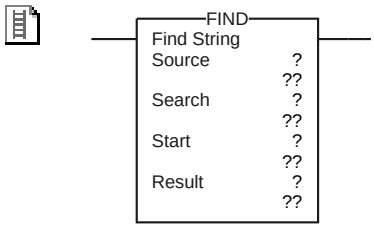
```
IF term_read.EM THEN

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: •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,

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.	N/A
Rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	N/A
EnableIn is set	N/A	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

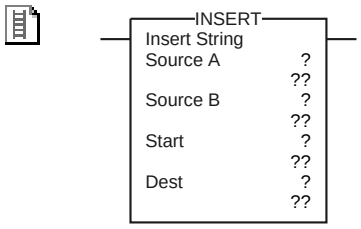
```
IF MV_read.EM THEN

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: •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	

Structured Text

INSERT(SourceA,SourceB,

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 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.	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.	N/A
Rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	N/A
EnableIn is set	N/A	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

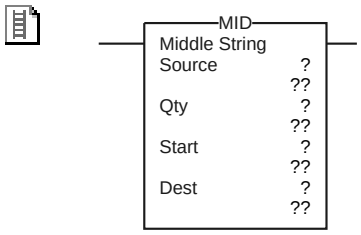
```
IF temp_high THEN

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: •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,

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:

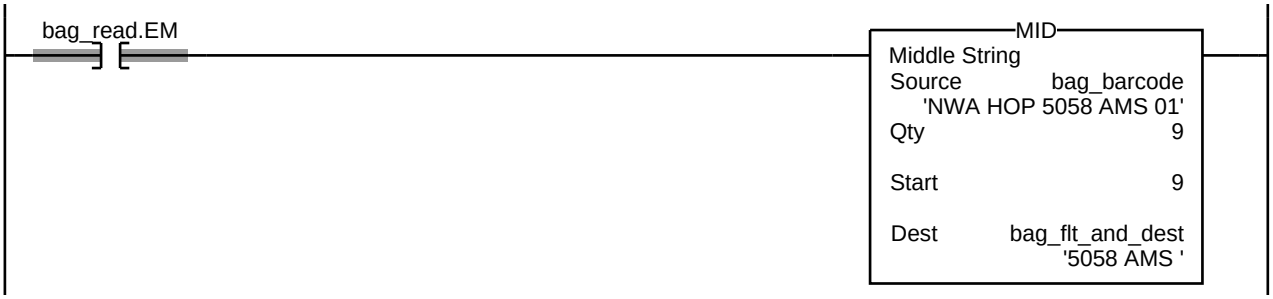
Type	Code	Cause	Recovery Method
4	51	The LEN value of the string tag is greater than the DATA size of the string tag.	1. Check that no instruction is writing to the LEN member of the string tag. 2. In the LEN value, enter the number of characters that the string contains.
4	56	The Start or Quantity value 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.	N/A
Rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	N/A
EnableIn is set	N/A	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;
```

Notes:

ASCII Conversion Instructions
(STOD, STOR, DTOS, RTOS, UPPER, LOWER)

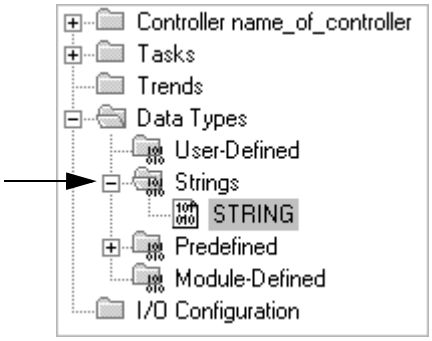
Use the ASCII conversion instructions to alter the format of data.

If you want to	For example	Use this instruction	Available in these languages	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	623
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	625
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	627
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	629
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	631
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	633

You can also use the following instructions to compare or manipulate ASCII characters.

If you want to	Use this instruction	Page
Add characters to the end of a string	CONCAT	610
Delete characters from a string	DELETE	612
Determine the starting character of a sub-string	FIND	614
Insert characters into a string	INSERT	616
Extract characters from a string	MID	618
Rearrange the bytes of a INT, DINT, or REAL tag	SWPB	308
Compare a string to another string	CMP	215
See if the characters are equal to specific characters	EQU	220
See if the characters are not equal to specific characters	NEQ	251
See if the characters are equal to or greater than specific characters	GEQ	224
See if the characters are greater than specific characters	GRT	228
See if the characters are equal to or less than specific characters	LEQ	232
See if the characters are less than specific characters	LES	236
Find a string in an array of strings	FSC	357

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 the Logix5000 Controllers Common Procedures Programming Manual, publication 1756-PM001.

Each string data type contains the following members.

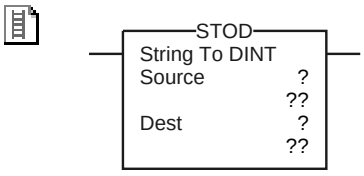
Name	Data Type	Description	Notes
LEN	DINT	Number of characters in the string	The LEN automatically updates to the new count of characters whenever you do the following: • Use the String Browser dialog box to enter characters. • Use instructions that read, convert, or manipulate a string. The LEN shows the length of the current string. The DATA member may contain additional, old characters, which are not included in the LEN count.
DATA	SINT array	ASCII characters of the string	• To access the characters of the string, address the name of the tag. For example, to access the characters of the *string_1* tag, enter *string_1*. •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: • 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).

Structured Text

STOD(Source,Dest);

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 (for example, /), the instruction converts only the first group of numbers.

Arithmetic Status Flags:

Arithmetic status flags are affected.

Fault Conditions

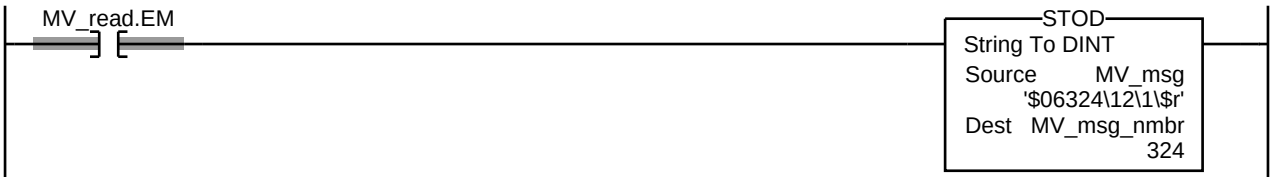
Type	Code	Cause	Recovery Method
4	51	The LEN value of the string tag is greater than the DATA size of the string tag.	1. Check that no instruction is writing to the LEN member of the string tag. 2. In the LEN value, enter the number of characters that the string contains.
4	53	The output number is beyond the limits of the destination data type.	Either: • 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.	N/A
Rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	N/A
EnableIn is set	N/A	EnableIn is always set. The instruction executes.
Instruction execution	SC 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

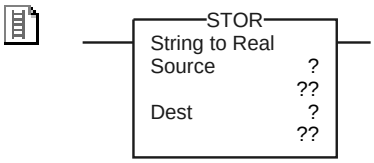
```
IF MV_read.EM THEN

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: • Default STRING data type • Any new string data type that you create
Destination	REAL	Tag	Tag to store the REAL value	

Structured Text

STOR(Source,Dest);

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 (for example, /), 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: • Reduce the size of the ASCII value. • Use a larger data type for the destination.

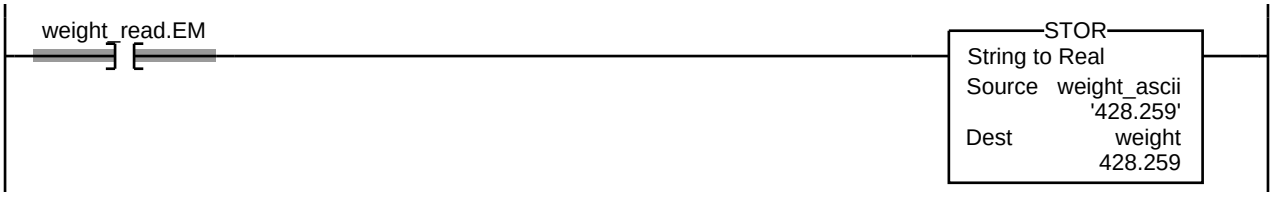
Execution:

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.	N/A
Rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	N/A
EnableIn is set	N/A	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

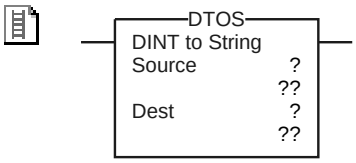
```
IF weight_read.EM THEN

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. .
Destination	String	Tag	Tag to store the ASCII value	String data types are: • default STRING data type. • any new string data type that you create.

Structured Text

DTOS(Source,Dest);

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:

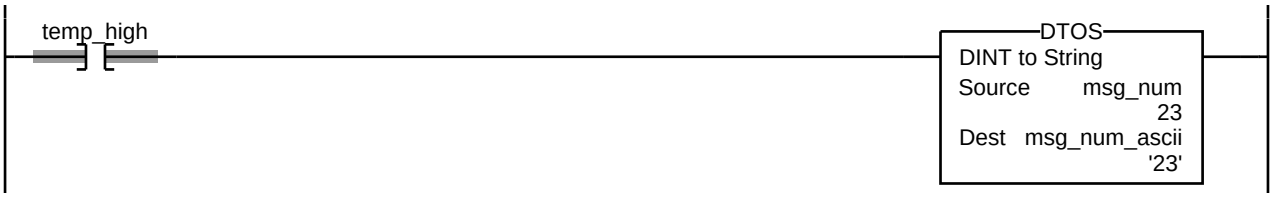
Type	Code	Cause	Recovery Method
4	52	The output string is larger than the destination.	Create a new string data type that is large enough for the output string. Use the new string data type as the data type for the destination.

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.	N/A
Rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	N/A
EnableIn is set	N/A	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

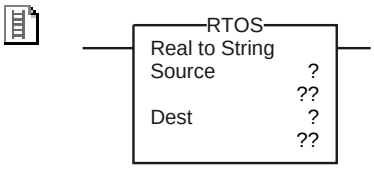
```
IF temp_high THEN

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: • Default STRING data type • Any new string data type that you create

Structured Text

RTOS(Source,Dest);

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	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.	N/A
Rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	N/A
EnableIn is set	N/A	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

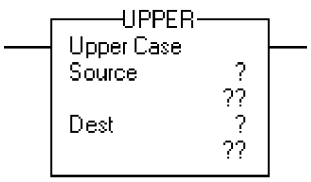
```
IF send_data THEN

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

Structured Text

UPPER(Source,Dest);

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.	N/A
Rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	N/A
EnableIn is set	N/A	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

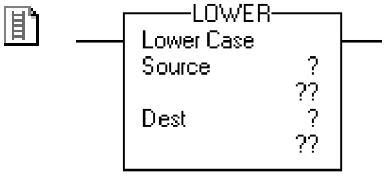
Structured Text

```
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:

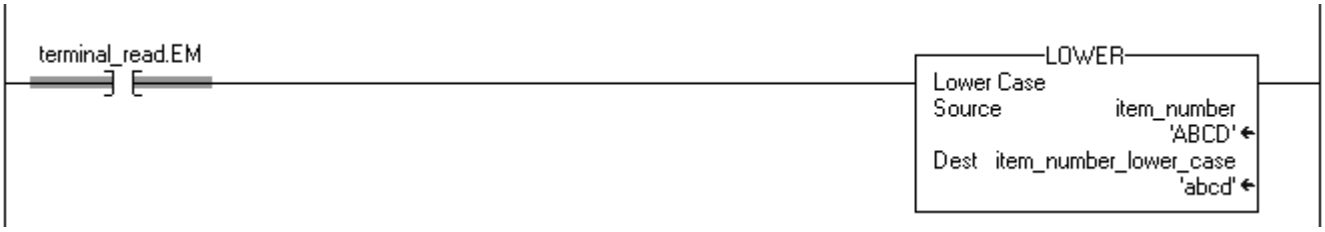
Condition	Relay Ladder Action	Structured Text Action
Prescan	The rung-condition-out is set to false.	No action taken.
Rung-condition-in is false	The rung-condition-out is set to false.	N/A
Rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	N/A
EnableIn is set	N/A	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;
```

Debug Instructions

(BPT, TPT)

Use the debug instructions to monitor the state of your logic when it is in conditions that you determine. These instructions are compatible only with RSLogix Emulate 5000 software, with which you can emulate a Logix5000 controller on your personal computer.

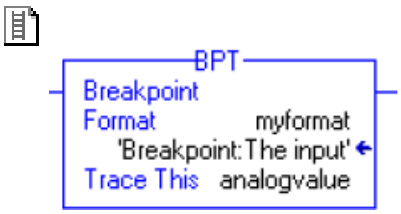
If you want to	Use this instruction	Available in these languages	Page
Stop program emulation when a rung is true	BPT	Relay ladder	635
Log data you select when a rung is true	TPT	Relay ladder	639

Breakpoints (BPT)

Breakpoints stop program emulation when a rung is true.

Operands:

Relay Ladder



Operand	Type	Format	Description
Format	String	Tag	A string that sets the formatting for the text that appears in the trace window for the breakpoint.
Trace This	BOOL, SINT, INT, DINT, REAL	Tag	The tag that has a value you want to display in the trace window.

Description: Breakpoints are programmed with the Breakpoint output instruction (BPT). When the inputs on a rung containing a BPT instruction are true, the BPT instruction stops program execution. The software displays a window indicating that the breakpoint triggered and the values that triggered it.

When a breakpoint triggers, the emulator displays a window informing you that a breakpoint occurred. The title bar of the window shows the slot containing the emulator that encountered the breakpoint.

When you click OK, the emulator resumes program execution. If the conditions that triggered the breakpoint persist, the breakpoint will recur.

In addition, the emulator opens a trace window for the breakpoint. The trace window displays information about the breakpoint and the values.

IMPORTANT	**ATTENTION:** When a breakpoint triggers, you will not be able to edit your project until you permit the execution to continue. You can go online with the emulator to observe the state of your project, but you will not be able to edit it. If you try to accept a rung edit while a breakpoint is triggered, you will see a dialog box saying the controller is not in the correct mode.

String Format

With the Format string in the tracepoint and breakpoint instructions, you can control how the traced tags appear in the traces or breakpoint windows. The format of the string is:

heading:(text)%(type)

where *heading* is a text string identifying the tracepoint or breakpoint, *text* is a string describing the tag (or any other text you choose), and %(*type*) indicates the format of the tag. You need one type indicator for each tag you are tracing with the tracepoint or breakpoint instruction.

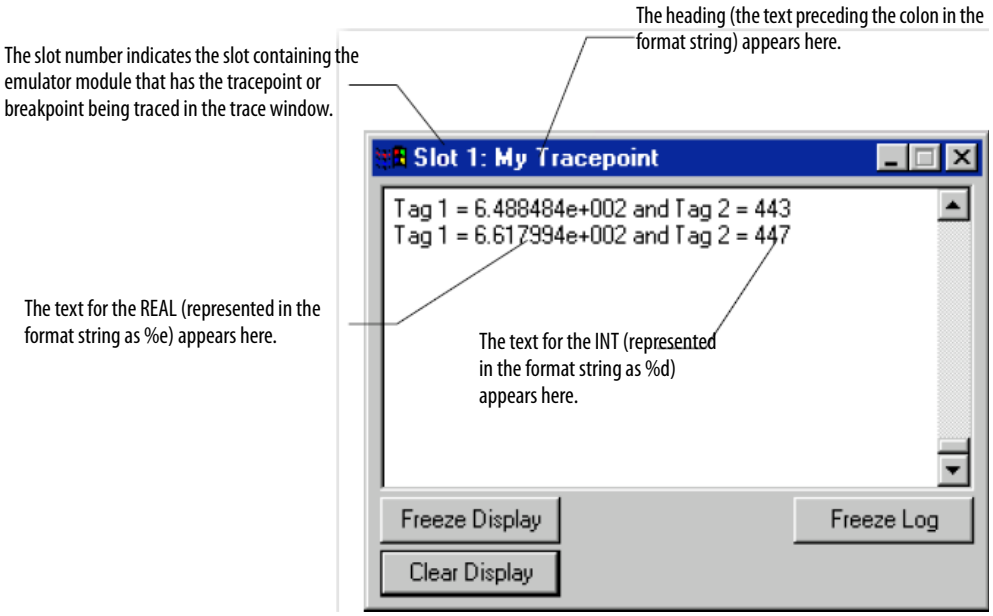
For example, you could format a tracepoint string as shown.

My tracepoint:Tag 1 = %e and Tag 2 = %d

The %e formats the first traced tag as double-precision float with an exponent, and %d formats the second traced tag as a signed decimal integer.

In this case, you would have a tracepoint instruction that has two Trace This operands (one for a REAL and one for an INT, although the value of any tag can be formatted with any flag).

The resulting tracepoint window that would appear when the tracepoint is triggered would look like the example.



Arithmetic Status Flags: Not affected

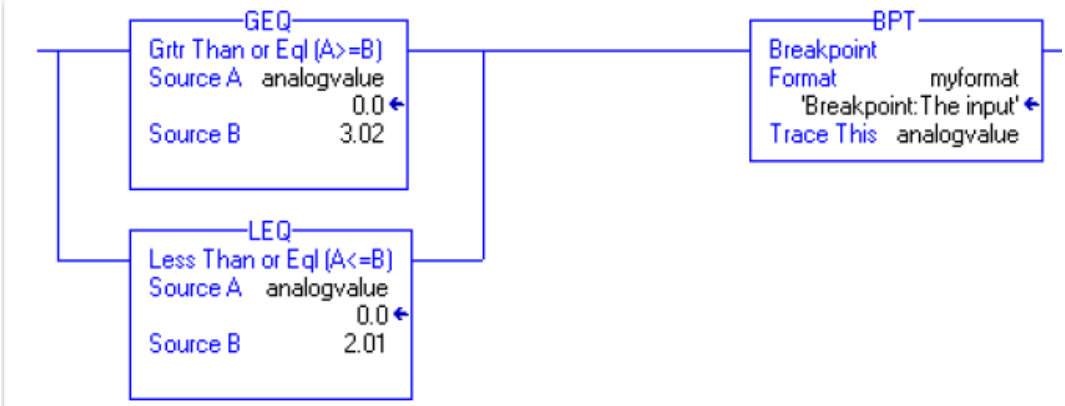
Fault Conditions: None

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 jumps to the rung that contains the LBL instruction with the referenced label name.
Postscan	The rung-condition-out is set to false.

Example: You can display many tag values with the BPT instruction. However, the formatting string can contain only 82 characters. Because the formatting string requires two characters for each tag you want in the breakpoint, you cannot trace more than 41 tags with a single BPT instruction. However, to separate tag data in your traces, you will need to include spaces and other formatting, thus reducing the number of tag values that one BPT instruction can effectively display to far fewer than 41.

This rung shows a breakpoint that stops program execution when an analog value is greater than 3.02 or less than 2.01.

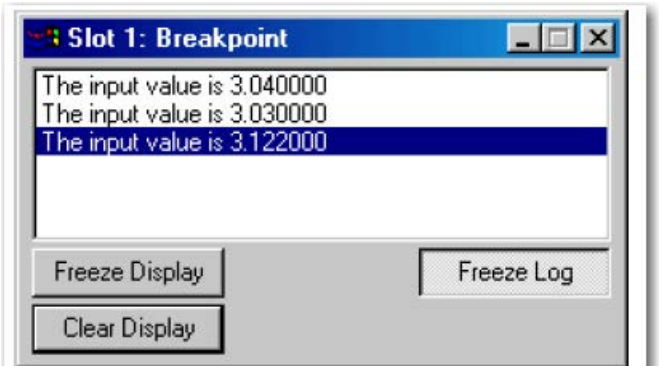


You want to display the breakpoint information in the Format string (myformat). In this case, the format string contains the following text:

```
Breakpoint:The input value is %f
```

When the breakpoint triggers, the breakpoint trace window shows the characters before the colon ('Breakpoint') in the title bar of the trace window. The other characters make up the traces. In this example, %f represents the first (and in this case, the only) tag to be traced ('analogvalue').

The resulting traces appear as shown here.

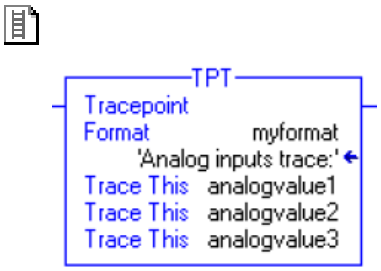


Tracepoints (TPT)

Trace points log data you select when a rung is true.

Operands:

Relay Ladder



Operand	Type	Format	Description
Format	String	Tag	A string that sets the formatting for the trace reports (both on-screen and logged to disk).
Trace This	BOOL, SINT, INT, DINT, REAL	Tag	The tag you want to trace.

Description:

Tracepoints are programmed with the tracepoint output instruction (TPT). When the inputs on a rung containing a TPT instruction are true, the TPT instruction writes a trace entry to a trace display or log file.

You can trace many tags with the TPT instruction. However, the formatting string can contain only 82 characters. Because the formatting string requires two characters for each tag you want to trace, you cannot trace more than 41 tags with a single TPT instruction. However, to separate tag data in your traces, you will need to include spaces and other formatting, thus reducing the number of tags that one TPT instruction can effectively trace to far fewer than 41.

String Format

With the Format string in the tracepoint and breakpoint instructions, you can control how the traced tags appear in the traces or breakpoint windows. The format of the string is as shown here:

heading:(text)%(type)

where *heading* is a text string identifying the tracepoint or breakpoint, *text* is a string describing the tag (or any other text you choose), and %(*type*) indicates the format of the tag. You need one type indicator for each tag you are tracing with the tracepoint or breakpoint instruction.

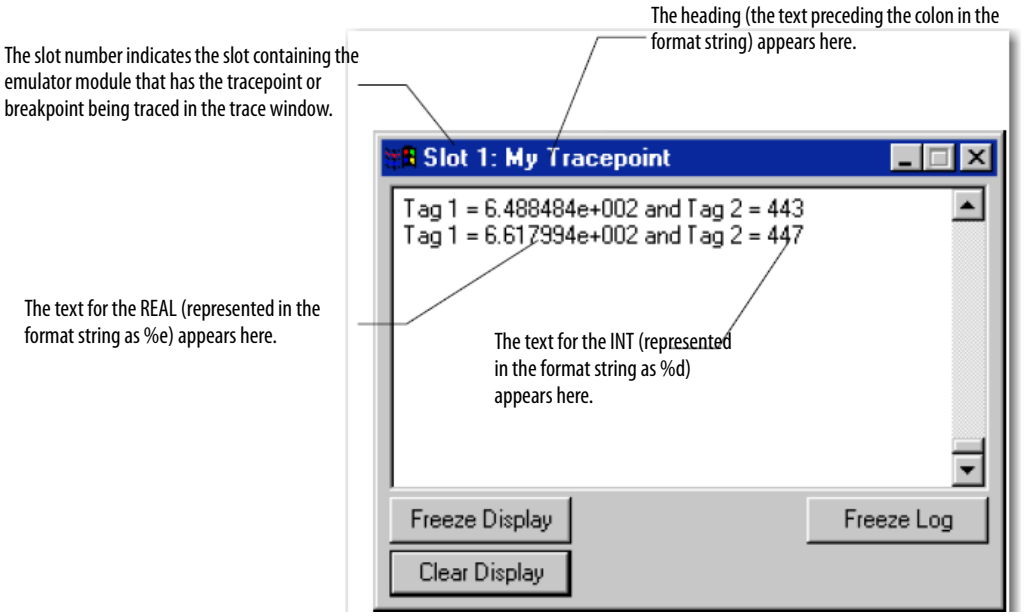
For example, you could format a tracepoint string as shown.

My tracepoint:Tag 1 = %e and Tag 2 = %d

The %e formats the first traced tag as double-precision float with an exponent, and %d formats the second traced tag as a signed decimal integer.

In this case, you would have a tracepoint instruction that has two Trace This operands (one for a REAL and one for an INT, although the value of any tag can be formatted with any flag).

The resulting tracepoint window that would appear when the tracepoint is triggered would look like the example.



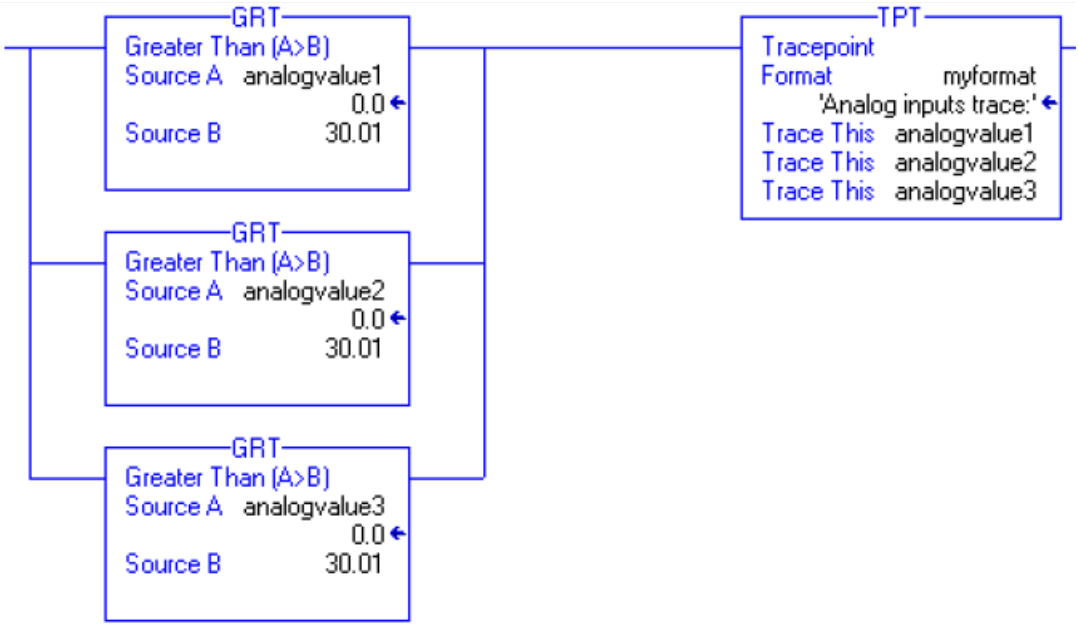
.Arithmetic Status Flags: Not affected

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 jumps to the rung that contains the LBL instruction with the referenced label name.
Postscan	The rung-condition-out is set to false.

Example: This rung triggers a trace of three analog values when any one of them exceeds a given value (30.01).

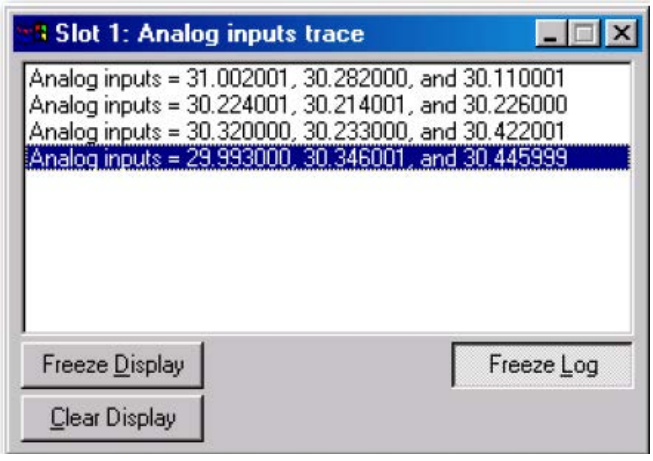


You want to display the tracepoint information in the Format string (myformat). In this case, the format string contains this text:

```
Analog inputs trace:Analog inputs = %f, %f, and %f
```

When the tracepoint triggers, the characters before the colon ('Analog inputs trace') appear in the title bar of the trace window. The other characters make up the traces. In this example, %f represents the tags to be traced ('analogvalue1,' 'analogvalue2,' and 'analogvalue3').

The resulting traces appear as shown here.



When this trace is logged to disk, the characters before the colon appear in the traces.

This indicates which tracepoint caused which trace entry. This is an example of a trace entry. 'Analog inputs trace:' is the heading text from the tracepoint's format string.

Analog inputs trace: Analog inputs = 31.00201, 30.282000, and 30.110001.

Common Attributes

Introduction

This appendix describes attributes that are common to the Logix instructions.

Status Flags

To access controller configuration and status in your logic, use the Get System Value (GSV) and Set System Value (SSV) instructions. There is a also a set of status Status Flags that you can access directly with relay and structured text instructions. These flags are not tags and you cannot create aliases for them.

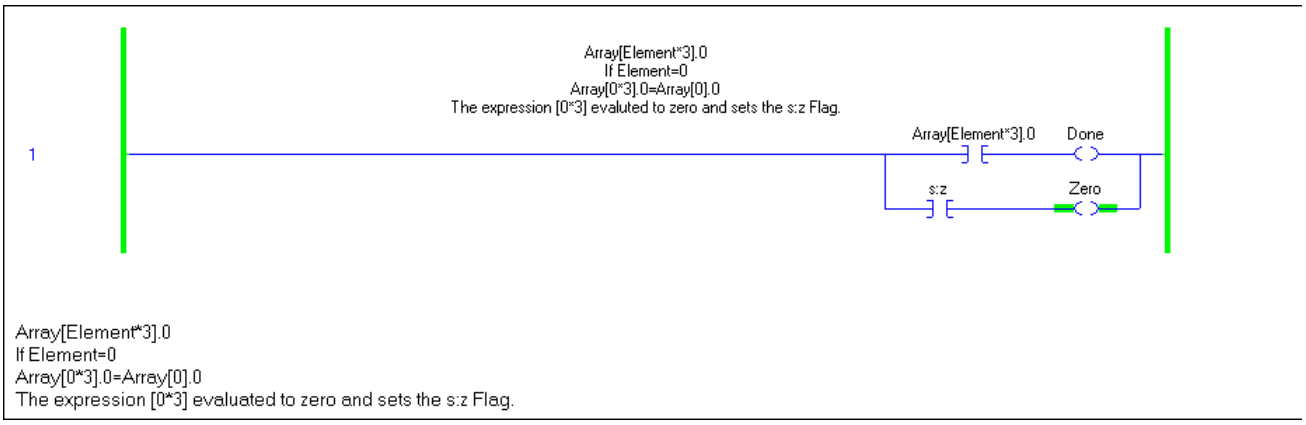
Status Flag	Description
S:FS	First scan flag The first scan flag is set by the controller on the first scan of routines in a program. This flag can be used with only the XIC and XIO relay instructions or in structured text like a Boolean variable. You can use this flag to initialize data for use in later scans. Sometimes called the 'first pass' bit.
S:N	Negative flag The controller sets the negative flag when the result of an arithmetic or logical operation is a negative value. This flag can be used only with XIC, XIO, OTE, OTL and OTU relay instructions or in structured text like a Boolean variable. You can use this flag as a quick test for a negative value. Using S:N is more efficient than using the CMP instruction. This math status flag is cleared at the start of executing an instruction capable of setting this flag. Sometimes called the 'sign' bit. If a computation results in a NAN value, the sign bit is irrelevant and could be positive or negative. In this situation, the software displays 1#.NAN with no sign. An integer divide returns the numerator when the denominator is 0, but does not set this flag.

Status Flag	Description
S:Z	Zero flag The zero flag is set by the controller when the result of an arithmetic or logical operation is zero. This flag can be used with only the XIC, XIO, OTE, OTL and OTU relay instructions or in structured text like a Boolean variable. You can use this flag as a quick test for a zero value. Using S:Z is more efficient than using the CMP instruction. The math status flag is cleared at the start of executing an instruction capable of setting this flag. An integer divide returns the numerator when the denominator is 0, but does not set this flag.
S:V	Overflow flag The controller sets the overflow flag when the result of an arithmetic operation resulted in an overflow. This flag can be used with only the XIC, XIO, OTE, OTL and OTU relay instructions or in structured text like a Boolean variable. You can use this flag to check that the result of an operation is still in range. For example, adding 1 to a SINT generates an overflow when the value goes from 127…128. The overlow flag generates a minor fault when it transitions from false to true. This math status flag is cleared at the start of executing an instruction capable of setting this flag. Sometimes called the 'overflow' bit. The MOD instruction does not capture an overflow that may have occurred during the calculation. If an overflow occurs, a minor fault will be generated, but the S:V bit will not be set.
S:C	Carry flag The controller sets the carry flag when the result of an arithmetic operation resulted in the generation of a carry out of the most significant bit. This flag can be used with only the XIC, XIO, OTE, OTU and OTL relay instructions or in structured text like a Boolean variable. This math status flag is cleared at the start of executing an instruction capable of setting this flag. Sometimes called the 'carry' bit.
S:MINOR	Minor fault flag The controller sets the minor fault flag when there is at least one minor program fault. This flag can be used with only the XIC, XIO, OTE, OTL and OTU relay instructions or in structured text like a Boolean variable. You could use this flag to test that a minor fault has occurred and take appropriate action. This bit is triggered only by programming faults (like overflow). It will not be triggered by a battery fault. The bit is cleared on every scan. An OTL can be used to explicitly program a minor fault occurrence.

Expressions in Array Subscripts

Expressions set status flags based on the results of arithmetic operations. If you have an array subscript as an expression, both the expression and the instruction could generate minor faults.

Under certain circumstances, the S:Z status flag may be set if the expression of an array element evaluates to zero. In this example, Array[Element*3], 0 if Element=0 the subscript [Element*3] evaluates to 0 and the S:Z bit will be set.



Immediate Values

When you enter an immediate value (constant) in decimal format (for example, -2, 3) the controller stores the value by 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).

IMPORTANT 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)

Floating Point Values

Logix controllers handle floating point values according to the IEEE 754 standard for binary floating-point arithmetic. This standard defines how floating point numbers are stored and calculated. The IEEE 754 standard for floating point math was designed to provide speed and the ability to handle very large numbers in a reasonable amount of storage space.

Not all decimal values can be represented in binary format to the exact value, which results in a loss of precision. This standard is widely used throughout industry, including in computer operating systems. As a result, there is no way to control the number of digits to the right of the decimal point in logic in a Logix5000 controller.

For example, in most cases, if you subtract 10 from 10.1, you expect the result to be 0.1. In a Logix controller, the result could very well be 0.10000038. In this example, the difference between 0.1 and 0.10000038 is .000038%, or practically zero. For most operations, this small inaccuracy is insignificant. To put things in perspective, if you were sending a floating point value to an analog output module, there would be no difference in the output voltage for a value being sent to the module that differs by .000038%. Again, this applies to most cases, but not necessarily every time.

Guidelines for Floating-point Math Operations

Follow these guidelines:

- When performing certain floating-point math operations, there may be a loss of precision and you may receive unexpected results. In general, floating-point numbers are quite accurate, but floating-point processors have their own internal precision that can impact resultant values.
- Do not use floating point math for money values or for totalizer functions. Use INT or DINT values, scale the values up, and keep track of the decimal place (or use one INT or DINT value for dollars, and a second INT or DINT value for cents).
- Do not compare floating-point numbers. Instead, check for values within a range. The LIM instruction is provided specifically for this purpose.

Totalizer Examples

The IEEE 754 standard affects totalization applications such that errors occur when adding very small numbers to very large numbers. The standard requires exponents in the two operands to be the same. Since the fractional component is only 23 bits, as the exponent gets larger, the fractional component approaches zero.

This can be seen when adding 1 to a number over a period of time. When the sum is in the 16 million range, the number 1 becomes 0 because the exponent is so large, a 1 is insignificant, and gets shifted out of the equation. The result is that a 0 is added instead of a 1.

To work around this, do math on small numbers until the results get large. Then, transfer them to another location for additional large-number math. For example:

x is the small incremented variable.

y is the large incremented variable.

z is the total current count that can be used anywhere.

```
x-x+1;

if x = 100,000;

        {

        y = y + 100,000;

        x = 0;

        }

z = y + x;
```

Or another example:

```
x+ x + some_tiny_number;

if (x >= 100)

        {

        z += 100;

        x -= 100; // there might be a tiny remainder ...

        }
```

Data Conversions

Data conversions occur when you mix data types in your programming.

When programming	Conversions can occur when you
Relay Ladder Logic	Mix data types for the parameters within one instruction
Function Block	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 support only 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 (for example, 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 doing the following:

- 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.

Data Types

LINT Data Type Considerations

When using LINT data types, many limitations apply. A LINT data type cannot be used in most instructions. Limitations are as follows:

- The LINT data type is not supported in Machine Edition. The LINT data type is a 64-bit word; ME uses only 32-bit words.
- The LINT data type is not supported by HMIs.
- The LINT data type is not supported in most instructions.

TIP LINTs can only be used with Move and Copy instructions. They are used with the CST/WallClock Time attribute, time synchronization and Add-On Instructions.You cannot Add, Subtract, Multiply, or Divide this tag type.

When using LINT data types, consider the following descriptions when these issues occur.

How to	Description
Move/copy two double-integer DINT values into one LINT	Create a double integer array of two elements, total of 64 bits (that is, DINT[2], which can then be copied into one long integer.
Correct Date/Time Display error	When a tag has a negative value, it cannot be displayed as Date/Time. In the tag editor, check whether the value is negative by changing the style of the tag from Date/Time to Binary. When the most significant bit (leftmost one) is 1, the value is negative and therefore cannot be displayed as a Date or Time.

SINT or INT to DINT

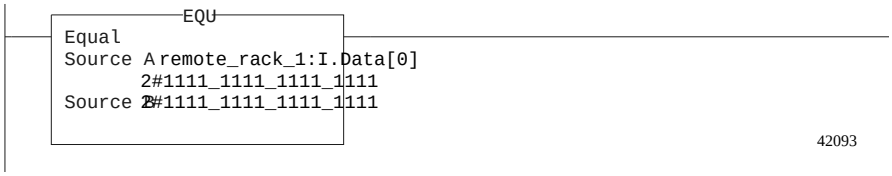
For those instructions that convert SINT or INT values to DINT values, the 'Operands' sections in this manual identify the conversion method.

This conversion method	Converts data by placing
Sign-extension	The value of the leftmost 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.



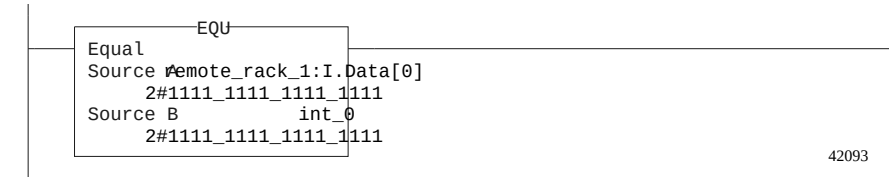
If you use a SINT or INT tag and an immediate value in an instruction that converts data by sign-extension, use one of these methods to handle immediate values.

- 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 leftmost 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.

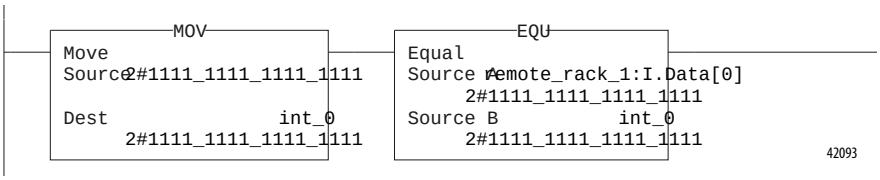
IMPORTANT Mixing an INT tag with an immediate value

ATTENTION: Since *remote_rack_1:I.Data[0]* is an INT tag, the value to check it against is also entered as an INT tag.



IMPORTANT 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 by 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.

IMPORTANT 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 in the following examples.

- Numbers other than *x*.5 round to the nearest whole number.
- *X*.5 rounds to the nearest even number.

The following example shows the result of converting REAL values to DINT values.

IMPORTANT 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

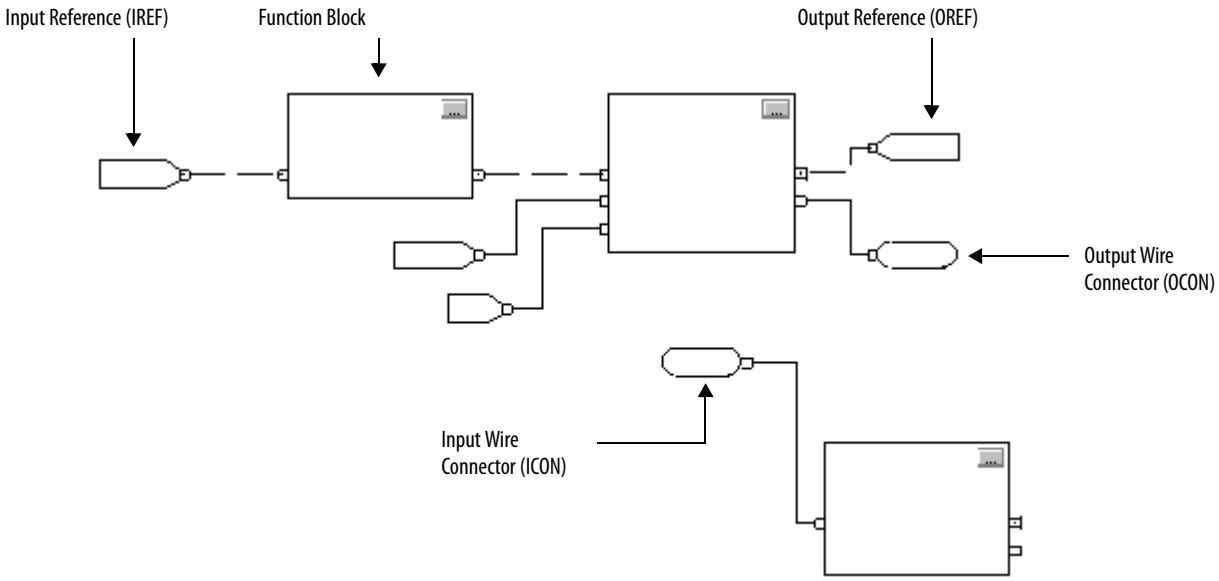
Introduction

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 $+/-10^{+/-15}$ because internal floating point calculations are done by 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 ($+/-10^{+/-38}$).

Function Block Elements

To control a device, use these elements.

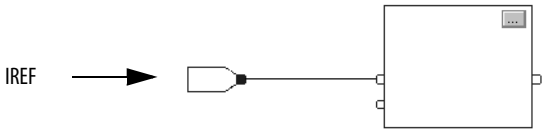


Use the table to choose your function block elements.

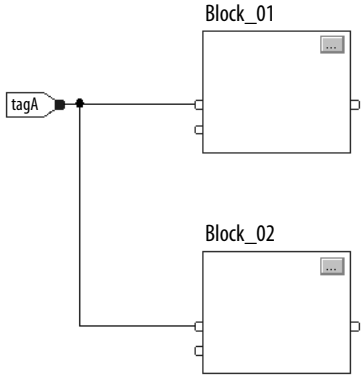
If you want to	Use a
Supply a value from an input device or tag	Input reference (IREF)
Send a value to an output device or tag	Output reference (OREF)
Perform an operation on an input value or values and produce an output value or values	Function block
Transfer data between function blocks when they are: •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

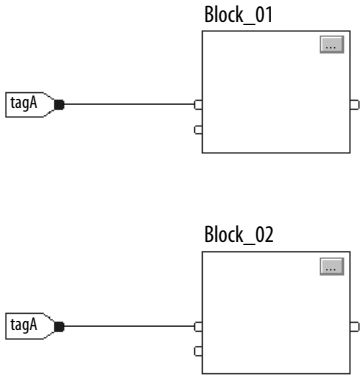
If you use an IREF to specify input data for a function block instruction, the data in that IREF is latched for the scan of the function block routine. The IREF latches data from program-scoped and controller-scoped tags. The controller updates all IREF data at the beginning of each scan.



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 Blcock_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.

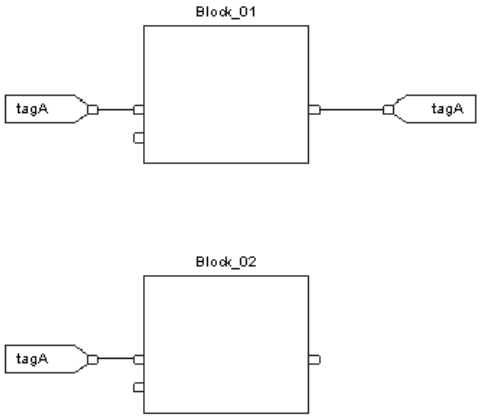


This example is the same as the one above. The value of tagA is stored only once at the beginning of the routine's execution. The routine uses this stored value throughout the routine.



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

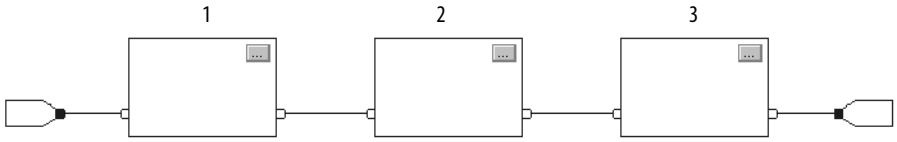
The Logix Designer application automatically determines the order of execution for the function blocks in a routine when you do the following:

- Verify a function block routine.
- Verify a project that contains a function block routine.
- Download a project that contains a function block routine.

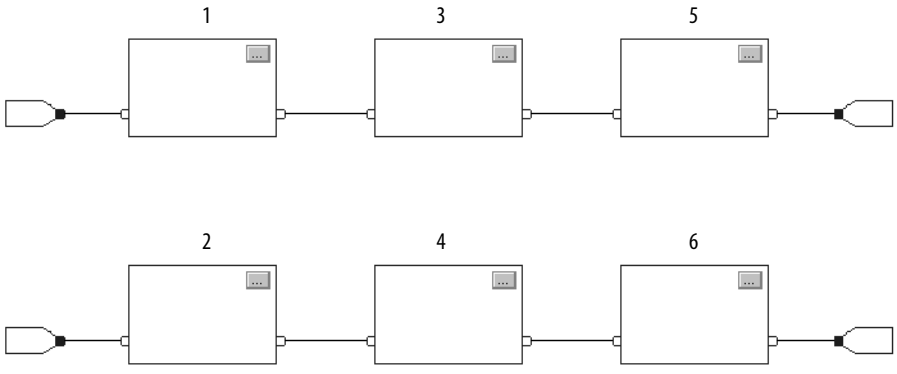
You define execution order by wiring function blocks together and indicating the data flow of any feedback wires, if necessary.

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.

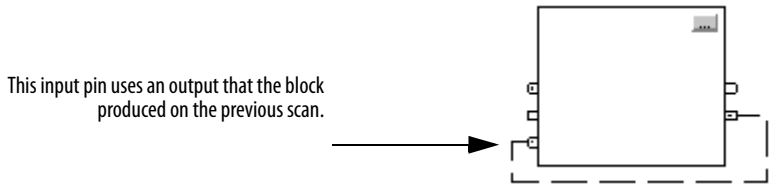


Execution order is relative only to the blocks that are wired together. The following example is fine because the two groups of blocks are not wired together. The blocks within a specific group execute in the appropriate order in relation to the blocks in that group.

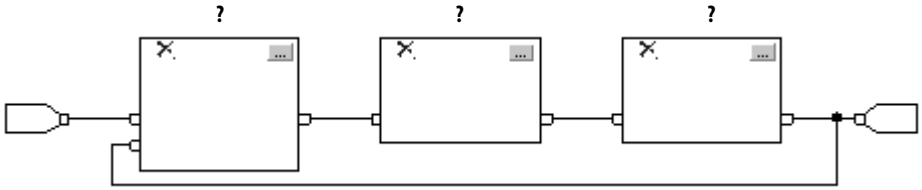


Resolve a Loop

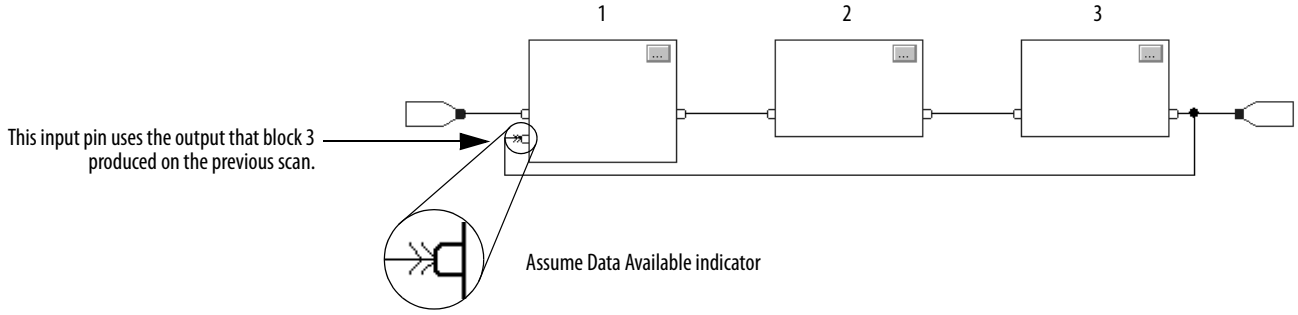
To create a feedback loop around a block, wire an output pin of the block to an input pin of the same block. The following example is okay. The loop contains only a single block, so execution order does not matter.



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.

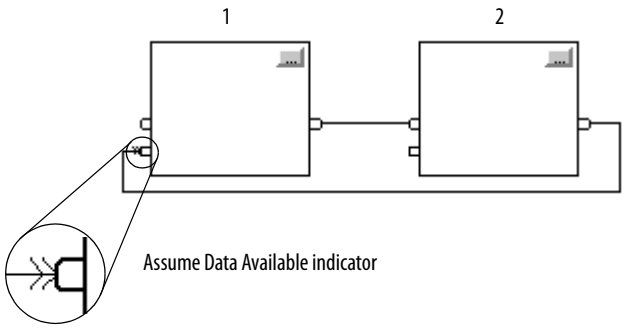
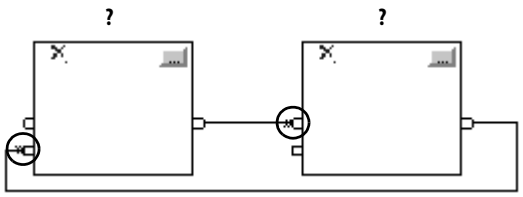


To identify which block to execute first, mark the input wire that creates the loop (the feedback wire) with the *Assume Data Available* indicator. In the following example, block 1 uses the output from block 3 that was produced in the previous execution of the routine.



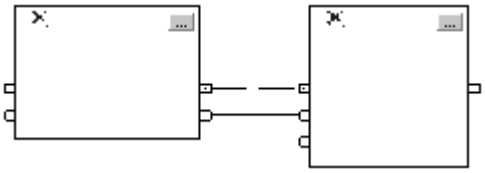
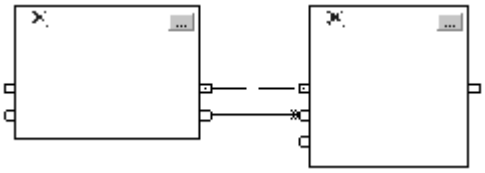
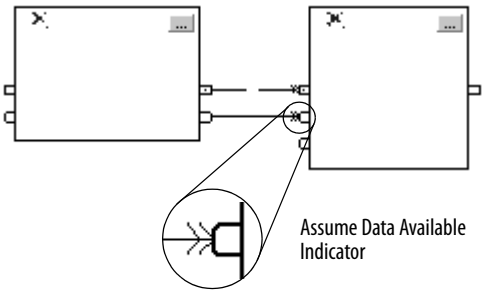
The *Assume Data Available* indicator defines the data flow within the loop. The arrow indicates that the data serves as input to the first block in the loop.

Do **not** mark all the wires of a loop with the Assume Data Available indicator.

This is okay	This is NOT okay
 The Assume Data Available indicator defines the data flow within the loop.	 The controller cannot resolve the loop because all the wires use the Assume Data Available indicator.

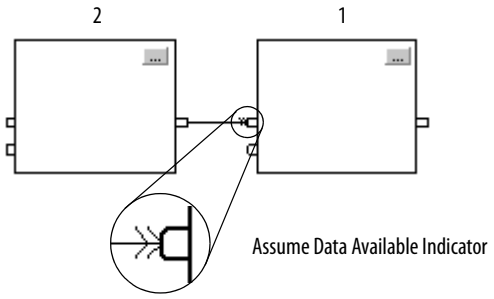
Resolve Data Flow between Two Blocks

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 okay	This is not okay
 Neither wire uses the Assume Data Available indicator.	 One wire uses the Assume Data Available indicator while the other wire does not.
 Both wires use the Assume Data Available indicator.	

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

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 ±NAN, or ±INF values when an overflow occurs. Each instruction has one of these responses to an overflow condition.

Response 1 Blocks execute their algorithm and check the result for ±NAN or ±INF. If ±NAN or ±INF, the block outputs ±NAN or ±INF.	**Response 2** Blocks with output limiting execute their algorithm and check the result for ±NAN or ±INF. The output limits are defined by the HighLimit and LowLimit input parameters. If ±INF, the block outputs a limited result. If ±NAN, the output limits are not used and the block outputs ±NAN.	**Response 3** The overflow condition does not apply. These instructions typically have a boolean output.
ALMNTCH DEDTPMUL DERVPOSP ESELRLIM FGENRMPS HPFSCRV LDL2SEL LDLGSNEG LPFSRTP MAVESSUM MAXCTOT MINCUPDN MSTD MUX	HLL INTG PI PIDE SCL SOC	BANDOSRI BNOTRESD BORRTOR BXORSETD CUTDTOFR D2SDTONR D3SD DFF JKFF OSFI

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	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:	
	If the instruction executes in a	**Then DeltaT equals**
	Periodic task	Period of the task
	Event or continuous task	Elapsed time since the previous execution 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.
	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.	
Oversample	In oversample mode, the delta time (DeltaT) used by the instruction is the value written into the OversampleDT parameter of the instruction. If the process input has a time stamp value, use the real time sampling mode instead. 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. 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.	
Real time sampling	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 process input has a time stamp associated with its updates and you need precise coordination. 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. 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.	

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 following happens:

- 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

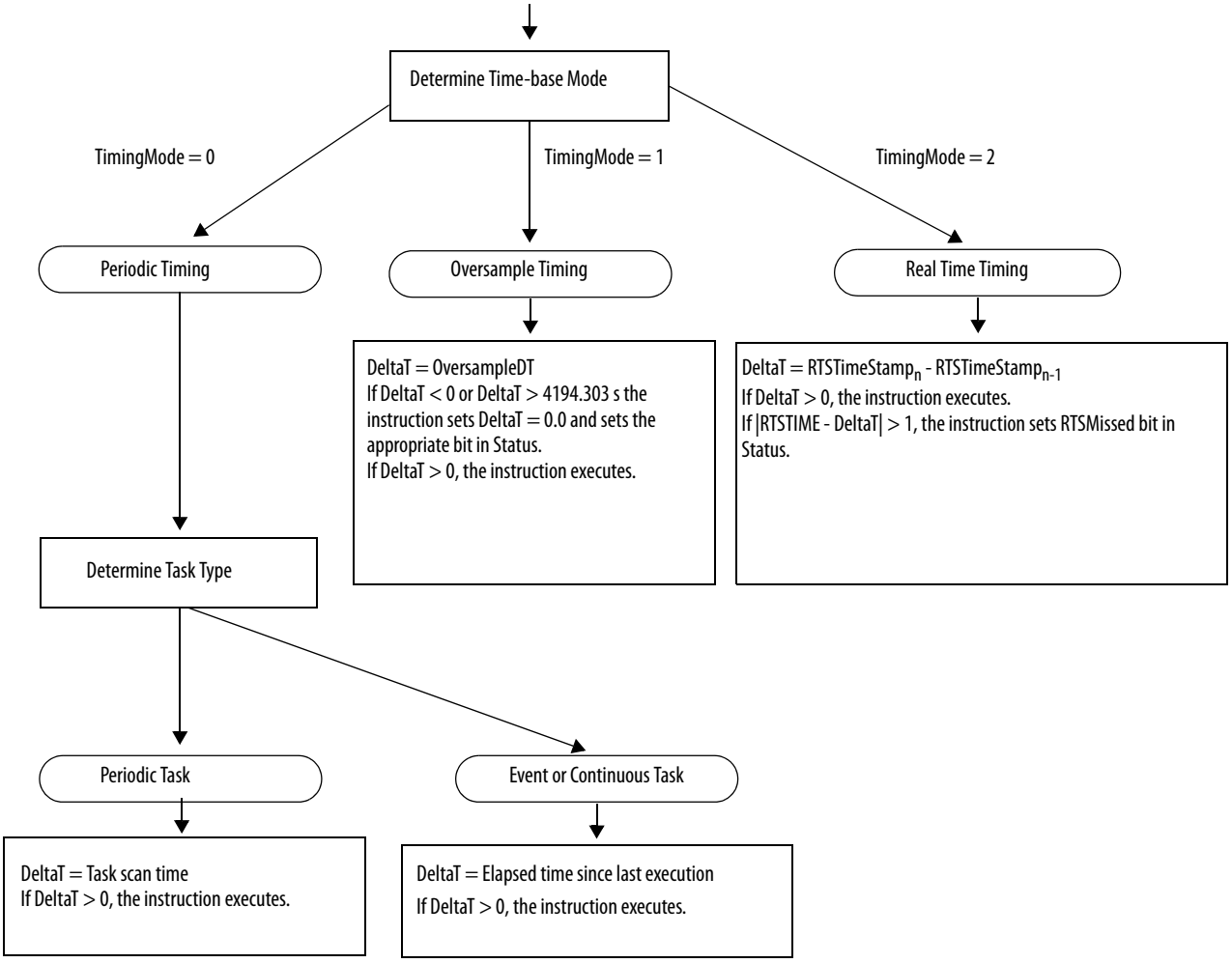
Input Parameter	Data Type	Description
TimingMode	DINT	Selects timing execution mode. **Value:** **Description:** 0 Periodic mode 1 Oversample mode 2 Real time sampling mode Valid = 0…2 Default = 0 When TimingMode = 0 and task is periodic, periodic timing is enabled and DeltaT is set to the task scan 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. When TimingMode = 1, oversample timing is enabled and DeltaT is set to the value of the OversampleDT parameter. 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. If TimingMode invalid, the instruction sets the appropriate bit in Status.
OversampleDT	REAL	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. Valid = 0…4194.303 seconds Default = 0.0
RTSTime	DINT	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. Valid = 1…32,767ms Default = 1
RTSTimeStamp	DINT	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. Valid =1…32,767ms (wraps from 32767 to 0) 1 count = 1 millisecond Default = 0

Output Parameters

Output Parameter	Data Type	Description
DeltaT	REAL	Elapsed time between updates. This is the elapsed time in seconds used by the control algorithm to calculate the process output. Periodic: DeltaT = task scan rate if task is Periodic task, DeltaT = elapsed time since previous instruction execution if task is Event or Continuous task Oversample: DeltaT = OversampleDT Real Time Sampling: DeltaT = ($RTSTimeStamp_n$ - $RTSTimeStamp_{n-1}$)
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 | DeltaT - RTSTime | > 1 (.001 second).
RTSTimeInv (Status.29)	BOOL	Invalid RTSTime value.
RTSTimeStampInv (Status.30)	BOOL	Invalid RTSTimeStamp 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 the following:

- 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.

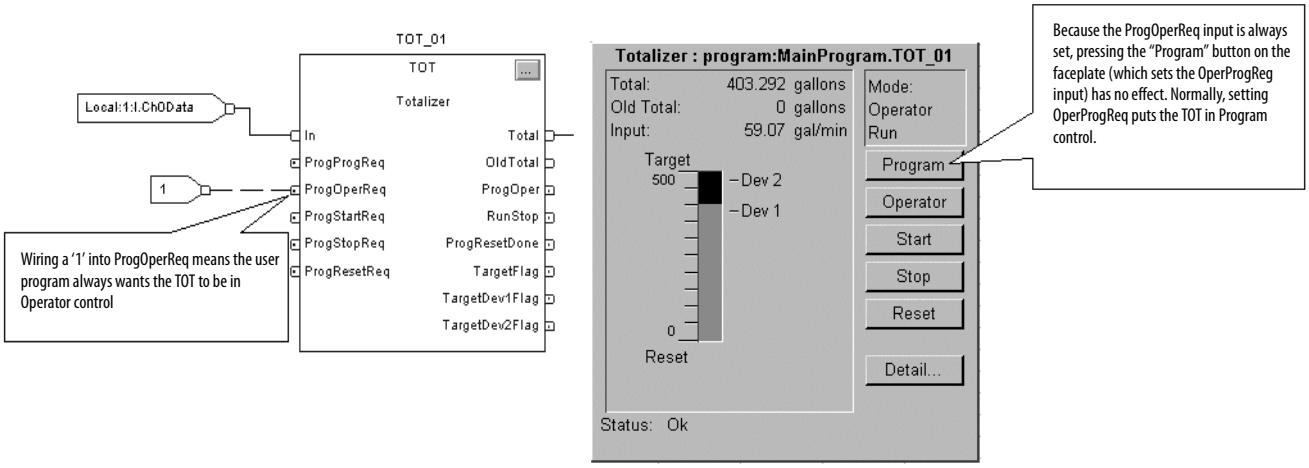
Input	Description
.ProgProgReq	A program request to go to Program control.
.ProgOperReq	A program request to go to Operator control.
.OperProgReq	An operator request to go to Program control.
.OperOperReq	An operator request to go to Operator control.

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.

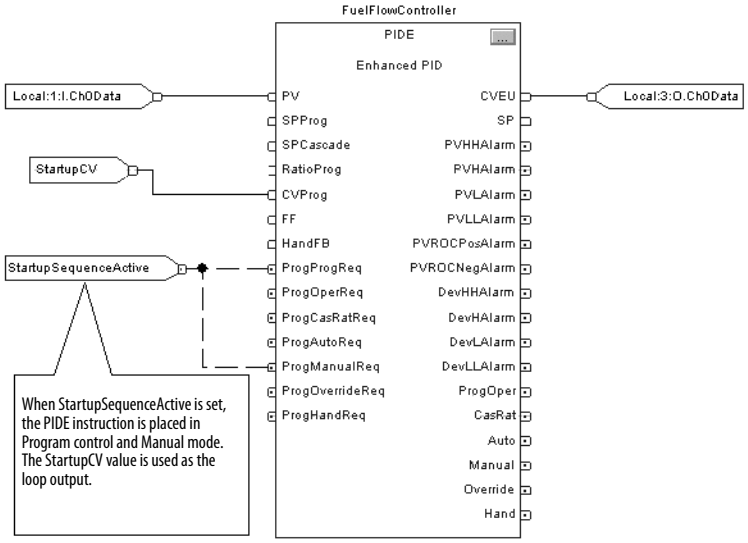


Likewise, constantly setting the ProgProgReq can 'lock' the instruction into Program control. This is useful for automatic startup sequences when you want the program to control the action of the instruction without worrying about an operator inadvertently taking control of the instruction.

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 push button 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 push button is pressed.

When the TIC101AutoReq push button 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 executes.

Structured Text Syntax

Structured text is a textual programming language that uses statements to define what to execute.

- Structured text is not case sensitive.
- Use tabs and carriage returns (separate lines) to make your structured text easier to read. They have no effect on the execution of the structured text.

Structured text is not case sensitive. Structured text can contain these components.

Term	Definition		Examples
Assignment (see page 670)	Use an assignment statement to assign values to tags. The := operator is the assignment operator. Terminate the assignment with a semi colon ';.'		tag := expression;
Expression (see page 673)	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).	value1
	Immediate	A constant value.	4
	Operators	A symbol or mnemonic that specifies an operation within an expression.	tag1 + tag2 tag1 >= value1
	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 be used only in expressions. Instructions cannot be used in expressions.	function(tag1)

Term	Definition	Examples

Instruction
(see page 679)
	An instruction is a standalone statement.
	An instruction uses parenthesis to contain its operands.
	Depending on the instruction, there can be zero, one, or multiple operands.
	When executed, an instruction yields one or more values that are part of a data structure.
	Terminate the instruction with a semi colon(;).
	Even though their syntax is similar, instructions differ from functions in that instructions cannot be used in expressions. Functions can be used only in expressions.
		instruction();
		
		instruction(operand);
		
		instruction(operand1, operand2,operand3);

Construct (see page 680)	A conditional statement used to trigger structured text code (that is, other statements). Terminate the construct with a semi colon (;).	IF...THEN CASE FOR...DO WHILE...DO REPEAT...UNTIL EXIT
Comment (see page 696)	Text that explains or clarifies what a section of structured text does. • 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.	//comment (*start of comment . . . end of comment*) /*start of comment . . . end of comment*/

Assignments

Use an assignment to change the value stored within a tag. An assignment has this syntax:

tag := expression ;

where:

Component	Description	
`Tag`	Represents the tag that is getting the new value The tag must be a BOOL, SINT, INT, DINT, or REAL	
:=	Is the assignment symbol	
`Expression`	Represents the new value to assign to the tag	
	If tag is this data type	**Use this type of expression**
	BOOL	BOOL expression
	SINT INT DINT REAL	Numeric expression
;	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 Expressions on page 673.

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 does the following:

- 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	
`Tag`	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	
`Expression`	Represents the new value to assign to the tag	
	If tag is this data type	**Use this type of expression**
	BOOL	BOOL expression
	SINT INT DINT REAL	Numeric expression
;	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.

The table shows some examples.

This Is Okay	This Is Not Okay
string1.DATA[0]:= 65;	string1.DATA[0] := A;
string1.DATA[0]:= string2.DATA[0];	string1 := string2;

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 678.

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 table to choose operators for your expressions.

If you want to	Then
Calculate an arithmetic value	Use Arithmetic Operators and Functions on page 674.
Compare two values or strings	Use Relational Operators on page 675.
Check if conditions are true or false	Use Logical Operators on page 677.
Compare the bits within values	Use Bitwise Operators on page 678.

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	ABS (*numeric_expression*)	DINT, REAL
Arc cosine	ACOS (*numeric_expression*)	REAL
Arc sine	ASIN (*numeric_expression*)	REAL
Arc tangent	ATAN (*numeric_expression*)	REAL
Cosine	COS (*numeric_expression*)	REAL
Radians to degrees	DEG (*numeric_expression)*	DINT, REAL
Natural log	LN (*numeric_expression*)	REAL
Log base 10	LOG (*numeric_expression*)	REAL
Degrees to radians	RAD (*numeric_expression*)	DINT, REAL
Sine	SIN (*numeric_expression*)	REAL
Square root	SQRT (*numeric_expression*)	DINT, REAL
Tangent	TAN (*numeric_expression*)	REAL
Truncate	TRUNC (*numeric_expression*)	DINT, REAL

The table shows some examples.

Use this format	Example	
	For this situation	**Write**
value1 operator value2	If gain_4 and gain_4_adj are DINT tags and your specification says: 'Add 15 to gain_4 and store the result in gain_4_adj'"	gain_4_adj := gain_4+15;
operator value1	If alarm and high_alarm are DINT tags and your specification says: 'Negate high_alarm and store the result in alarm.'	alarm:= -high_alarm;
function(numeric_expression)	If overtravel and overtravel_POS are DINT tags and your specification says: 'Calculate the absolute value of overtravel and store the result in overtravel_POS.'	overtravel_POS := ABS(overtravel);
value1 operator (function((value2+value3)/2)	If adjustment and position are DINT tags and sensor1 and sensor2 are REAL tags and your specification says: 'Find the absolute value of the average of sensor1 and sensor2, add the adjustment, and store the result in position.'	position := adjustment + ABS((sensor1 + sensor2)/2);

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 these 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

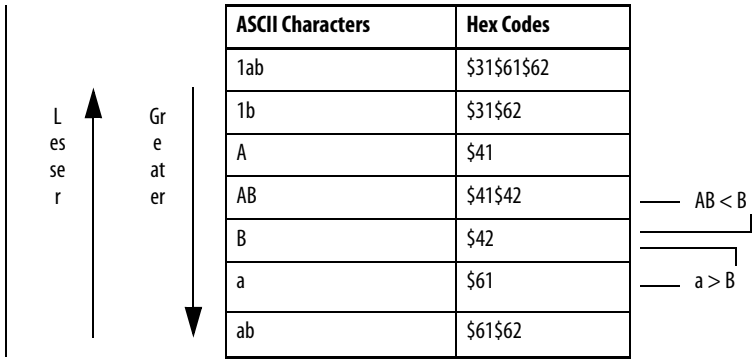
The table shows some examples.

Use this format	Example	
	For this situation	**Write**
value1 operator value2	If temp is a DINT tag and your specification says: 'If temp is less than 100· then…'	IF temp<100 THEN...
stringtag1 operator stringtag2	If bar_code and dest are string tags and your specification says: 'If bar_code equals dest then…'	IF bar_code=dest THEN...
char1 operator char2 To enter an ASCII character directly into the expression, enter the decimal value of the character.	If bar_code is a string tag and your specification says: 'If bar_code.DATA[0] equals 'A' then…'	IF bar_code.DATA[0]=65 THEN...
bool_tag := *bool_expressions*	If count and length are DINT tags, done is a BOOL tag, and your specification says: 'If count is greater than or equal to length, 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
1ab	$31$61$62
1b	$31$62
A	$41
AB	$41$42
B	$42
a	$61
ab	$61$62

- Strings are equal if their characters match.
- Characters are case sensitive. Uppercase ‘A’ ($41) is **not** equal to lowercase ‘a’ ($61).

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 these 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

The table shows some examples.

Use this format	Example	
	For this situation	**Write**
BOOLtag	If photoeye is a BOOL tag and your specification says: 'If photoeye is on then…'	IF photoeye THEN...
NOT *BOOLtag*	If photoeye is a BOOL tag and your specification says: 'If photoeye is off then…'	IF NOT photoeye THEN...
expression1 & *expression2*	If photoeye is a BOOL tag, temp is a DINT tag, and your specification says: 'If photoeye is on and temp is less than 100· then…'	IF photoeye & (temp<100) THEN...
expression1 OR *expression2*	If photoeye is a BOOL tag, temp is a DINT tag, and your specification says: 'If photoeye is on or temp is less than 100· then…'	IF photoeye OR (temp<100) THEN...
expression1 XOR *expression2*	If photoeye1 and photoeye2 are BOOL tags and your specification says: 'If: • photoeye1 is on while photoeye 2 is off • photoeye1 is off while photoeye 2 is on then…'	IF photoeye1 XOR photoeye2 THEN...
BOOLtag := *expression1* & *expression2*	If photoeye1 and photoeye2 are BOOL tags, open is a BOOL tag, and your specification says: 'If photoeye1 and photoeye2 are both on, set open 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

This is an example.

Use this format	Example	
	For this situation	**Write**
value1 operator value2	If input1, input2, and result1 are DINT tags and your specification says: 'Calculate the bitwise result of input1 and input2. Store the result in result1.'	result1 := input1 AND input2;

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 on page 29 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 alone or nested within other constructs.

If you want to	Use this construct	Available in these languages	Page
Do something if or when specific conditions occur	IF...THEN	Structured text	681
Select what to do based on a numerical value	CASE...OF	Structured text	684
Do something a specific number of times before doing anything else	FOR...DO	Structured text	687
Keep doing something as long as certain conditions are true	WHILE...DO	Structured text	690
Keep doing something until a condition is true	REPEAT...UNTIL	Structured text	693

Some Key Words are Reserved

These constructs are not available:

- GOTO
- REPEAT

Logix Designer application will not let you use them as tag names or constructs.

IF...THEN

Use IF...THEN to do something if or when specific conditions occur.

Operands:

Structured Text

IF *bool_expression* THEN

<statement>;

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 described in the table.

```
IF bool_expression1 THEN
      <statement >;          <-- Statements to execute when bool_expression1 is true
      .
      .
      .
optional
ELSIF bool_expression2 THEN
      <statement>;           <-- Statements to execute when bool_expression2 is true
      .
      .
      .
optional
ELSE
      <statement>;           <-- Statements to execute when both expressions are false
      .
      .
      .
END_IF;
```

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 table summarizes different combinations of IF, THEN, ELSIF, and ELSE.

If you want to	And	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

Arithmetic Status Flags

Not affected

Fault Conditions:

None

Example 1: IF…THEN

If you want this	Enter this structured text
IF rejects > 3 then	IF rejects > 3 THEN
conveyor = off (0)	conveyor := 0;
alarm = on (1)	alarm := 1;
	END_IF;

Example 2: IF…THEN…ELSE

If you want this	Enter this structured text
If conveyor direction contact = forward (1) then	IF conveyor_direction THEN
light = off	light := 0;
Otherwise light = on	ELSE
	light [:=] 1;
	END_IF;

The [:=] tells the controller to clear *light* whenever the controller does the following:

- 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	IF Sugar.Low & Sugar.High THEN
inlet valve = open (on)	Sugar.Inlet [:=] 1;
Until sugar high limit switch = high (off)	ELSIF NOT(Sugar.High) THEN
	Sugar.Inlet := 0;
	END_IF;

The [:=] tells the controller to clear *Sugar.Inlet* whenever the controller does the following:

- 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	IF tank.temp > 200 THEN
then pump = slow	pump.fast :=1; pump.slow :=0; pump.off :=0;
If tank temperature > 200	ELSIF tank.temp > 100 THEN
then pump = fast	pump.fast :=0; pump.slow :=1; pump.off :=0;
Otherwise pump = off	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:

Structured Text

CASE *numeric_expression* OF

selector1: *statement*;

selectorN: *statement*;

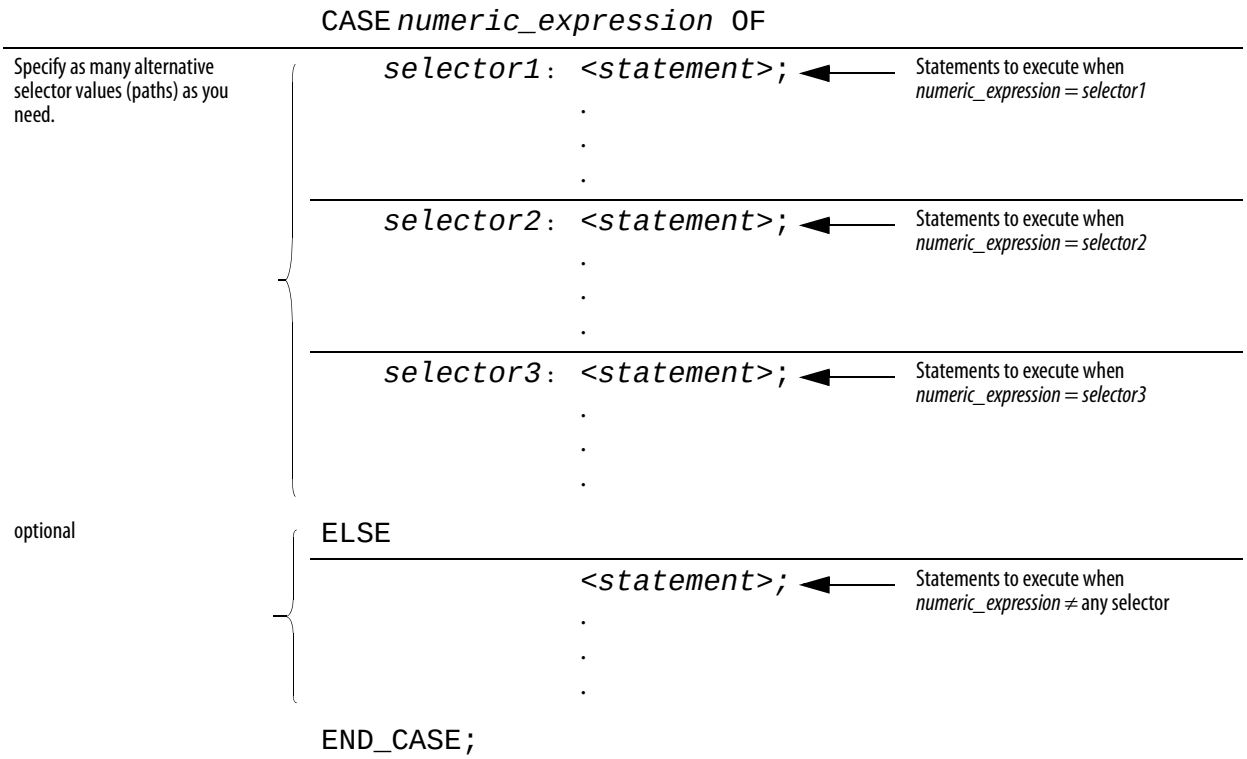
ELSE

Operand	Type	Format	Enter
`Numeric_expression`	SINT INT DINT REAL	Tag expression	Tag or expression that evaluates to a number (numeric expression)
`Selector`	SINT INT DINT REAL	Immediate	Same type as `numeric_expression`

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 described in the table.



See the table on page 686 for valid selector values.

These are the syntax for entering the selector values.

When selector is	Enter
One value	*value*: *statement*
Multiple, distinct values	*value1*, *value2*, *valueN* : <*statement*> Use a comma (,) to separate each value.
A range of values	*value1*..*valueN* : <*statement*> Use two periods (..) to identify the range.
Distinct values plus a range of values	*valuea, valueb, value1*..*valueN* : <*statement*>

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.

Arithmetic Status Flags: Not affected

Fault Conditions: None

Example

If you want this	Enter this structured text
If recipe number = 1 then Ingredient A outlet 1 = open (1) Ingredient B outlet 4 = open (1)	CASE recipe_number OF
	1: Ingredient_A.Outlet_1 :=1; Ingredient_B.Outlet_4 :=1;
If recipe number = 2 or 3 then Ingredient A outlet 4 = open (1) Ingredient B outlet 2 = open (1)	2,3: Ingredient_A.Outlet_4 :=1; Ingredient_B.Outlet_2 :=1;
If recipe number = 4, 5, 6, or 7 then Ingredient A outlet 4 = open (1) Ingredient B outlet 2 = open (1)	4…7: Ingredient_A.Outlet_4 :=1; Ingredient_B.Outlet_2 :=1;
If recipe number = 8, 11, 12, or 13 then Ingredient A outlet 1 = open (1) Ingredient B outlet 4 = open (1)	8,11…13 Ingredient_A.Outlet_1 :=1; Ingredient_B.Outlet_4 :=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 does the following:

- 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:

Structured Text

FOR *count*:= *initial_value* TO *final_value* BY *increment* DO

<*statement*>;

END_FOR;

Operand	Type	Format	Description
count	SINT INT DINT	Tag	Tag to store count position as the FOR…DO executes
initial_ value	SINT INT DINT	Tag expression Immediate	Must evaluate to a number Specifies initial value for count
final_ value	SINT INT DINT	Tag expression Immediate	Specifies final value for count, which determines when to exit the loop
increment	SINT INT DINT	Tag expression Immediate	(*Optional*) amount to increment count each time through the loop If you don't specify an increment, the count increments by 1.

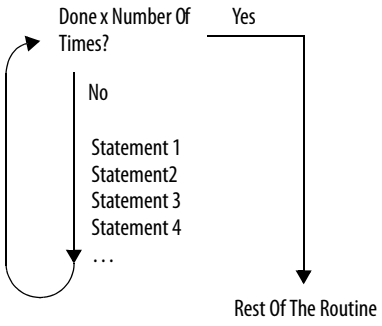
IMPORTANT Make sure that you **do not** iterate within the loop too many times in a single scan.

- 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.

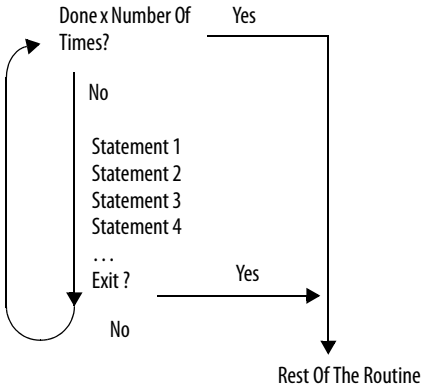
Description: The syntax is described in the table.

		FOR *count* := *initial_value*	
		TO *final_value*	
Optional	{	BY *increment*	If you don't specify an increment, the loop increments by 1.
		DO	
		<*statement*>;	
Optional	{	IF *bool_expression* THEN	
		EXIT;	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.
		END_IF;	
		END_FOR;	

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.

Arithmetic Status Flags:

Not affected

Fault Conditions:

A major fault will occur if	Fault type	Fault code
The construct loops too long.	6	1

Example 1:

If you want this	Enter this structured text

Clear bits 0…31 in an array of BOOLs:
1. Initialize the *subscript* tag to 0.
2. Clear *array[subscript]* . For example, when *subscript* = 5, clear *array[5]*.
3. Add 1 to *subscript*.
4. If *subscript* 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

A user-defined data type (structure) stores the following information about an item in your inventory:
• Barcode ID of the item (string data type)
• Quantity in stock of the item (DINT data type)
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.
1. Get the size (number of items) of the Inventory array and store the result in Inventory_Items (DINT tag).
2. Initialize the position tag to 0.
3. If Barcode matches the ID of an item in the array, then:
a. Set the Quantity tag = Inventory[position].Qty. This produces the quantity in stock of the item.
b. Stop.
Barcode is a string tag that stores the bar code of the item for which you are searching. For example, when position = 5, compare Barcode to Inventory[5].ID.
4. Add 1 to position.
5. If position is ≤ to (Inventory_Items -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.
	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;

WHILE...DO

Use the WHILE...DO loop to keep doing something as long as certain conditions are true.

Operands:

WHILE *bool_expression* DO

<*statement*>;

Structured Text

Operand	Type	Format	Enter
Bool_ expression	BOOL	Tag expression	BOOL tag or expression that evaluates to a BOOL value

IMPORTANT

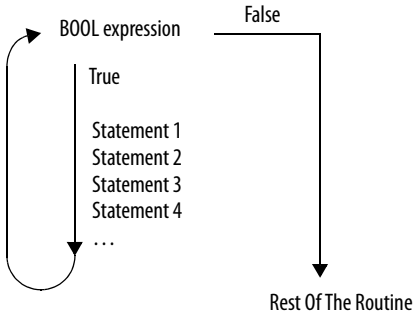
Make sure that you do not iterate within the loop too many times in a single scan.

- 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.

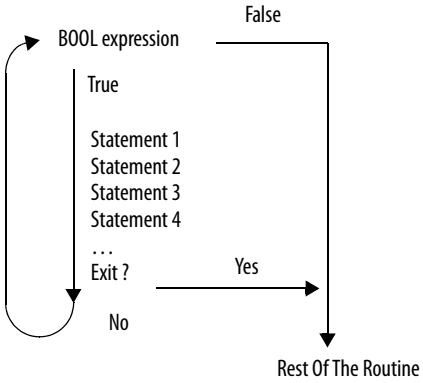
Description: The syntax is described in the table.

WHILE *bool_expression1* DO		
<*statement*>;	←	Statements to execute while *bool_expression1* is true
Optional		
IF *bool_expression2* THEN		
EXIT;	←	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.
END_IF;		
END_WHILE;		

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.

Arithmetic Status Flags:

Not affected

Fault Conditions:

A major fault will occur if	Fault type	Fault code
The construct loops too long	6	1

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.
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.
	pos := 0;
	While ((pos <= 100) & structarray[pos].value <> targetvalue)) do
	  pos := pos + 2;
	  String_tag.DATA[pos] := SINT_array[pos];
	end_while;

Example 2:

If you want this	Enter this structured text

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.
1. Initialize Element_number to 0.
2. Count the number of elements in SINT_array (array that contains the ASCII characters) and store the result in SINT_array_size (DINT tag).
3. If the character at SINT_array[element_number] = 13 (decimal value of the carriage return), then stop.
4. Set String_tag[element_number] = the character at SINT_array[element_number].
5. Add 1 to element_number. This lets the controller check the next character in SINT_array.
6. Set the Length member of String_tag = element_number. (This records the number of characters in String_tag so far.)
7. If element_number = SINT_array_size, then stop. (You are at the end of the array and it does not contain a carriage return.)
8. Go to step 3.
	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;

REPEAT...UNTIL

Use the REPEAT...UNTIL loop to keep doing something until conditions are true.

Operands:

Structured Text

REPEAT

<*statement*>;

Operand	Type	Format	Enter
bool_ expression	BOOL	Tag expression	BOOL tag or expression that evaluates to a BOOL value (BOOL expression)

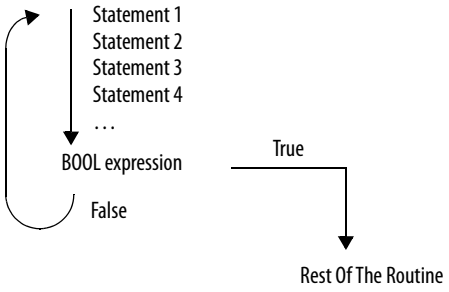
IMPORTANT **Make sure that you do not iterate within the loop too many times in a single scan.**

- 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.

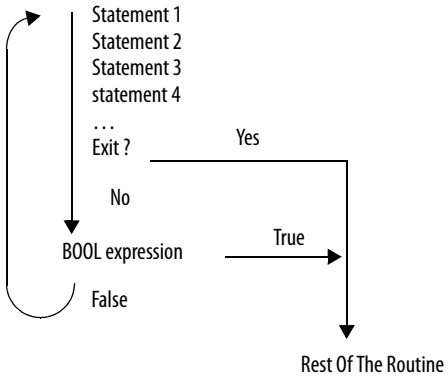
Description: The syntax is described in the table.

REPEAT		
<statement>;	←	statements to execute while *bool_expression1* is false
Optional **IF *bool_expression2* THEN**		
EXIT;	←	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.
END_IF;		
UNTIL *bool_expression1*		
END_REPEAT;		

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.

Arithmetic Status Flags:

Not affected

Fault Conditions:

A major fault will occur if	Fault type	Fault code
The construct loops too long	6	1

Example 1:

If you want this	Enter this structured text

The REPEAT...UNTIL loop executes the statements in the construct and then determines if the conditions are true before executing the statements again. 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.	pos := -1;
	REPEAT
	pos := pos + 2;
	UNTIL ((pos = 101) OR (structarray[pos].value = targetvalue))
	end_repeat;

Example 2:

If you want this	Enter this structured text

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.
1. Initialize Element_number to 0.
2. Count the number of elements in SINT_array (array that contains the ASCII characters) and store the result in SINT_array_size (DINT tag).
3. Set String_tag[element_number] = the character at SINT_array[element_number].
4. Add 1 to element_number. This lets the controller check the next character in SINT_array.
5. Set the Length member of String_tag = element_number. (This records the number of characters in String_tag so far.)
6. If element_number = SINT_array_size, then stop. (You are at the end of the array and it does not contain a carriage return.)
7. If the character at SINT_array[element_number] = 13 (decimal value of the carriage return), then stop.
Otherwise, go to step 3.
	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;

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.

The table describes how to add comments to your structured text.

To add a comment	Use one of these formats
On a single line	`//comment`

At the end of a line of structured text	`(*comment*)`
	
	`/*comment*/`

Within a line of structured text	`(*comment*)`
	
	`/*comment*/`

That spans more than one line	`(*start of comment . . . end of comment*)`
	
	`/*start of comment . . . end of comment*/`

The table shows some examples.

Format	Example

//*comment*	At the beginning of a line
	//Check conveyor belt direction
	IF conveyor_direction THEN...
	
	At the end of a line
	ELSE //If conveyor isn't moving, set alarm light
	light := 1;
	END_IF;

(**comment**)	Sugar.Inlet[:=]1;(*open the inlet*) IF Sugar.Low (*low level LS*)& Sugar.High (*high level LS*)THEN... (*Controls the speed of the recirculation pump. The speed depends on the temperature in the tank.*) IF tank.temp > 200 THEN...
/**comment*/	Sugar.Inlet:=0;/*close the inlet*/ IF bar_code=65 /*A*/ THEN... /*Gets the number of elements in the Inventory array and stores the value in the Inventory_Items tag*/ SIZE(Inventory,0,Inventory_Items);

A

B

C

D

E

F

G

I

J

L

M

N

O

P

R

S

T

U

W

X

Notes:

Notes:

Notes:

Rockwell Automation Support

Rockwell Automation provides technical information on the Web to assist you in using its products. At http://www.rockwellautomation.com/support, you can find technical manuals, 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. You can also visit our Knowledgebase at http://www.rockwellautomation.com/knowledgebase for FAQs, technical information, support chat and forums, software updates, and to sign up for product notification updates.

For an additional level of technical phone support for installation, configuration, and troubleshooting, we offer TechConnect^{SM} support programs. For more information, contact your local distributor or Rockwell Automation representative, or visit http://www.rockwellautomation.com/support/.

Installation Assistance

If you experience a problem within the first 24 hours of installation, review the information that is contained in this manual. You can contact Customer Support for initial help in getting your product up and running.

United States or Canada	1.440.646.3434
Outside United States or Canada	Use the Worldwide Locator at http://www.rockwellautomation.com/support/americas/phone_en.html, or contact your local Rockwell Automation representative.

New Product Satisfaction Return

Rockwell Automation tests all of its 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, follow these procedures.

United States	Contact your distributor. You must provide a Customer Support case number (call the phone number above to obtain one) to your distributor to complete the return process.
Outside United States	Please contact your local Rockwell Automation representative for the return procedure.

Documentation Feedback

Your comments will help us serve your documentation needs better. If you have any suggestions on how to improve this document, complete this form, publication RA-DU002, available at http://www.rockwellautomation.com/literature/.

Rockwell Otomasyon Ticaret A.Ş., Kar Plaza İş Merkezi E Blok Kat:6 34752 İçerenköy, İstanbul, Tel: +90 (216) 5698400

www.rockwellautomation.com

Power, Control and Information Solutions Headquarters

Americas: Rockwell Automation, 1201 South Second Street, Milwaukee, WI 53204-2496 USA, Tel: (1) 414.382.2000, Fax: (1) 414.382.4444
Europe/Middle East/Africa: Rockwell Automation NV, Pegasus Park, De Kleetlaan 12a, 1831 Diegem, Belgium, Tel: (32) 2 663 0600, Fax: (32) 2 663 0640
Asia Pacific: Rockwell Automation, Level 14, Core F, Cyberport 3, 100 Cyberport Road, Hong Kong, Tel: (852) 2887 4788, Fax: (852) 2508 1846

Publication 1756-RM003O-EN-P - November 2012
Supersedes Publication 1756-RM003N-EN-P - October 2011