

***Allen-Bradley***

# **Logix5000 Controllers General Instructions**

**Reference Manual**

**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://literature.rockwellautomation.com>) 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.

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

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

<b>WARNING</b> 	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.
<b>IMPORTANT</b>	Identifies information that is critical for successful application and understanding of the product.
<b>ATTENTION</b> 	Identifies information about practices or circumstances that can lead to personal injury or death, property damage, or economic loss. Attentions help you to identify a hazard, avoid a hazard, and recognize the consequences.
<b>SHOCK HAZARD</b> 	Labels may be on or inside the equipment, for example, a drive or motor, to alert people that dangerous voltage may be present.
<b>BURN HAZARD</b> 	Labels may be on or inside the equipment, for example, a drive or motor, to alert people that surfaces may be dangerous temperatures.

Allen-Bradley, ControlLogix, FactoryTalk, Logix5000, RSLogix, RSLogix 5000, Rockwell Automation, RSNetWorx, and RSLinx are trademarks of Rockwell Automation, Inc.

Trademarks not belonging to Rockwell Automation are property of their respective companies.

cuu duong than cong. com

cuu duong than cong. com

cuu duong than cong. com

cuu duong than cong. com

## Table of Contents

<b>Summary of Changes</b>	Introduction . . . . .	15
	Updated Information. . . . .	15
<b>Preface</b>	Introduction . . . . .	17
	Who Should Use This Manual . . . . .	17
	Purpose of This Manual. . . . .	18
	Common Information for All Instructions . . . . .	19
	Conventions and Related Terms. . . . .	19
	Set and clear . . . . .	19
	Relay ladder rung condition . . . . .	20
	Function block states . . . . .	21
<b>Instruction Locator</b>	Where to Find an Instruction . . . . .	23
	<b>Chapter 1</b>	
<b>Digital Alarm Instruction (ALMD)</b>	Introduction . . . . .	31
	About Operator Parameters . . . . .	31
	Using the ALMD Instruction to Subscribe to and Display Alarms. . . . .	31
	Digital Alarm Operands. . . . .	32
	Ladder Logic Operands . . . . .	32
	Structured Text Operands. . . . .	32
	Function Block Operands. . . . .	33
	Structure Definition for ALARM_DIGITAL Tag . . . . .	34
	Input Parameters . . . . .	34
	Output Parameters . . . . .	37
	Example . . . . .	40
	Ladder Logic . . . . .	40
	Function Block . . . . .	41
	Execution . . . . .	42
	Ladder Logic . . . . .	42
	Function Block . . . . .	42
	Digital State Timing Diagrams . . . . .	43
	Alarm Acknowledge Required and Latched. . . . .	43
	Alarm Acknowledge Required and Not Latched. . . . .	44
	Alarm Acknowledge Not Required and Latched. . . . .	45
	Alarm Acknowledge Not Required and Not Latched . . . .	46
	<b>Chapter 2</b>	
<b>Analog Alarm Instruction (ALMA)</b>	Introduction . . . . .	47
	About Operator Parameters . . . . .	47
	Using the ALMA Instruction to Subscribe to and Display Alarms. . . . .	47
	Analog Alarm Operands . . . . .	48
	Ladder Logic Operands . . . . .	48
	Structured Text Operands. . . . .	48
	Function Block Operands. . . . .	49
	Structure Definition For ALARM_ANALOG Tag. . . . .	50
	Input Parameters . . . . .	50

Output Parameters . . . . .	55
Example . . . . .	62
Ladder Logic . . . . .	62
Structured Text . . . . .	62
Function Block . . . . .	63
Execution . . . . .	64
Ladder Logic . . . . .	64
Function Block . . . . .	64
Analog State Timing Diagrams . . . . .	65
Alarm Level Condition Acknowledge Required . . . . .	65
Alarm Level Condition Acknowledge Not Required . . . . .	66
Alarm Rate of Change Acknowledge Required . . . . .	67
Alarm Rate of Change Acknowledge Not Required . . . . .	68

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

#### **Chapter 3**

Introduction . . . . .	69
Examine If Closed (XIC) . . . . .	70
Examine If Open (XIO) . . . . .	72
Output Energize (OTE) . . . . .	74
Output Latch (OTL) . . . . .	76
Output Unlatch (OTU) . . . . .	78
One Shot (ONS) . . . . .	80
One Shot Rising (OSR) . . . . .	83
One Shot Falling (OSF) . . . . .	86
One Shot Rising with Input (OSRI) . . . . .	89
One Shot Falling with Input (OSFI) . . . . .	92

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

#### **Chapter 4**

Introduction . . . . .	95
Timer On Delay (TON) . . . . .	96
Timer Off Delay (TOF) . . . . .	100
Retentive Timer On (RTO) . . . . .	105
Timer On Delay with Reset (TONR) . . . . .	110
Timer Off Delay with Reset (TOFR) . . . . .	114
Retentive Timer On with Reset (RTOR) . . . . .	118
Count Up (CTU) . . . . .	123
Count Down (CTD) . . . . .	127
Count Up/Down (CTUD) . . . . .	131
Reset (RES) . . . . .	136

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

#### **Chapter 5**

Introduction . . . . .	139
Message (MSG) . . . . .	140
MSG Error Codes . . . . .	148
Error Codes . . . . .	148

Extended Error Codes . . . . .	150
PLC and SLC Error Codes (.ERR). . . . .	152
Block-Transfer Error Codes . . . . .	154
Specify the Configuration Details . . . . .	155
Specify CIP Data Table Read and Write messages . . . . .	156
Reconfigure an I/O module . . . . .	157
Specify CIP Generic messages . . . . .	158
Specify PLC-5 messages . . . . .	159
Specify SLC messages. . . . .	161
Specify block-transfer messages . . . . .	161
Specify PLC-3 messages . . . . .	162
Specify PLC-2 messages . . . . .	163
MSG Configuration Examples . . . . .	164
Specify the Communication Details . . . . .	165
Specify a path . . . . .	165
For Block Transfers . . . . .	168
Specify a Communication Method Or Module Address . . . . .	169
Choose a cache option. . . . .	170
Guidelines. . . . .	172
Get System Value (GSV) and Set System Value (SSV) . . . . .	173
GSV/SSV Objects. . . . .	176
Access the CONTROLLER object. . . . .	177
Access the CONTROLLERDEVICE object. . . . .	177
Access the CST object . . . . .	181
Access the DF1 object . . . . .	182
Access the FAULTLOG object. . . . .	185
Access The MESSAGE Object . . . . .	186
Access The MODULE Object . . . . .	188
Access The MOTIONGROUP Object. . . . .	189
Access The PROGRAM Object . . . . .	190
Access The Routine object . . . . .	192
Access The SERIALPORT Object. . . . .	192
Access The TASK Object . . . . .	194
Access The WALLCLOCKTIME Object. . . . .	196
GSV/SSV Programming Example . . . . .	197
Get Fault Information. . . . .	197
Set Enable And Disable Flags . . . . .	199
Immediate Output (IOT) . . . . .	200

## Chapter 6

### Compare Instructions

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

Introduction . . . . .	205
Compare (CMP) . . . . .	207
CMP expressions . . . . .	209
Valid operators . . . . .	209
Format Expressions . . . . .	210
Determine The Order of Operation. . . . .	210

Use Strings In an Expression . . . . .	211
Equal to (EQU). . . . .	212
Greater than or Equal to (GEQ). . . . .	216
Greater Than (GRT) . . . . .	220
Less Than or Equal to (LEQ) . . . . .	224
Less Than (LES) . . . . .	228
Limit (LIM) . . . . .	232
Mask Equal to (MEQ) . . . . .	238
Entering an Immediate Mask Value. . . . .	239
Not Equal to (NEQ). . . . .	243

## Chapter 7

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

Introduction . . . . .	247
Compute (CPT). . . . .	249
Valid operators . . . . .	251
Format Expressions . . . . .	251
Determine the order of operation. . . . .	252
Add (ADD). . . . .	253
Subtract (SUB) . . . . .	257
Multiply (MUL) . . . . .	260
Divide (DIV). . . . .	263
Modulo (MOD). . . . .	268
Square Root (SQR) . . . . .	272
Negate (NEG) . . . . .	276
Absolute Value (ABS) . . . . .	279

## Chapter 8

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

Introduction . . . . .	283
Move (MOV). . . . .	285
Masked Move (MVM) . . . . .	287
Enter an immediate mask value . . . . .	288
Masked Move with Target (MVMT) . . . . .	290
Bit Field Distribute (BTD) . . . . .	293
Bit Field Distribute with Target (BTDT) . . . . .	296
Clear (CLR). . . . .	299
Swap Byte (SWPB) . . . . .	301
Bitwise AND (AND) . . . . .	305
Bitwise OR (OR) . . . . .	308
Bitwise Exclusive OR (XOR) . . . . .	311
Bitwise NOT (NOT) . . . . .	315
Boolean AND (BAND) . . . . .	319
Boolean OR (BOR) . . . . .	322
Boolean Exclusive OR (BXOR) . . . . .	325
Boolean NOT (BNOT). . . . .	328



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

### Chapter 9

Introduction . . . . .	331
Selecting Mode of Operation . . . . .	332
All mode . . . . .	332
Numerical mode . . . . .	333
Incremental mode . . . . .	335
File Arithmetic and Logic (FAL) . . . . .	337
FAL Expressions. . . . .	346
Valid operators . . . . .	347
Format Expressions . . . . .	347
Determine the order of operation . . . . .	348
File Search and Compare (FSC) . . . . .	349
FSC expressions. . . . .	354
Valid Operators . . . . .	355
Format Expressions . . . . .	355
Determine the order of operation . . . . .	356
Use Strings In an Expression . . . . .	357
Copy File (COP) Synchronous Copy File (CPS). . . . .	358
File Fill (FLL) . . . . .	364
File Average (AVE) . . . . .	368
File Sort (SRT). . . . .	373
File Standard Deviation (STD) . . . . .	378
Size In Elements (SIZE) . . . . .	384

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

### Chapter 10

Introduction . . . . .	387
Bit Shift Left (BSL). . . . .	388
Bit Shift Right (BSR) . . . . .	392
FIFO Load (FFL) . . . . .	396
FIFO Unload (FFU) . . . . .	402
LIFO Load (LFL) . . . . .	408
LIFO Unload (LFU) . . . . .	414

## Sequencer Instructions (SQI, SQO, SQL)

### Chapter 11

Introduction . . . . .	421
Sequencer Input (SQI). . . . .	422
Enter an Immediate Mask Value . . . . .	423
Use SQI without SQO . . . . .	425
Sequencer Output (SQO) . . . . .	426
Enter an Immediate Mask Value . . . . .	427
Using SQI with SQO . . . . .	429
Resetting the position of SQO . . . . .	429
Sequencer Load (SQL). . . . .	430

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

### Chapter 12

Introduction . . . . .	435
Jump to Label (JMP)	
Label (LBL) . . . . .	436
Jump to Subroutine (JSR)	
Subroutine (SBR) Return (RET) . . . . .	438
Jump to External Routine (JXR) . . . . .	449
Temporary End (TND) . . . . .	452
Master Control Reset (MCR) . . . . .	454
User Interrupt Disable (UID) User Interrupt Enable (UIE) . .	456
Always False Instruction (AFI) . . . . .	458
No Operation (NOP) . . . . .	459
End of Transition (EOT) . . . . .	460
SFC Pause (SFP) . . . . .	462
SFC Reset (SFR) . . . . .	464
Trigger Event Task (EVENT) . . . . .	466
Programmatically Determine if an EVENT Instruction Triggered a Task . . . . .	466

## For/Break Instructions (FOR, FOR...DO, BRK, EXIT, RET)

### Chapter 13

Introduction . . . . .	471
For (FOR) . . . . .	472
Break (BRK) . . . . .	475
Return (RET) . . . . .	476

## Special Instructions (FBC, DDT, DTR, PID)

### Chapter 14

Introduction . . . . .	479
File Bit Comparison (FBC) . . . . .	480
Selecting the Search Mode . . . . .	482
Diagnostic Detect (DDT) . . . . .	488
Selecting the search mode . . . . .	490
Data Transitional (DTR) . . . . .	496
Enter an immediate mask value . . . . .	497
Proportional Integral Derivative (PID) . . . . .	499
Configure a PID Instruction . . . . .	505
Specify tuning . . . . .	506
Specify configuration . . . . .	507
Specifying Alarms . . . . .	507
Specifying Scaling . . . . .	508
Using PID Instructions . . . . .	508
Anti-reset Windup And Bumpless Transfer From Manual To Auto . . . . .	510
PID instruction timing . . . . .	511
Bumpless restart . . . . .	515
Derivative Smoothing . . . . .	516

	Set the deadband . . . . .	517
	Use output limiting . . . . .	517
	Feedforward or output biasing . . . . .	518
	Cascading loops . . . . .	518
	Control a Ratio . . . . .	519
	PID Theory . . . . .	520
	PID Process . . . . .	520
	PID Process With Master/slave Loops . . . . .	520
	<b>Chapter 15</b>	
<b>Trigonometric Instructions</b> <b>(SIN, COS, TAN, ASN, ASIN, ACS,</b> <b>ACOS, ATN, ATAN)</b>	Introduction . . . . .	521
	Sine (SIN) . . . . .	522
	Cosine (COS) . . . . .	525
	Tangent (TAN) . . . . .	529
	Arc Sine (ASN) . . . . .	532
	Arc Cosine (ACS) . . . . .	536
	Arc Tangent (ATN) . . . . .	540
	<b>Chapter 16</b>	
<b>Advanced Math Instructions</b> <b>(LN, LOG, XPY)</b>	Introduction . . . . .	545
	Natural Log (LN) . . . . .	546
	Log Base 10 (LOG) . . . . .	549
	X to the Power of Y (XPY) . . . . .	552
	<b>Chapter 17</b>	
<b>Math Conversion Instructions</b> <b>(DEG, RAD, TOD, FRD, TRN,</b> <b>TRUNC)</b>	Introduction . . . . .	555
	Degrees (DEG) . . . . .	556
	Radians (RAD) . . . . .	559
	Convert to BCD (TOD) . . . . .	562
	Convert to Integer (FRD) . . . . .	565
	Truncate (TRN) . . . . .	567
	<b>Chapter 18</b>	
<b>ASCII Serial Port Instructions</b> <b>(ABL, ACB, ACL, AHL, ARD, ARL,</b> <b>AWA, AWT)</b>	Introduction . . . . .	571
	Instruction Execution . . . . .	572
	ASCII Error Codes . . . . .	574
	String Data Types . . . . .	574
	ASCII Test For Buffer Line (ABL) . . . . .	575
	ASCII Chars in Buffer (ACB) . . . . .	578
	ASCII Clear Buffer (ACL) . . . . .	581
	ASCII Handshake Lines (AHL) . . . . .	583
	ASCII Read (ARD) . . . . .	587
	ASCII Read Line (ARL) . . . . .	591
	ASCII Write Append (AWA) . . . . .	595
	ASCII Write (AWT) . . . . .	600

<b>ASCII String Instructions (CONCAT, DELETE, FIND, INSERT, MID)</b>	<b>Chapter 19</b>	
	Introduction . . . . .	605
	String Data Types. . . . .	607
	String Concatenate (CONCAT). . . . .	608
	String Delete (DELETE). . . . .	610
	Find String (FIND) . . . . .	612
	Insert String (INSERT) . . . . .	614
	Middle String (MID) . . . . .	616
<b>ASCII Conversion Instructions (STOD, STOR, DTOS, RTOS, UPPER, LOWER)</b>	<b>Chapter 20</b>	
	Introduction . . . . .	619
	String Data Types. . . . .	621
	String To DINT (STOD). . . . .	622
	String To REAL (STOR) . . . . .	624
	DINT to String (DTOS) . . . . .	626
	REAL to String (RTOS). . . . .	629
	Upper Case (UPPER). . . . .	631
	Lower Case (LOWER) . . . . .	633
<b>Common Attributes</b>	<b>Appendix A</b>	
	Introduction . . . . .	635
	Immediate Values . . . . .	635
	Data Conversions . . . . .	635
	SINT or INT to DINT . . . . .	637
	Integer to REAL . . . . .	639
	DINT to SINT or INT . . . . .	639
	REAL to an integer. . . . .	640
<b>Function Block Attributes</b>	<b>Appendix B</b>	
	Introduction . . . . .	641
	Choose the Function Block Elements. . . . .	641
	Latching Data . . . . .	642
	Order of Execution . . . . .	644
	Resolve a Loop . . . . .	645
	Resolve Data Flow Between Two Blocks . . . . .	647
	Create a One Scan Delay . . . . .	648
	Summary. . . . .	648
	Function Block Responses to Overflow Conditions. . . . .	649
	Timing Modes. . . . .	650
	Common instruction parameters for timing modes. . . . .	652
	Overview of timing modes. . . . .	654
	Program/Operator Control. . . . .	655

<b>Structured Text Programming</b>	<b>Appendix C</b>	
	Introduction . . . . .	659
	Structured Text Syntax. . . . .	659
	Assignments . . . . .	661
	Specify a non-retentive assignment. . . . .	662
	Assign an ASCII character to a string. . . . .	663
	Expressions . . . . .	663
	Use arithmetic operators and functions . . . . .	665
	Use relational operators . . . . .	666
	Use logical operators . . . . .	668
	Use bitwise operators. . . . .	669
	Determine the order of execution. . . . .	669
	Instructions. . . . .	670
	Constructs. . . . .	671
	Some key words are reserved for future use . . . . .	671
	IF...THEN . . . . .	672
	CASE...OF. . . . .	675
	FOR...DO. . . . .	678
	WHILE...DO. . . . .	681
	REPEAT...UNTIL. . . . .	684
	Comments . . . . .	687
	ASCII Character Codes . . . . .	689
<b>Index</b>	Rockwell Automation Support . . . . .	705
	Installation Assistance . . . . .	705
<b>Back Cover</b>	New Product Satisfaction Return. . . . .	705

cuu duong than cong. com

cuu duong than cong. com

### Introduction

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

### Updated Information

This document contains the following changes:

Change	Page
Instruction locator table — Added the new digital and analog alarm instructions.	Instruction Locator
Chapter 1 — Added new chapter 1, Digital Alarm Instruction	31
Chapter 2 — Added new chapter 2, Analog Alarm Instruction	47
Remaining chapters 3...20 — Renumbered.	69...619

cuu duong than cong. com

cuu duong than cong. com

## Notes:


cuu duong than cong. com

cuu duong than cong. com



## Introduction

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

Task/Goal	Documents
Program the controller for sequential applications	<i>Logix5000 Controllers General Instructions Reference Manual</i> , publication 1756-RM003
<b>You are here</b> 	
Program the controller for process or drives applications	<i>Logix5000 Controllers Process Control and Drives Instructions Reference Manual</i> , publication 1756-RM006
Program the controller for motion applications	<i>Logix5000 Controllers Motion Instruction Set Reference Manual</i> , publication 1756-RM007
Program the controller to use equipment phases	<i>PhaseManager User Manual</i> , publication LOGIX-UM001
Import a text file or tags into a project	<i>Logix5000 Controllers Import/Export Reference Manual</i> , publication 1756-RM084
Export a project or tags to a text file	
Convert a PLC-5 or SLC 500 application to a Logix5000 application	<i>Logix5550 Controller Converting PLC-5 or SLC 500 Logic to Logix5550 Logic Reference Manual</i> , publication 1756-6.8.5




## Who Should Use This Manual

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




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

## Purpose of This Manual

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

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

The following icons help identify language specific information:

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

## Common Information for All Instructions

The Logix5000 instruction set has some common attributes:

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

## Conventions and Related Terms

### Set and clear

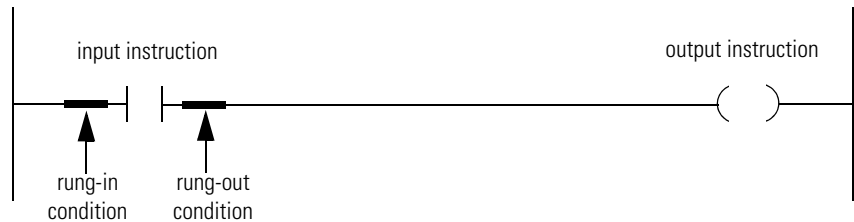
This manual uses set and clear to define the status of bits (booleans) and values (non-booleans):

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

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

## Relay ladder rung condition

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



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

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

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

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

Prescan does not occur for a program when:

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

## Function block states

### IMPORTANT

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

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

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

Every function block instruction also includes EnableIn and EnableOut parameters:

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

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

## Notes:

cuu duong than cong. com

cuu duong than cong. com

### 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
motion	<i>Logix5000 Controllers Motion Instruction Set Reference Manual</i> , publication 1756-RM007
PhaseManager	<i>PhaseManager User Manual</i> , publication LOGIX-UM001
process control	<i>Logix5000 Controllers Process Control and Drives Instruction Set Reference Manual</i> , publication 1756-RM006

cuu duong than cong. com

cuu duong than cong. com

Instruction:	Location:	Languages:
ABL ASCII Test For Buffer Line	616	relay ladder structured text
ABS Absolute Value	279	relay ladder structured text function block
ACB ASCII Chars in Buffer	578	relay ladder structured text
ACL ASCII Clear Buffer	581	relay ladder structured text
ACS Arc Cosine	536	relay ladder structured text function block
ADD Add	253	relay ladder structured text function block
AFI Always False Instruction	458	relay ladder
AHL ASCII Handshake Lines	583	relay ladder structured text
ALM Alarm	process control	structured text function block
ALMA Analog Alarm	47	relay ladder structured text function block
ALMD Digital Alarm	31	relay ladder structured text function block
AND Bitwise AND	305	relay ladder structured text function block
ARD ASCII Read	587	relay ladder structured text
ARL ASCII Read Line	591	relay ladder structured text
ASN Arc Sine	532	relay ladder structured text function block
ATN Arc Tangent	540	relay ladder structured text function block
AVE File Average	368	relay ladder
AWA ASCII Write Append	595	relay ladder structured text
AWT ASCII Write	600	relay ladder structured text
BAND Boolean AND	319	structured text function block

Instruction:	Location:	Languages:
BNOT Boolean NOT	328	structured text function block
BOR Boolean OR	322	structured text function block
BRK Break	475	relay ladder
BSL Bit Shift Left	388	relay ladder
BSR Bit Shift Right	392	relay ladder
BTD Bit Field Distribute	296	relay ladder
BTDT Bit Field Distribute with Target	296	structured text function block
BTR Message	140	relay ladder structured text
BTW Message	140	relay ladder structured text
BXOR Boolean Exclusive OR	325	structured text function block
CLR Clear	296	relay ladder structured text
CMP Compare	207	relay ladder
CONCAT String Concatenate	608	relay ladder structured text
COP Copy File	358	relay ladder structured text
COS Cosine	525	relay ladder structured text function block
CPS Synchronous Copy File	358	relay ladder structured text
CPT Compute	249	relay ladder
CTD Count Down	127	relay ladder
CTU Count Up	123	relay ladder
CTUD Count Up/Down	131	structured text function block
D2SD Discrete 2-State Device	process control	structured text function block



Instruction:	Location:	Languages:
D3SD Discrete 3-State Device	process control	structured text function block
DDT Diagnostic Detect	488	relay ladder
DEDT Deadtime	process control	structured text function block
DEG Degrees	559	relay ladder structured text function block
DELETE String Delete	610	relay ladder structured text
DERV Derivative	process control	structured text function block
DFF D Flip-Flop	process control	structured text function block
DIV Divide	263	relay ladder structured text function block
DTOS DINT to String	626	relay ladder structured text
DTR Data Transitional	496	relay ladder
EOT End of Transition	460	relay ladder structured text
EQU Equal to	207	relay ladder structured text function block
ESEL Enhanced Select	process control	structured text function block
EVENT Trigger Event Task	466	relay ladder structured text
FAL File Arithmetic and Logic	337	relay ladder
FBC File Bit Comparison	480	relay ladder
FFL FIFO Load	396	relay ladder
FFU FIFO Unload	402	relay ladder
FGEN Function Generator	process control	structured text function block
FIND Find String	612	relay ladder structured text
FLL File Fill	364	relay ladder

Instruction:	Location:	Languages:
FOR For	472	relay ladder
FRD Convert to Integer	565	relay ladder function block
FSC File Search and Compare	349	relay ladder
GEQ Greater than or Equal to	216	relay ladder structured text function block
GRT Greater Than	220	relay ladder structured text function block
GSV Get System Value	173	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	641	function block
INSERT Insert String	614	relay ladder structured text
INTG Integrator	process control	structured text function block
IOT Immediate Output	200	relay ladder structured text
IREF Input Reference	641	function block
JKFF JK Flip-Flop	process control	structured text function block
JMP Jump to Label	436	relay ladder
JSR Jump to Subroutine	438	relay ladder structured text function block
JXR Jump to External Routine	449	relay ladder
LBL Label	436	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	224	relay ladder structured text function block

Instruction:	Location:	Languages:
LES Less Than	228	relay ladder structured text function block
LFL LIFO Load	408	relay ladder
LFU LIFO Unload	414	relay ladder
LIM Limit	232	relay ladder function block
LN Natural Log	546	relay ladder structured text function block
LOG Log Base 10	(1)	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
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

Instruction:	Location:	Languages:
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	motion	relay ladder structured text
MCCM Motion Coordinated Circular Move	motion	relay ladder structured text
MCCP Motion Calculate Cam Profile	motion	relay ladder structured text
MCD Motion Change Dynamics	motion	relay ladder structured text
MCLM Motion Coordinated Linear Move	motion	relay ladder structured text
MCR Master Control Reset	454	relay ladder
MCSD Motion Coordinated Shutdown	motion	relay ladder structured text
MCS Motion Coordinated Stop	motion	relay ladder structured text
MCSR Motion Coordinated Shutdown Reset	motion	relay ladder structured text
MCT Motion Coordinated Transform	motion	relay ladder structured text
MCTP Motion Calculate Transform Position	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

Instruction:	Location:	Languages:
MDW Motion Disarm Watch	motion	relay ladder structured text
MEQ Mask Equal to	238	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	616	relay ladder structured text
MINC Minimum Capture	process control	structured text function block
MOD Modulo	268	relay ladder structured text function block
MOV Move	285	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	140	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	260	relay ladder structured text function block
MUX Multiplexer	process control	function block
MVM Masked Move	287	relay ladder

Instruction:	Location:	Languages:
MVMT Masked Move with Target	290	structured text function block
NEG Negate	276	relay ladder structured text function block
NEQ Not Equal to	243	relay ladder structured text function block
NOP No Operation	459	relay ladder
NOT Bitwise NOT	315	relay ladder structured text function block
NTCH Notch Filter	process control	structured text function block
OCON Output Wire Connector	641	function block
ONS One Shot	80	relay ladder
OR Bitwise OR	308	relay ladder structured text function block
OREF Output Reference	641	function block
OSFI One Shot Falling with Input	92	structured text function block
OSF One Shot Falling	86	relay ladder
OSRI One Shot Rising with Input	83	structured text function block
OSR One Shot Rising	83	relay ladder
OTE Output Energize	74	relay ladder
OTL Output Latch	76	relay ladder
OTU Output Unlatch	78	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

Instruction:	Location:	Languages:
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	499	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
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	559	relay ladder structured text function block
RESD Reset Dominant	process control	structured text function block
RES Reset	136	relay ladder
RET Return	438 and 476	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	105	relay ladder

Instruction:	Location:	Languages:
RTOR Retentive Timer On with Reset	118	structured text function block
RTOS REAL to String	629	relay ladder structured text
SBR Subroutine	438	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	462	relay ladder structured text
SFR SFC Reset	464	relay ladder structured text
SIN Sine	522	relay ladder structured text function block
SIZE Size In Elements	384	relay ladder structured text
SNEG Selected Negate	process control	structured text function block
SOC Second-Order Controller	process control	structured text function block
SQL Sequencer Input	422	relay ladder
SQL Sequencer Load	430	relay ladder
SQO Sequencer Output	426	relay ladder
SQR Square Root	272	relay ladder function block
SQRT Square Root	272	structured text
SRT File Sort	373	relay ladder structured text
S RTP Split Range Time Proportional	process control	structured text function block
SSUM Selected Summer	process control	structured text function block

Instruction:	Location:	Languages:
SSV Set System Value	173	relay ladder structured text
STD File Standard Deviation	378	relay ladder
STOD String To DINT	622	relay ladder structured text
STOR String To REAL	624	relay ladder structured text
SUB Subtract	257	relay ladder structured text function block
SWPB Swap Byte	301	relay ladder structured text
TAN Tangent	529	relay ladder structured text function block
TND Temporary End	452	relay ladder
TOD Convert to BCD	562	relay ladder function block
TOFR Timer Off Delay with Reset	114	structured text function block
TOF Timer Off Delay	100	relay ladder
TONR Timer On Delay with Reset	110	structured text function block
TON Timer On Delay	96	relay ladder
TOT Totalizer	process control	structured text function block
TRN Truncate	567	relay ladder function block
TRUNC Truncate	567	structured text
UID User Interrupt Disable	456	relay ladder structured text
UIE User Interrupt Enable	456	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	70	relay ladder
XIO Examine If Open	72	relay ladder

Instruction:	Location:	Languages:
XOR Bitwise Exclusive OR	311	relay ladder structured text function block
XPY X to the Power of Y	552	relay ladder structured text function block

cuu duong than cong. com

cuu duong than cong. com

## Digital Alarm Instruction (ALMD)

### Introduction

Use the ALMD instruction to detect alarms based on Boolean (true/false) conditions.

You can use the ALMD instruction to provide control of boolean alarms using program or operator interface control ("Prog" or "Oper" control parameters).

The ALMD instruction has parameters that appear as operands on the instruction. The instruction operands are not common to all languages. The ALMD instruction has a corresponding tag structure (ALARM\_DIGITAL) which is common to all languages.

Refer to Structure Definition for ALARM\_DIGITAL Tag on page 34 for descriptions of the tag elements and alarm execution.

cuu duong than cong. com

### About Operator Parameters

Operator parameters (for example, OperSuppress) work with any Rockwell Automation or third-party operator interface to allow control of alarm states.

When an Operator request is set, the ALMD instruction evaluates whether it can respond to the request, then always resets the request. This lets operator interfaces work with this instruction by merely resetting the desired request bit. You don't have to program the operator interface to reset the request bits.

CUU

### Using the ALMD Instruction to Subscribe to and Display Alarms

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 alarms in the controller. Using several output parameters (shown in the output parameter tables that follow), you can monitor the instruction to see the alarm

subscription status and to display alarm status changes. If a connection to RSLinx Enterprise software is lost, the controller can briefly buffer alarm data until the connection is restored.

## Digital Alarm Operands



These operands are located on the instruction.

## Ladder Logic Operands

Operand	Type	Format	Description
ALMD tag	ALARM_DIGITAL	structure	ALMD structure
ProgAck	BOOL	Tag Immediate	Value is copied to .ProgAck when instruction executes. On transition from False to True, acknowledges alarm (if acknowledgement is required).
ProgReset	BOOL	Tag Immediate	Value is copied to .ProgReset when instruction executes. On transition from False to True, resets alarm (if resetting is required).
ProgDisable	BOOL	Tag Immediate	Value is copied to .ProgDisable when instruction executes. When True, disables alarm (does not override Enable Commands).
ProgEnable	BOOL	Tag Immediate	Value is copied to .ProgEnable when instruction executes. When True, enables alarm (takes precedence over Disable commands).
MinDurationPRE	DINT	Immediate	Specifies how long the alarm condition must be met before it is reported (milliseconds).
MinDurationACC	DINT	Immediate	Indicates the current accumulator value for the alarm's MinDuration timer.

## Structured Text Operands

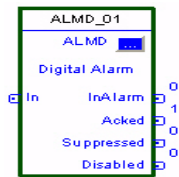


ALMD (ALMD, In, ProgAck, ProgReset, ProgDisable, ProgEnable)

Operand	Type	Format	Description
ALMD	ALARM_DIGITAL	structure	ALMD structure
In	BOOL	Tag Immediate	Alarm Condition to be monitored. Value is copied to .In when instruction executes.
ProgAck	BOOL	Tag Immediate	Value is copied to .ProgAck when instruction executes. On transition from False to True, acknowledges alarm (if acknowledgement is required).



Operand	Type	Format	Description
ProgReset	BOOL	Tag Immediate	Value is copied to .ProgReset when instruction executes. On transition from False to True, resets alarm (if resetting is required).
ProgDisable	BOOL	Tag Immediate	Value is copied to .ProgDisable when instruction executes. When True, disables alarm (does not override Enable Commands).
ProgEnable	BOOL	Tag Immediate	Value is copied to .ProgEnable when instruction executes. When True, enables alarm (takes precedence over Disable commands).



## Function Block Operands

Operand	Type	Format	Description
ALMD tag	ALARM_DIGITAL	structure	ALMD structure

cuu duong than cong. com

cuu duong than cong. com

## Structure Definition for ALARM\_DIGITAL Tag

The tag structure, ALARM\_DIGITAL, which corresponds to the ALMD tag, is common to all languages, except where noted.

### Input Parameters

Input Parameter	Data Type	Description
EnableIn	BOOL	System-defined input.  Ladder Logic: Corresponds to the rung state. Does not affect processing.  Structured Text: Does not affect processing.  Function Block: Enable input. If cleared, the instruction does not execute and outputs are not updated.  Default is set.
In	BOOL	The digital signal input to the instruction.  Ladder Logic: Follows rung state.  Structured Text: Copied from instruction operand.  Default is 0.0.
InFault	BOOL	Input bad health indicator. If In is read from a digital input, then InFault is normally controlled by the digital input fault status. When InFault is set, it indicates the input signal has an error.  Default is cleared = good health.
Condition	BOOL	Specifies how alarm is activated.  When set, alarm condition is activated when In is Set.  When reset, alarm condition is activated when In is Cleared.  Default is set.
AckRequired	BOOL	Specifies whether alarm acknowledgement is required.  Set - Acknowledgement required.  Cleared - Acknowledgement not required.  Default is set.
Latched	BOOL	Specifies whether alarm is latched. Latched alarms remain InAlarm when the alarm condition becomes false, until a Reset command is received.  Set - Latched.  Cleared - Unlatched.  Default is cleared.

Input Parameter	Data Type	Description
ProgAck	BOOL	<p>Program Acknowledge. Set by the user program to acknowledge the alarm. Requires a False-to-True transition while the alarm is Unacknowledged.</p> <p>Ladder Logic: Copied from instruction operand.</p> <p>Structured Text: Copied from instruction operand.</p> <p>Default is cleared.</p>
OperAck	BOOL	<p>Operator Acknowledge. Set by the operator interface to acknowledge the alarm. Requires a False-to-True transition while the alarm is Unacknowledged. The alarm instruction clears this parameter.</p> <p>Default is cleared.</p>
ProgReset	BOOL	<p>Program Reset. Set by the user program to reset the alarm. Requires a False-to-True transition while the alarm is InAlarm and the In condition is not in alarm.</p> <p>Ladder Logic: Copied from instruction operand.</p> <p>Structured Text: Copied from instruction operand.</p> <p>Default is cleared.</p>
OperReset	BOOL	<p>Operator Reset. Set by the operator interface to reset the alarm. Requires a False-to-True transition while the alarm is InAlarm and the In condition is not in alarm. The alarm instruction clears this parameter.</p> <p>Default is cleared.</p>
ProgSuppress	BOOL	<p>Program Suppress. Set by the user program to suppress the alarm.</p> <p>Default is cleared.</p>
OperSuppress	BOOL	<p>Operator Suppress. Set by the operator interface to suppress the alarm. The alarm instruction clears this parameter.</p> <p>Default is cleared.</p>
ProgUnsuppress	BOOL	<p>Program Unsuppress. Set by the user program to unsuppress the alarm. Takes precedence over Suppress commands.</p> <p>Default is cleared.</p>
OperUnsuppress	BOOL	<p>Operator Unsuppress. Set by the operator interface to unsuppress the alarm. Takes precedence over Suppress commands. The alarm instruction clears this parameter.</p> <p>Default is cleared.</p>
ProgDisable	BOOL	<p>Program Disable. Set by the user program to disable the alarm.</p> <p>Ladder Logic: Copied from instruction operand.</p> <p>Structured Text: Copied from instruction operand.</p> <p>Default is cleared.</p>

Input Parameter	Data Type	Description
OperDisable	BOOL	Operator Disable. Set by the operator interface to disable the alarm. The alarm instruction clears this parameter.  Default is cleared.
ProgEnable	BOOL	Program Enable. Set by the user program to enable the alarm. Takes precedence over Disable command.  Ladder Logic: Copied from instruction operand.  Structured Text: Copied from instruction operand.  Default is cleared.
OperEnable	BOOL	Operator Enable. 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	A False-to-True transition resets the alarm count to zero.  Default is cleared.
UseProgTime	BOOL	Specifies whether the controller's clock is used to timestamp the InAlarm and ReturnToNormal events, or if these events are timestamped by the user program (using the ProgTime parameter).  Set - ProgTime value provides timestamp.  Cleared - Controller's clock provides timestamp.  Default is cleared.
ProgTime	LINT	Specifies a timestamp value for the InAlarm and ReturnToNormal events, if UseProgTime is Set.
Severity	DINT	Specifies the severity of the alarm.  Valid = 1 to 1000 (1000 = most severe; 1 = least severe).  Default is 500.
MinDurationPRE	DINT	Specifies the minimum duration preset for the alarm condition to remain true before the alarm is marked as InAlarm and alarm notification is sent to clients (milliseconds).  Valid = 0 to 2147483647.  Default is 0.

## Output Parameters

Output Parameter	Data Type	Description
EnableOut	BOOL	System-defined output parameter. Follows the state of EnableIn.
InAlarm	BOOL	Indicates whether the alarm is active.  Set - In alarm.  Cleared - Normal.
Acked	BOOL	Indicates whether the alarm is acknowledged.  Set - Alarm acknowledged.  Cleared - Alarm not acknowledged.  (Always set when AckRequired is false).
InAlarmUnack	BOOL	Indicates whether an alarm is active (InAlarm) and unacknowledged.  Set - Alarm is both active (InAlarm) and unacknowledged.  Cleared - Alarm is either inactive or acknowledged (or both).
Suppressed	BOOL	Indicates whether the alarm is suppressed.  Set - Alarm suppressed.  Cleared - Alarm unsuppressed.
Disabled	BOOL	Indicates whether the alarm is disabled.  Set - Alarm disabled.  Cleared - Alarm enabled.
MinDurationACC	DINT	Indicates the elapsed time since the alarm was detected. When this value reaches MinDurationPRE, the alarm becomes active (InAlarm), and a notification is sent to clients.
AlarmCount	DINT	The number of times the alarm has been activated (InAlarm). 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.
DeliveryER	BOOL	Indicates alarm notification message delivery error.  Set – delivery error – either no alarm subscriber was subscribed or at least one subscriber did not receive the latest alarm change state message.  Cleared – delivery successful or in progress.

Output Parameter	Data Type	Description
DeliveryDN	BOOL	Indicates alarm notification message delivery success.  Set – delivery success – at least one subscriber was subscribed and all subscribers received the latest alarm change state message successfully.  Cleared – delivery not completed successfully or in progress.
DeliveryEN	BOOL	Indicates alarm notification message delivery in process.  Set – delivery in progress.  Cleared – delivery not in progress.
NoSubscriber	BOOL	Indicates that the alarm had no subscribers when attempting to deliver the most recent state change message.  Set – no subscribers.  Cleared – At least one subscriber.
NoConnection	BOOL	Indicates that all of the alarm's subscribers were disconnected when attempting to deliver the most recent state change message:  Set – all subscribers disconnected.  Cleared – at least one subscriber connected.
CommError	BOOL	Indicates that there was a communication error when delivering last alarm message to at least one subscriber:  Set – communication errors – all retries exhausted.  Cleared – all connected subscribers successfully received alarm message  If this error is indicated then it means that a subscriber was subscribed, and it had a connection opened, but the message was not delivered successfully.
AlarmBuffered	BOOL	Indicates that the alarm message was buffered when not delivered to subscriber(s), either due to a CommError or a lost Connection:  Set – alarm message buffered for at least one subscriber.  Cleared – alarm message is not buffered.
Subscribers	DINT	Indicates number of subscribers for this alarm.
SubscNotified	DINT	Indicates number of subscribers successfully notified about the most recent alarm state change.
Status	DINT	Indicates the bit-mapped status of the instruction execution.

Output Parameter	Data Type	Description
InstructFault	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.
InFaulted	BOOL	User program has set InFault to indicate bad quality input data.
SeverityInv	BOOL	Indicates invalid alarm severity configuration.  If severity <1, the instruction uses Severity = 1.  If severity >1000, the instruction uses Severity = 1000.

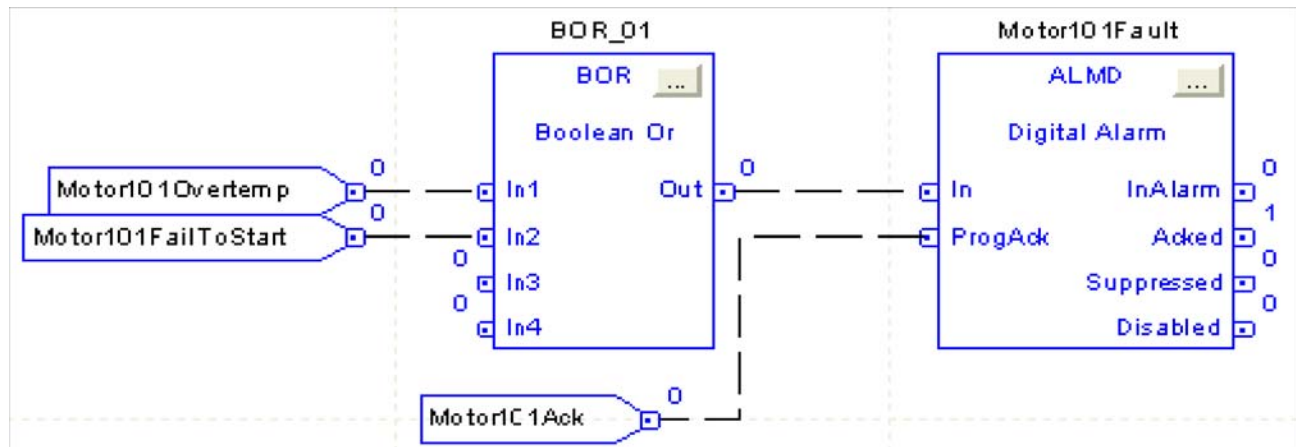
cuu duong than cong. com

cuu duong than cong. com





## Function Block



cuu duong than cong. com

cuu duong than cong. com

## Execution

The tables below show execution action for Ladder Logic and Function Block programming languages.

### Ladder Logic

Condition	Action
prescan	The rung-condition-out is set to false. All operator requests, timestamps, and delivery flags are cleared. The alarm condition is set to OutOfAlarm and Acknowledged.
rung-condition-in is false	The rung-condition-out is set to false. 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. The .In parameter is set, and the instruction evaluates to determine the alarm state.
postscan	The rung-condition-out is set to false.

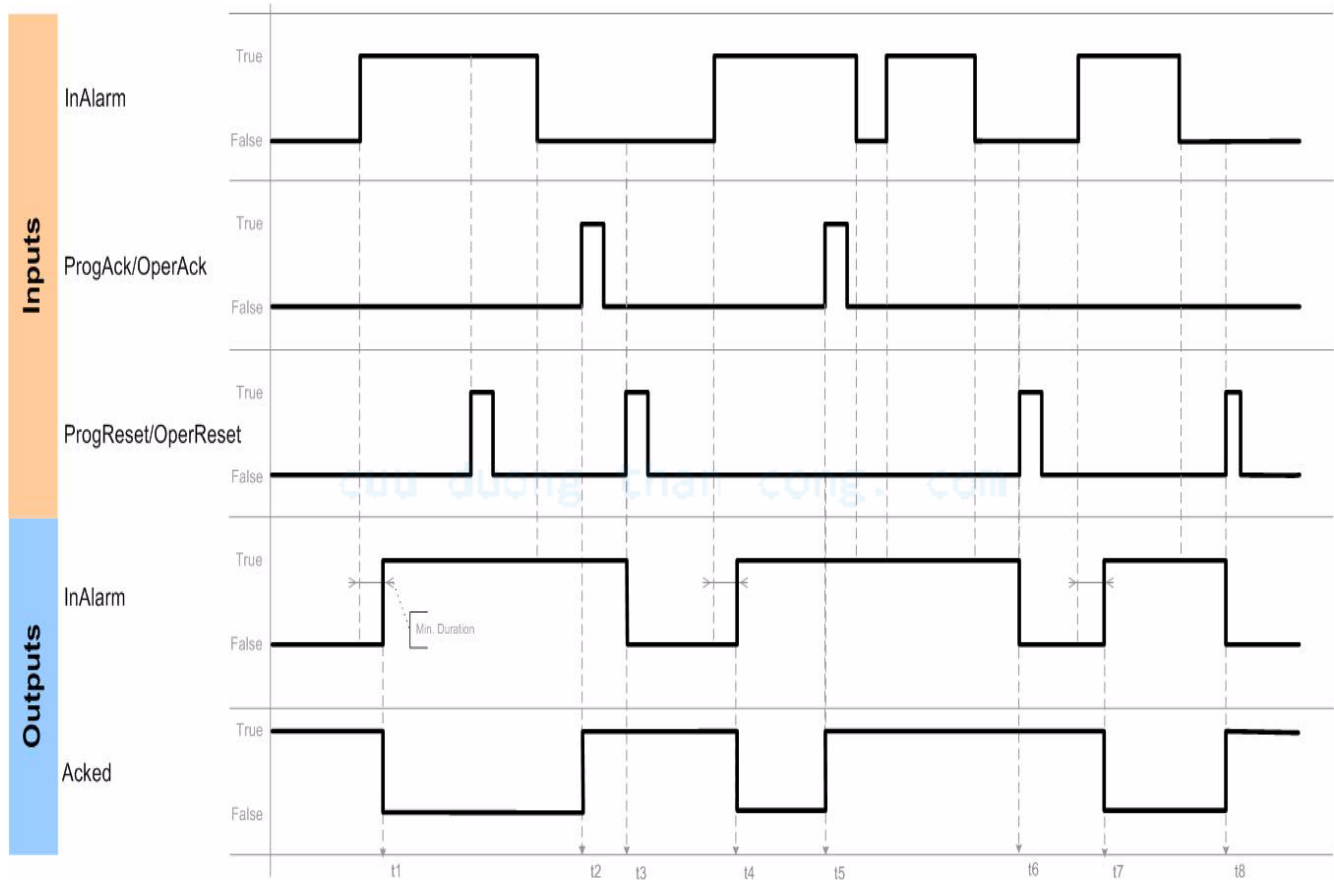
### Function Block

Condition	Action
prescan	All operator requests, timestamps, and delivery flags are cleared. The alarm condition is set to OutOfAlarm and Acknowledged.
instruction first scan	No action taken.
instruction first run	No action taken.
EnableIn is cleared	The instruction does not execute. EnableOut is cleared.
EnableIn is set	The instruction executes. EnableOut is set.
postscan	No action taken.

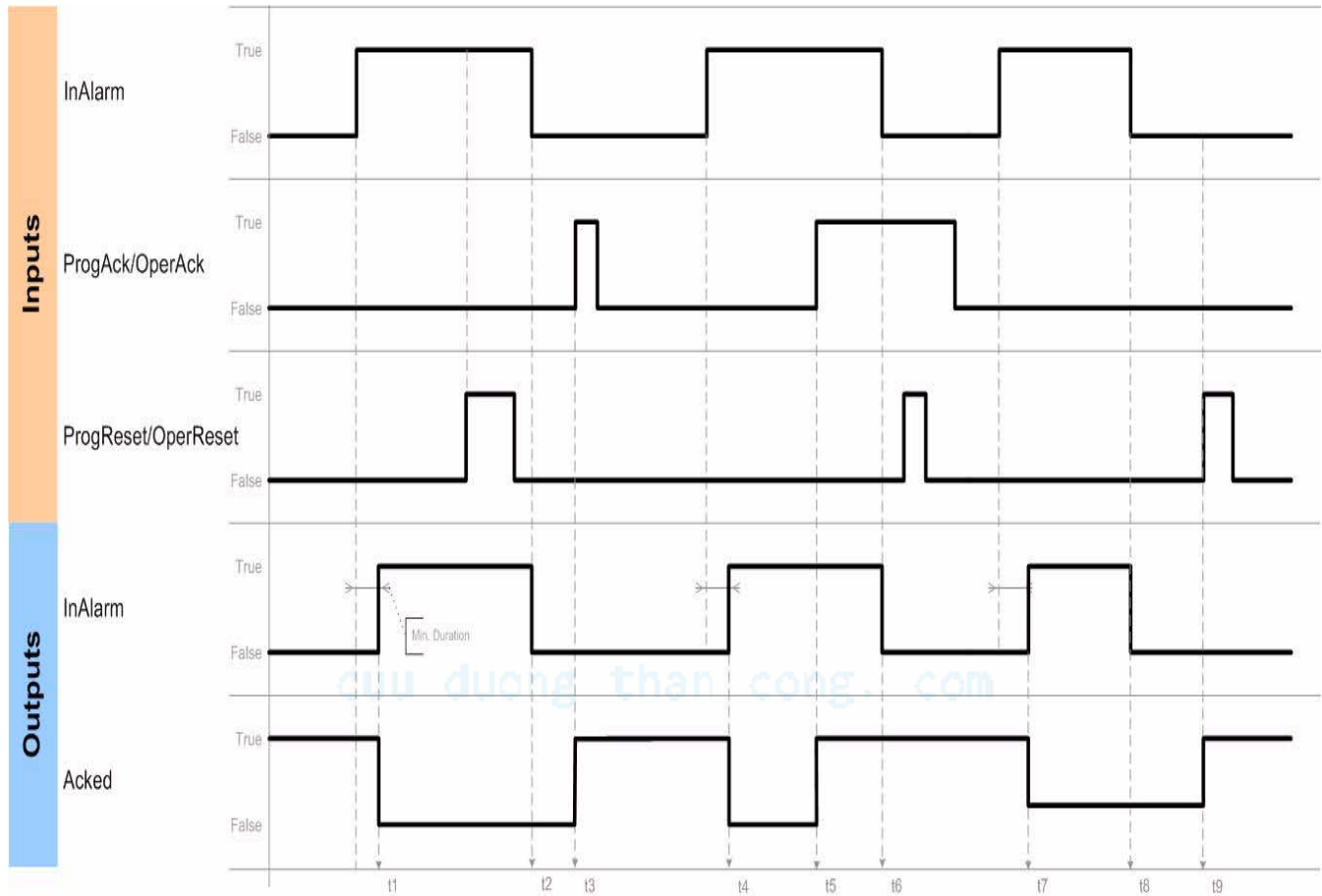
## Digital State Timing Diagrams

These timing diagrams show the sequence of bit operations in a typical system configuration.

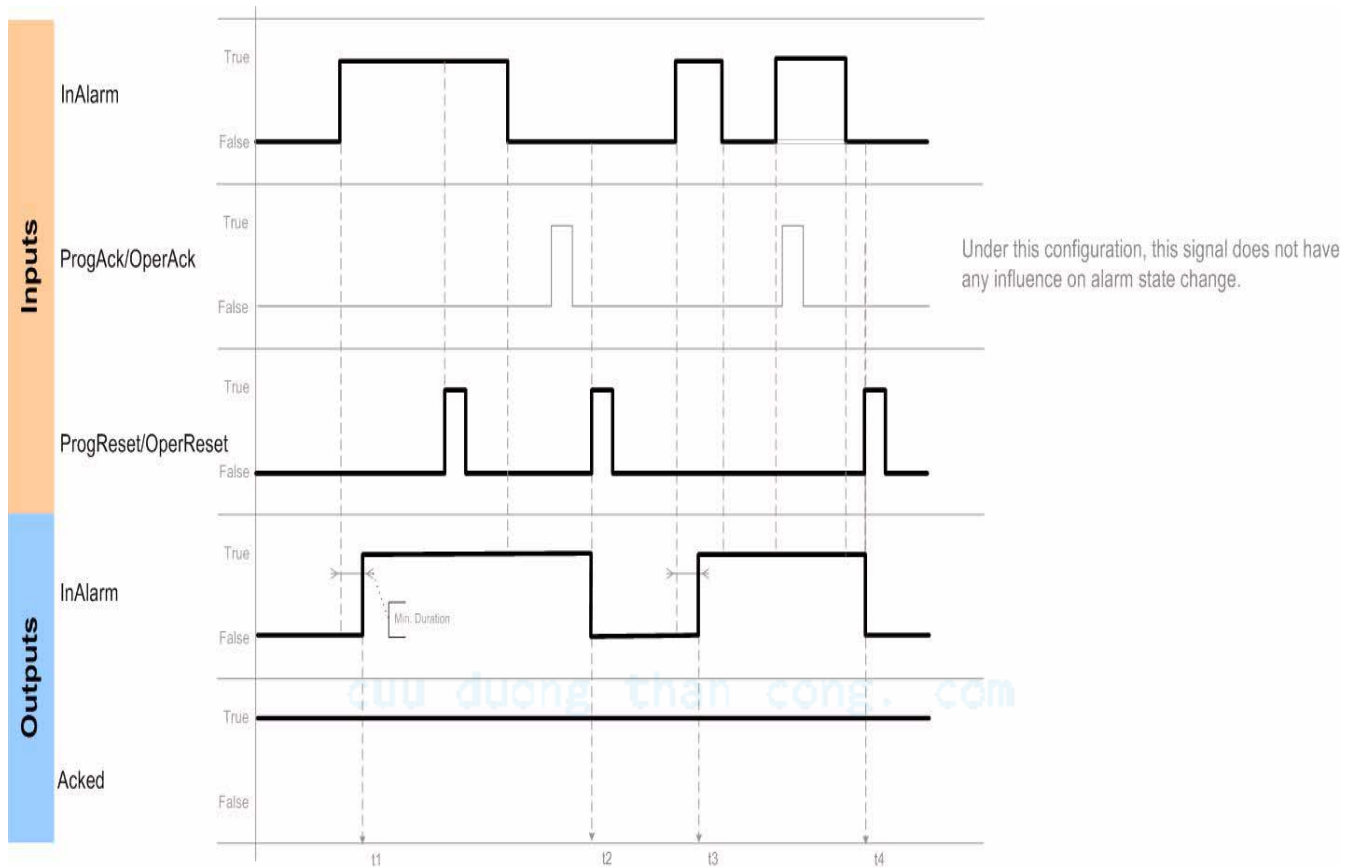
### Alarm Acknowledge Required and Latched



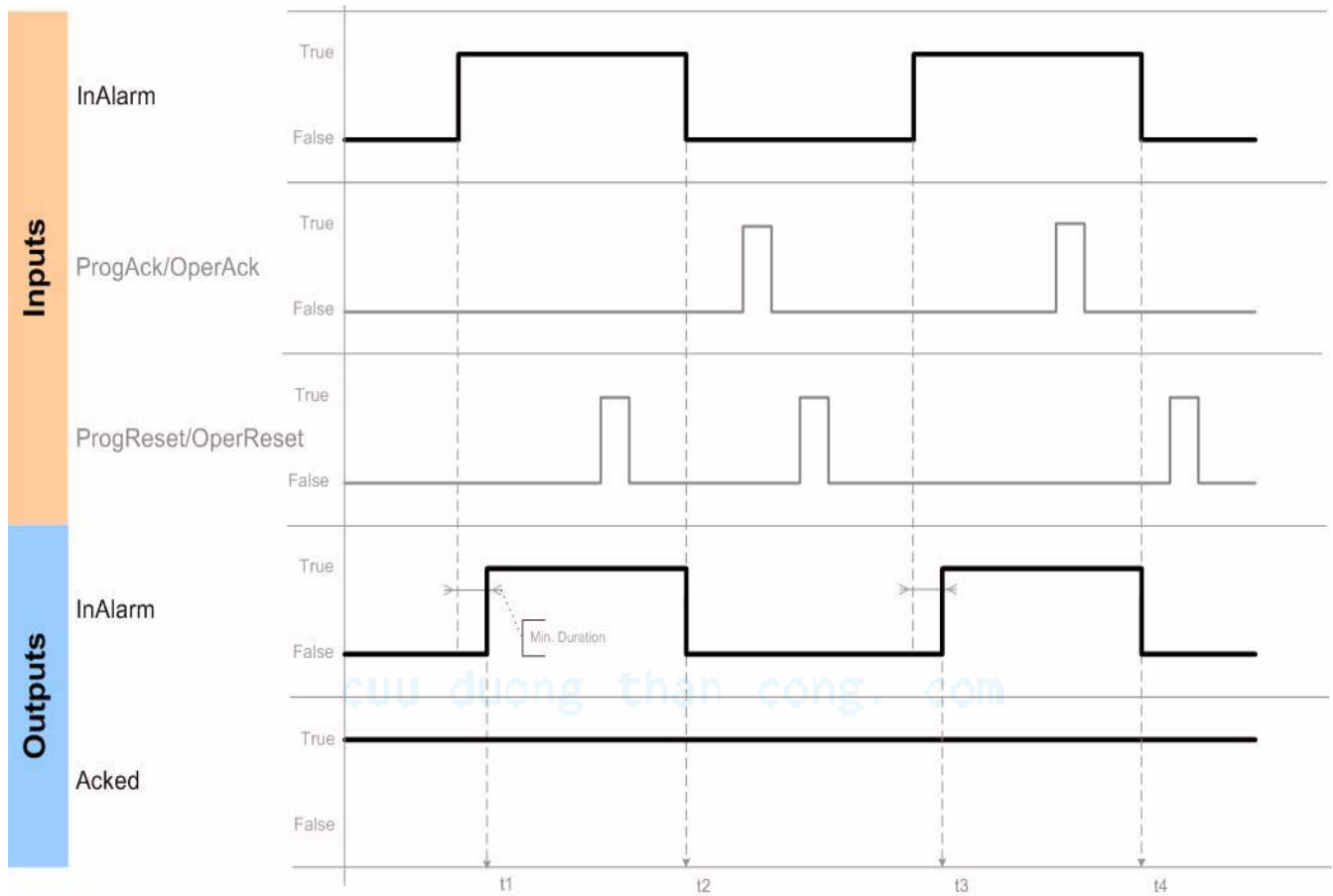
## Alarm Acknowledge Required and Not Latched



## Alarm Acknowledge Not Required and Latched



## Alarm Acknowledge Not Required and Not Latched



## Analog Alarm Instruction (ALMA)

### Introduction

Use the ALMA instruction to detect alarms based on the level or rate of change of analog value.

The ALMA instruction has parameters which appear as operands on the instruction. The instruction operands are not common to all languages. The ALMA instruction has a corresponding tag structure (ALARM\_ANALOG) that is common to all languages.

Refer to Structure Definition For ALARM\_ANALOG Tag on page 50 for descriptions of the tag elements and alarm execution.

### About Operator Parameters

Operator parameters (for example, OperSuppress) work with any Rockwell Automation or third-party operator interface to allow control of alarm states.

When an Operator request is set, the ALMA instruction evaluates whether it can respond to the request, then always resets the request. This lets operator interfaces work with this instruction by merely setting the desired request bit. You don't have to program the operator interface to reset the request bits.

### Using the ALMA Instruction to Subscribe to and Display Alarms

The ALMA 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 alarms in the controller. Using several output parameters (shown in the output parameter tables that follow), you can monitor the instruction to see the alarm subscription status and to display alarm status changes. If a connection to RSLinx Enterprise software is lost, the controller can briefly buffer alarm data until the connection is restored.

## Analog Alarm Operands



ALMA	
ALMA	Alarm Alarm
In	77
ProgAckAll	77
ProgDisable	77
ProgEnable	77
HHLimit	77
HLimit	77
LLimit	77
LLLimit	77

These operands are located on the instruction.

## Ladder Logic Operands

Operand	Type	Format	Description
ALMA tag	ALARM_ANALOG	structure	ALMA structure
In	REAL DINT INT SINT	Tag Immediate	Value is copied to .In when instruction executes. The alarm input value, which is compared with alarm limits to detect the alarm conditions.
ProgAckAll	BOOL	Tag Immediate	Value is copied to .ProgAckAll when instruction executes. On transition from False to True, acknowledges all alarm conditions that require acknowledgement.
ProgDisable	BOOL	Tag Immediate	Value is copied to .ProgDisable when instruction executes. When True, disables alarm (does not override Enable Commands).
ProgEnable	BOOL	Tag Immediate	Value is copied to .ProgEnable when instruction executes. When True, enables alarm (takes precedence over Disable commands).
HHLimit	REAL	Immediate	High High alarm limit
HLimit	REAL	Immediate	High alarm limit
LLimit	REAL	Immediate	Low alarm limit
LLLimit	REAL	Immediate	Low low alarm limit

## Structured Text Operands



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



Operand	Type	Format	Description
ALMA	ALARM_ANALOG	structure	ALMA structure
In	REAL DINT INT SINT	Tag Immediate	Value is copied to .In when instruction executes. The alarm input value, which is compared with alarm limits to detect the alarm conditions.
ProgAckAll	BOOL	Tag Immediate	Value is copied to .ProgAckAll when instruction executes. On transition from False to True, acknowledges all alarm conditions that require acknowledgement.
ProgDisable	BOOL	Tag Immediate	Value is copied to .ProgDisable when instruction executes. When True, disables alarm (does not override Enable Commands).
ProgEnable	BOOL	Tag Immediate	Value is copied to .ProgEnable when instruction executes. When True, enables alarm (takes precedence over Disable commands).



ALMA_01	
ALMA	
Analog Alarm	
In	0
HHInAlarm	0
HInAlarm	0
LInAlarm	0
LLInAlarm	0
ROCPInAlarm	0
ROCNInAlarm	0
HHAcked	1
HAcked	1
LAcked	1
LLAcked	1
ROCPAcked	1
ROCNAcked	1
Suppressed	0
Disabled	0

Function Block Operands

Operand	Type	Format	Description
ALMA tag	ALARM_ANALOG	structure	ALMA structure

## Structure Definition For ALARM\_ANALOG Tag

The tag structure, ALARM\_ANALOG, which corresponds to the ALMA tag, is common to all languages, except where noted.

### Input Parameters

Input Parameter	Data Type	Description
EnableIn	BOOL	System-defined input.  Ladder Logic: Corresponds to the rung state. If cleared, the instruction does not execute and outputs are not updated.  Structured Text: Does not affect processing.  Function Block: Enable input. 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 the alarm condition.  Ladder Logic: copied from instruction operand.  Structured Text: copied from instruction operand.  Default is 0.0.
InFault	BOOL	Input bad health indicator. If In is read from an analog input, then InFault is normally controlled by the analog input fault status. When InFault is set, it indicates the input signal has an error.  Default is cleared = good health.
HHEnabled	BOOL	Specifies whether a high-high alarm condition detection is enabled. Set - Condition detection enabled. Cleared - Condition detection disabled.  Default is set.
HEnabled	BOOL	Specifies whether a high alarm condition detection is enabled. Set - Condition detection enabled. Cleared - Condition detection disabled.  Default is set.
LEnabled	BOOL	Specifies whether a low alarm condition detection is enabled. Set - Condition detection enabled. Cleared - Condition detection disabled.  Default is set.
LLEnabled	BOOL	Specifies whether a low-low alarm condition detection is enabled. Set - Condition detection enabled. Cleared - Condition detection disabled.  Default is set.
AckRequired	BOOL	Specifies whether alarm acknowledgement is required. Set - Acknowledgement required. Cleared - Acknowledgement not required.  Default is set.

Input Parameter	Data Type	Description
ProgAckAll	BOOL	Program Acknowledge All. Set by the user program to acknowledge all conditions of this alarm. Requires a False-to-True transition while the alarm condition(s) are Unacknowledged.  Default is cleared.
OperAckAll	BOOL	Operator Acknowledge All. Set by the operator interface to acknowledge all conditions of this alarm. Requires a False-to-True transition while the alarm condition(s) are Unacknowledged. The alarm instruction clears this parameter.  Default is cleared.
HHProgAck	BOOL	High-High Alarm Program Acknowledge. Set by the user program to acknowledge the alarm high-high condition. Requires a False-to-True transition while the alarm condition is Unacknowledged.  Default is cleared.
HHOperAck	BOOL	High-High Alarm Operator Acknowledge. Set by the operator interface to acknowledge the alarm high-high condition. Requires a False-to-True transition while the alarm condition is Unacknowledged. The alarm instruction clears this parameter.  Default is cleared.
HProgAck	BOOL	High Alarm Program Acknowledge. Set by the user program to acknowledge the alarm high condition. Requires a False-to-True transition while the alarm condition is Unacknowledged.  Default is cleared.
HOperAck	BOOL	High Alarm Operator Acknowledge. Set by the operator interface to acknowledge the alarm high condition. Requires a False-to-True transition while the alarm condition is Unacknowledged. The alarm instruction clears this parameter.  Default is cleared.
LProgAck	BOOL	Low Alarm Program Acknowledge. Set by the user program to acknowledge the alarm low condition. Requires a False-to-True transition while the alarm condition is Unacknowledged.  Default is cleared.
LOperAck	BOOL	Low Alarm Operator Acknowledge. Set by the operator interface to acknowledge the alarm low condition. Requires a False-to-True transition while the alarm condition is Unacknowledged. The alarm instruction clears this parameter.  Default is cleared.
LLProgAck	BOOL	Low-Low Alarm Program Acknowledge. Set by the user program to acknowledge the alarm low-low condition. Requires a False-to-True transition while the alarm condition is Unacknowledged.  Default is cleared.
LLOperAck	BOOL	Low-Low Alarm Operator Acknowledge. Set by the operator interface to acknowledge alarm low-low conditions. Requires a False-to-True transition while the alarm condition is Unacknowledged. The alarm instruction clears this parameter.  Default is cleared.

Input Parameter	Data Type	Description
ROCPosProgAck	BOOL	Rate Of Change Positive Alarm Program Acknowledge. Set by the user program to acknowledge a positive rate-of-change alarm condition. Requires a False-to-True transition while the alarm condition is Unacknowledged.  Default is cleared.
ROCPosOperAck	BOOL	Rate Of Change Positive Alarm Operator Acknowledge. Set by the operator interface to acknowledge a positive rate-of-change alarm condition. Requires a False-to-True transition while the alarm condition is Unacknowledged. The alarm instruction clears this parameter.  Default is cleared.
ROCNegProgAck	BOOL	Rate Of Change Negative Alarm Program Acknowledge. Set by the user program to acknowledge a negative rate-of-change alarm condition. Requires a False-to-True transition while the alarm condition is Unacknowledged.  Default is cleared.
ROCNegOperAck	BOOL	Rate Of Change Negative Alarm Operator Acknowledge. Set by the operator interface to acknowledge a negative rate-of-change alarm condition. Requires a False-to-True transition while the alarm condition is Unacknowledged. The alarm instruction clears this parameter.  Default is cleared.
ProgSuppress	BOOL	Program Suppress. Set by the user program to suppress the alarm.  Default is cleared.
OperSuppress	BOOL	Operator Suppress. Set by the operator interface to suppress the alarm. The alarm instruction clears this parameter.  Default is cleared.
ProgUnsuppress	BOOL	Program Unsuppress. Set by the user program to unsuppress the alarm. Takes precedence over Suppress command.
OperUnsuppress	BOOL	Operator Unsuppress. Set by the operator interface to unsuppress the alarm. Takes precedence over Suppress command. The alarm instruction clears this parameter.  Default is cleared.
ProgDisable	BOOL	Program Disable. Set by the user program to disable the alarm.  Default is cleared.
OperDisable	BOOL	Operator Disable. Set by the operator interface to disable the alarm. The alarm instruction clears this parameter.  Default is cleared.
ProgEnable	BOOL	Program Enable. Set by the user program to enable the alarm. Takes precedence over Disable command.  Default is cleared.
OperEnable	BOOL	Operator Enable. 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	A False-to-True transition resets the alarm counts for all conditions to zero. Default is cleared.
HHLimit	REAL	Specifies the high-high alarm limit. Valid = HLimit < HHLimit < maximum positive float. Default = 0.0.
HHSeverity	DINT	Specifies the severity of the high-high alarm condition. Valid = 1 to 1000 (1000 = most severe; 1 = least severe). Default = 500.
HLimit	REAL	Specifies the high alarm limit. Valid = LLimit < HLimit < HHLimit. Default = 0.0.
HSeverity	DINT	Specifies the severity of the high alarm condition. Valid = 1 to 1000 (1000 = most severe; 1 = least severe). Default = 500.
LLimit	REAL	Specifies the low alarm limit. Valid = LLLimit < LLimit < HLimit. Default = 0.0.
LSeverity	DINT	Specifies the severity of the low alarm condition. Valid = 1 to 1000 (1000 = most severe; 1 = least severe). Default = 500.
LLLimit	REAL	Specifies the low-low alarm limit. Valid = maximum negative float < LLLimit < LLimit. Default = 0.0.
LLSeverity	DINT	Specifies the severity of the low-low alarm condition. Valid = 1 to 1000 (1000 = most severe; 1 = least severe). Default = 500.

Input Parameter	Data Type	Description
MinDurationPRE	DINT	<p>Specifies the minimum duration preset for the alarm In value to be in any level-based non-normal condition before the alarm notification is sent to clients (milliseconds). Does not apply to Rate of Change limits. Once the minimum duration preset is reached, any alarm notifications for subsequent alarm level events are sent immediately. The minimum duration time will be reset once the alarm returns to normal.</p> <p>Valid = 0 to 2147483647.</p> <p>Default = 0.</p>
Deadband	REAL	<p>Specifies the deadband for detecting that high-high, high, low, and low-low alarm levels have returned to normal.</p> <p>Valid = 0.0 to maximum possible float.</p> <p>Default = 0.0.</p>
ROCPosLimit	REAL	<p>Specifies the limit for an increasing rate-of-change in units per second. Detection is enabled for any value &gt; 0.0 if ROCPeriod is also &gt; 0.0.</p> <p>Valid = 0.0 to maximum possible float.</p> <p>Default = 0.0.</p>
ROCPosSeverity	DINT	<p>Specifies the severity of the increasing rate-of-change alarm condition.</p> <p>Valid = 1 to 1000 (1000 = most severe; 1 = least severe).</p> <p>Default = 500.</p>
ROCNegLimit	REAL	<p>Specifies the limit for a decreasing rate-of-change in units per second. Detection is enabled for any value &gt; 0.0 if ROCPeriod is also &gt; 0.0.</p> <p>Valid = 0.0 to maximum possible float.</p> <p>Default = 0.0.</p>
ROCNegSeverity	DINT	<p>Specifies the severity of the negative rate-of-change alarm condition.</p> <p>Valid = 1 to 1000 (1000 = most severe; 1 = least severe).</p> <p>Default = 500.</p>
ROCPeriod	REAL	<p>Specifies the time period in seconds for calculation of the ROC value. This value specifies the sampling interval for calculating the ROC value. Each time the sampling interval expires, the difference between the current sample and the previous sample is divided by the time interval.</p> <p>Rate-of-change detection is enabled for any value &gt; 0.0.</p> <p>Valid = 0.0 to maximum possible float.</p> <p>Default = 0.0.</p>

## Output Parameters

Output Parameter	Data Type	Description
EnableOut	BOOL	System-defined output parameter. Follows the state of EnableIn.
InAlarm	BOOL	Indicates whether any alarm condition is active.  Set – at least one Alarm condition active (InAlarm).  Cleared – all Alarm conditions inactive.
AnyInAlarmUnack	BOOL	Indicates whether any alarm condition is detected and unacknowledged.  Set – at least one Alarm condition is both active (InAlarm) and unacknowledged.  Cleared – all Alarm conditions are either inactive or acknowledged (or both).
HHInAlarm	BOOL	Indicates a high-high alarm condition.  Set - In alarm.  Cleared - Normal.
HInAlarm	BOOL	Indicates a high alarm condition.  Set - In alarm.  Cleared - Normal.
LInAlarm	BOOL	Indicates a low alarm condition.  Set - In alarm.  Cleared - Normal.
LLInAlarm	BOOL	Indicates a low-low alarm condition.  Set - In alarm.  Cleared - Normal.
ROCPosInAlarm	BOOL	Indicates whether a positive rate-of-change alarm condition is detected.  Set - In alarm.  Cleared - Normal.
ROCNegInAlarm	BOOL	Indicates whether a negative rate-of-change alarm condition is detected.  Set - In alarm.  Cleared - Normal.

Output Parameter	Data Type	Description
ROC	REAL	Indicates the calculated rate-of-change of the In value.
HHAcked	BOOL	Indicates whether the high-high alarm condition is acknowledged.  Set - Acknowledged.  Cleared - Not acknowledged.  (Always Set when AckRequired is false).
HAcked	BOOL	Indicates whether the high alarm condition is acknowledged.  Set - Acknowledged.  Cleared - Not acknowledged.  (Always Set when AckRequired is false).
LAcked	BOOL	Indicates whether the low alarm condition is acknowledged.  Set - Acknowledged.  Cleared - Not acknowledged.  (Always Set when AckRequired is false).
LLAcked	BOOL	Indicates whether the low-low alarm condition is acknowledged.  Set - Acknowledged.  Cleared - Not acknowledged.  (Always Set when AckRequired is false).
ROCPoSAcked	BOOL	Indicates whether the positive rate-of-change alarm condition is acknowledged.  Set - Acknowledged. Cleared - Not acknowledged.  (Always Set when AckRequired is false).
ROCNegAcked	BOOL	Indicates whether the negative rate-of-change alarm condition is acknowledged.  Set - Acknowledged.  Cleared - Not acknowledged.  (Always Set when AckRequired is false).



Output Parameter	Data Type	Description
HHInAlarmUnack	BOOL	<p>Indicates whether the high-high alarm is active (InAlarm) and unacknowledged.</p> <p>Set - Alarm is both active (InAlarm) and unacknowledged.</p> <p>Cleared - Alarm is either inactive or acknowledged (or both).</p>
HInAlarmUnack	BOOL	<p>Indicates whether the high alarm is active (InAlarm) and unacknowledged.</p> <p>Set - Alarm is both active and unacknowledged.</p> <p>Cleared - Alarm is either inactive or acknowledged (or both).</p>
LInAlarmUnack	BOOL	<p>Indicates whether the low alarm condition is active (InAlarm) and unacknowledged.</p> <p>Set - Alarm is both active and unacknowledged.</p> <p>Cleared - Alarm is either inactive or acknowledged (or both).</p>
LLInAlarmUnack	BOOL	<p>Indicates whether the low-low alarm condition is active (InAlarm) and unacknowledged.</p> <p>Set - Alarm is both active and unacknowledged.</p> <p>Cleared - Alarm is either inactive or acknowledged (or both).</p>
ROCPosInAlarmUnack	BOOL	<p>Indicates whether the positive rate-of-change alarm condition is active and unacknowledged.</p> <p>Set - Alarm is both active (InAlarm) and unacknowledged.</p> <p>Cleared - Alarm is either inactive or acknowledged (or both).</p>
ROCNegInAlarmUnack	BOOL	<p>Indicates whether the negative rate-of-change alarm condition is active and unacknowledged.</p> <p>Set - Alarm is both active (InAlarm) and unacknowledged.</p> <p>Cleared - Alarm is either inactive or acknowledged (or both).</p>

Output Parameter	Data Type	Description
Suppressed	BOOL	Indicates whether the alarm is suppressed.  Set - Alarm suppressed.  Cleared - Alarm unsuppressed.
Disabled	BOOL	Indicates whether the alarm is disabled.  Set - Alarm disabled.  Cleared - Alarm enabled.
MinDurationACC	DINT	Indicates the elapsed time since the first non-normal level excursion was detected. When this value reaches MinDurationPRE the pertinent alarm condition(s) become active and notification(s) are sent to clients.
HHInAlarmTime	LINT	Timestamp of high-high condition detection.
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 of high condition detection.
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 of low condition detection.
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 of low-low condition detection.
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 of positive rate-of-change condition detection.
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 of negative rate-of-change condition detection.
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.

Output Parameter	Data Type	Description
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.
DeliveryER	BOOL	Indicates alarm notification message delivery error:  Set – delivery error – either no alarm subscriber was subscribed or at least one subscriber did not receive the latest alarm change state message.  Cleared – delivery successful or in progress.
DeliveryDN	BOOL	Indicates alarm notification message delivery success:  Set – delivery success – at least one subscriber was subscribed and all subscribers received the latest alarm change state message successfully.  Cleared – delivery not completed successfully or in progress.
DeliveryEN	BOOL	Indicates alarm notification message delivery in process:  Set – delivery in progress.  Cleared – delivery not in progress.
NoSubscriber	BOOL	Indicates that the alarm had no subscribers when attempting to deliver the most recent state change message:  Set – no subscribers.  Cleared – At least one subscriber.
NoConnection	BOOL	Indicates that all of the alarm's subscribers were disconnected when attempting to deliver the most recent state change message:  Set – all subscribers disconnected.  Cleared – at least one subscriber connected.

Output Parameter	Data Type	Description
CommError	BOOL	<p>Indicates that there was a communication error when delivering last alarm message to at least one subscriber:</p> <p>Set – communication errors – all retries exhausted.</p> <p>Cleared – all connected subscribers successfully received alarm message.</p> <p>This error means that a subscriber was subscribed, and it had a connection opened, but the message was not delivered successfully.</p>
AlarmBuffered	BOOL	<p>Indicates that the alarm message was buffered when not delivered to subscriber(s), either due to a CommError or a lost Connection:</p> <p>Set – alarm message buffered for at least one subscriber.</p> <p>Cleared – alarm message is not buffered.</p>
Subscribers	DINT	Indicates number of subscribers for this alarm.
SubscNotified	DINT	Indicates number of subscribers successfully notified about the most recent alarm state change.
Status	DINT	Indicates the bit-mapped status of the instruction execution.
InstructFault	BOOL	The instruction detected an execution error. This is not a minor or major controller error. Check the remaining status bits to determine what occurred.
InFaulted	BOOL	User program has set InFault to indicate bad quality input data.
SeverityInv	BOOL	Indicates invalid alarm severity configuration. If severity <1, the instruction uses Severity = 1. If severity >1000, the instruction uses Severity = 1000.
AlarmLimitsInv	BOOL	Limits invalid (for example, LLimit < LLLimit). When this condition is detected, the instruction clears all level conditions active bit(s). Until the fault is cleared, no new level conditions can be detected.
DeadbandInv	BOOL	When this condition is detected, the instruction uses Deadband = 0.0.

Output Parameter	Data Type	Description
ROCPosLimitInv	BOOL	When this condition is detected, the instruction uses ROCPosLimit = 0.0.
ROCNegLimitInv	BOOL	When this condition is detected, the instruction uses ROCNegLimit = 0.0.
ROCPeriodInv	BOOL	When this condition is detected, the instruction uses ROCPeriod = 0.0.

cuu duong than cong. com

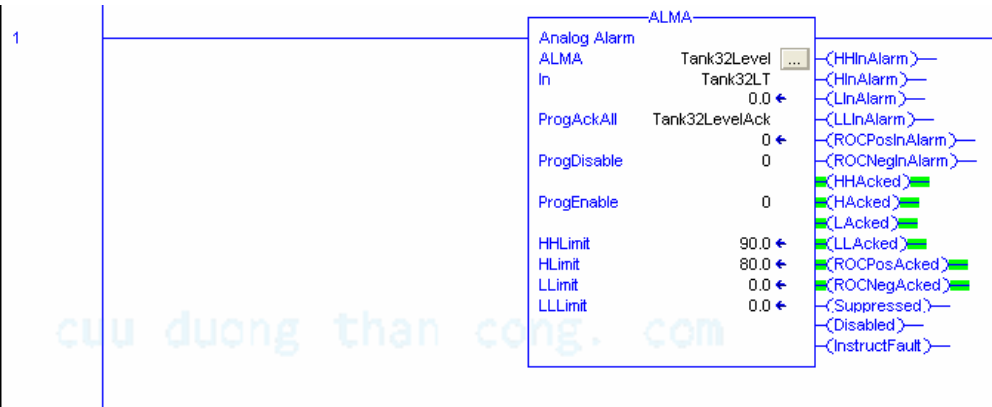
cuu duong than cong. com

# Example

This illustration shows the manner in which an analog alarm executes in a typical system configuration.

Alarm execution is shown below. In these examples, level in a tank is monitored, and an alarm is activated if the level surpasses a high or high-high limit. A programmatic acknowledge is sometimes used to acknowledge all the level alarms.

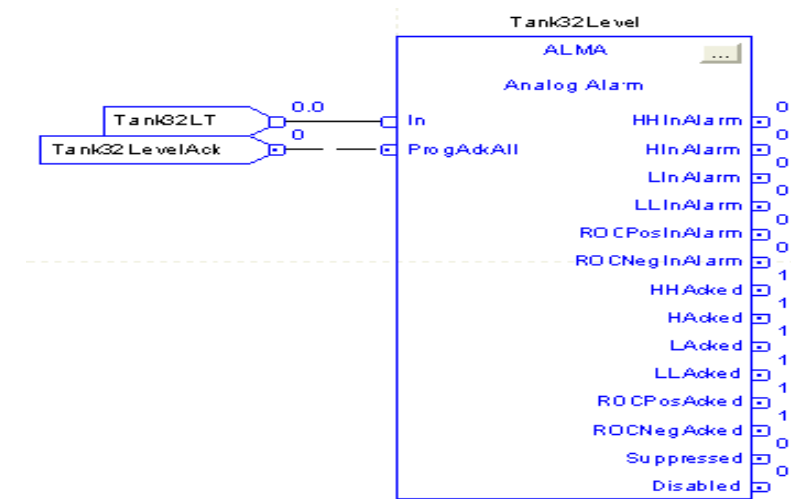
## Ladder Logic



## Structured Text

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

Function Block



cuu duong than cong. com

cuu duong than cong. com

## Execution

The tables below show execution action for Ladder Logic and Function Block programming languages.

### Ladder Logic

Condition	Action
prescan	The rung-condition-out is set to false. All operator requests, timestamps, and delivery flags are cleared. All alarm conditions are set to OutOfAlarm and Acknowledged.
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.

### Function Block

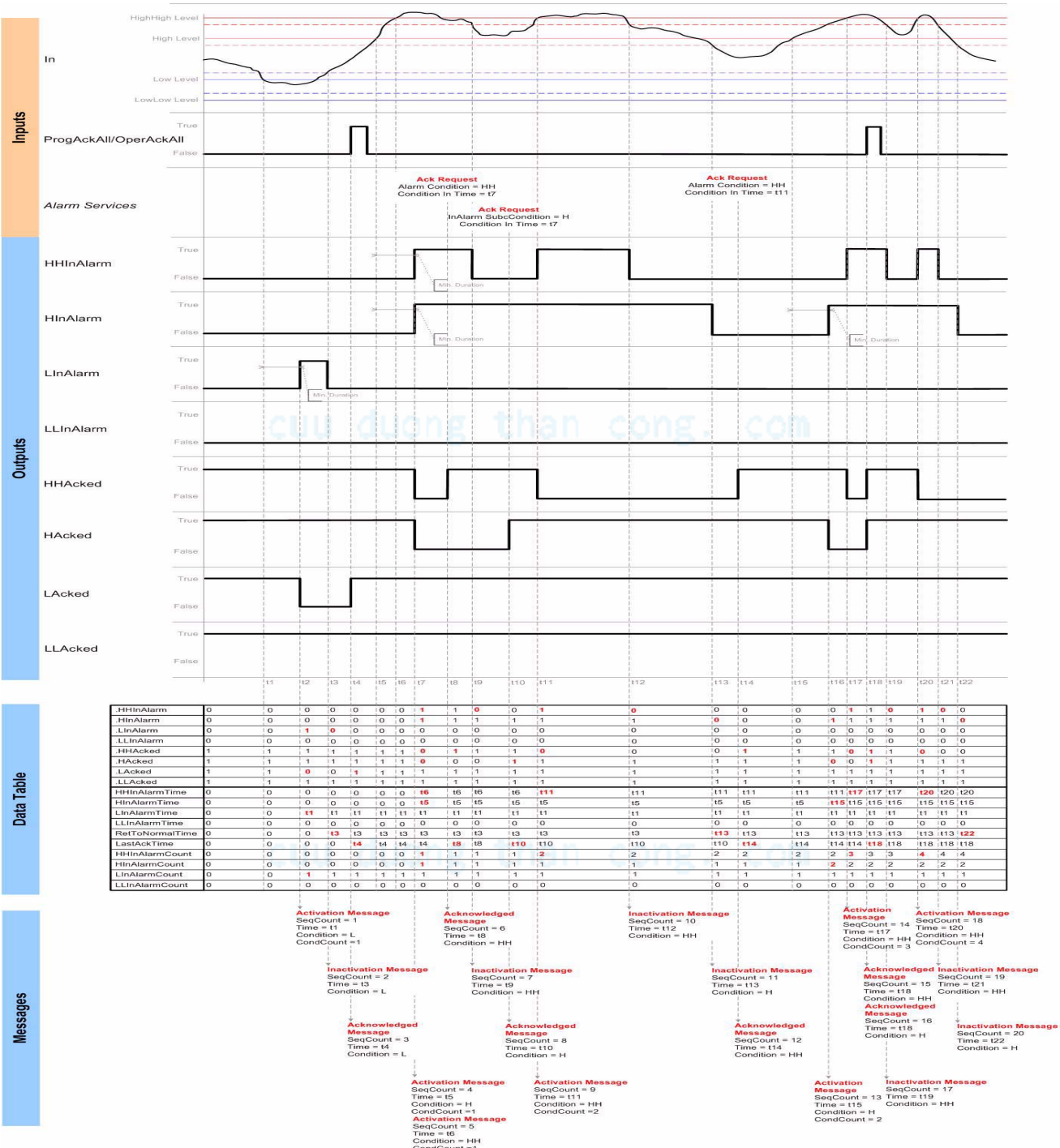
Condition	Action
prescan	All operator requests, timestamps, and delivery flags are cleared. All alarm conditions are set to OutOfAlarm and Acknowledged.
instruction first scan	No action taken.
instruction first run	No action taken.
EnableIn is cleared	The instruction does not execute. EnableOut is cleared.
EnableIn is set	The instruction executes. EnableOut is set.
postscan	No action taken.



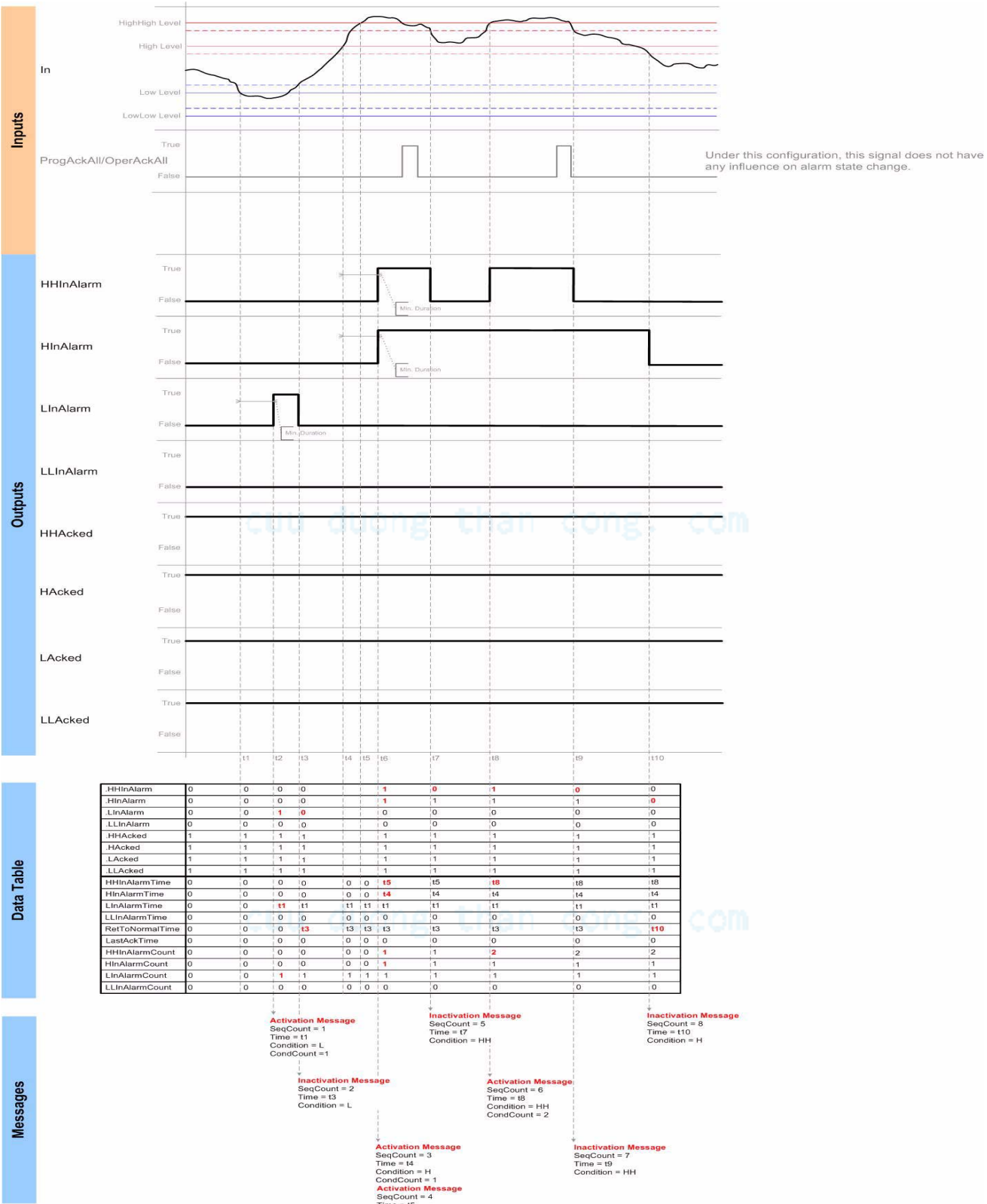
# Analog State Timing Diagrams

These timing diagrams show the sequence of bit operations in a typical system configuration.

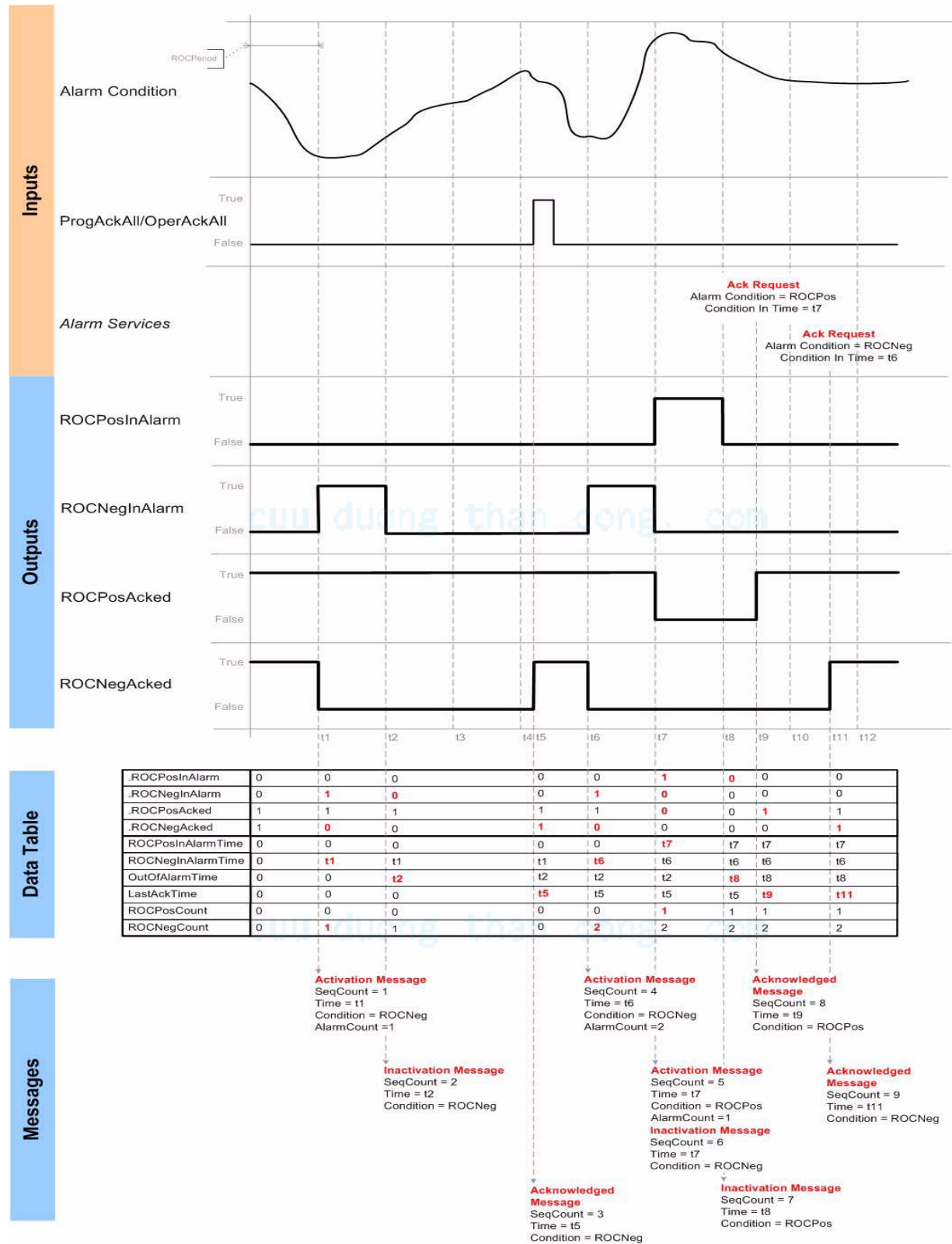
## Alarm Level Condition Acknowledge Required



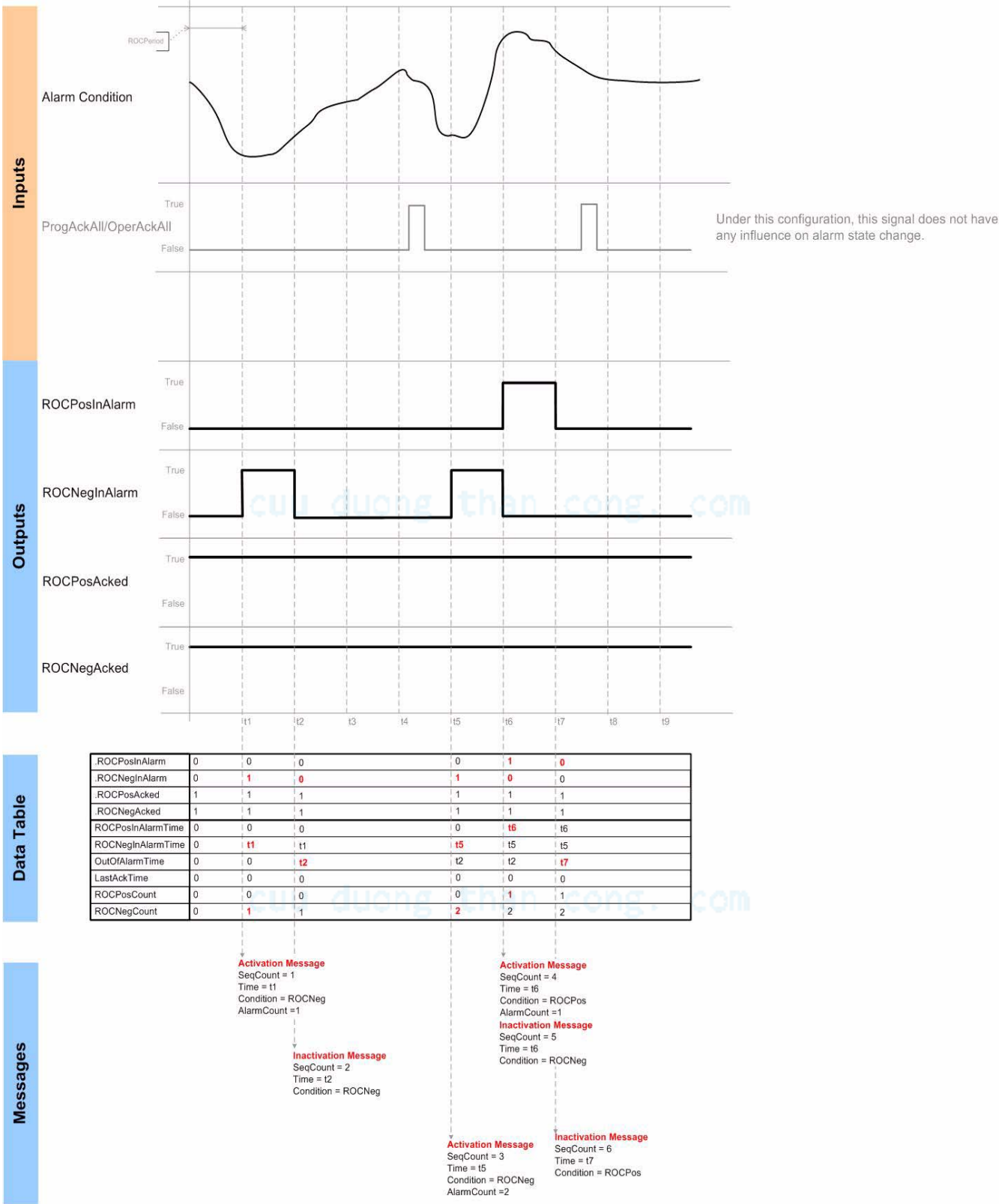
# Alarm Level Condition Acknowledge Not Required



## Alarm Rate of Change Acknowledge Required



# Alarm Rate of Change Acknowledge Not Required



## Bit Instructions

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

### Introduction

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

If You Want To	Use This Instruction	Available In These Languages	See Page
enable outputs when a bit is set	XIC	relay ladder structured text <sup>(1)</sup>	70
enable outputs when a bit is cleared	XIO	relay ladder structured text <sup>(1)</sup>	72
set a bit	OTE	relay ladder structured text <sup>(1)</sup>	74
set a bit (retentive)	OTL	relay ladder structured text <sup>(1)</sup>	76
clear bit (retentive)	OTU	relay ladder structured text <sup>(1)</sup>	78
enable outputs for one scan each time a rung goes true	ONS	relay ladder structured text <sup>(1)</sup>	80
set a bit for one scan each time a rung goes true	OSR	relay ladder	83
set a bit for one scan each time the rung goes false	OSF	relay ladder	86
set a bit for one scan each time the input bit is set in function block	OSRI	structured text function block	3-89
set a bit for one scan each time the input bit is cleared in function block	OSFI	structured text function block	92

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

# Examine If Closed (XIC)

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

## Operands:



## Relay Ladder

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



## Structured Text

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

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

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

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

**Arithmetic Status Flags:** not affected

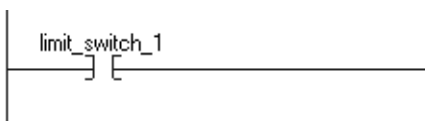
**Fault Conditions:** none

## Execution:

Condition	Relay Ladder Action
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	<pre> graph TD     Start(( )) --&gt; Exam{examine data bit}     Exam -- "data bit = 0" --&gt; SetFalse[rung-condition-out is set to false]     Exam -- "data bit = 1" --&gt; SetTrue[rung-condition-out is set to true]     SetFalse --&gt; End((end))     SetTrue --&gt; End                     </pre>
postscan	The rung-condition-out is set to false.

**Example 1:** If *limit\_switch\_1* is set, this enables the next instruction (the rung-condition-out is true).

### Relay Ladder



### Structured Text

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

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

### Relay Ladder



### Structured Text

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

# Examine If Open (XIO)

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

## Operands:



## Relay Ladder

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



## Structured Text

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

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

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

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

**Arithmetic Status Flags:** not affected

**Fault Conditions:** none

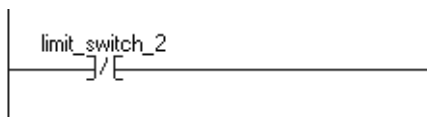
## Execution:

Condition	Relay Ladder Action
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	<pre> graph TD     Start(( )) --&gt; Examine{examine data bit}     Examine -- "data bit = 0" --&gt; SetTrue[rung-condition-out is set to true]     Examine -- "data bit = 1" --&gt; SetFalse[rung-condition-out is set to false]     SetTrue --&gt; End((end))     SetFalse --&gt; End                     </pre>
postscan	The rung-condition-out is set to false.



**Example 1:** If *limit\_switch\_2* is cleared, this enables the next instruction (the rung-condition-out is true).

### Relay Ladder

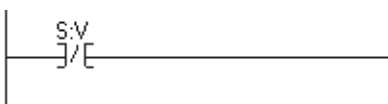


### Structured Text

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

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

### Relay Ladder



### Structured Text

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

## Output Energize (OTE)

The OTE instruction sets or clears the data bit.

### Operands:



### Relay Ladder

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



### Structured Text

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

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

See Appendix B 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

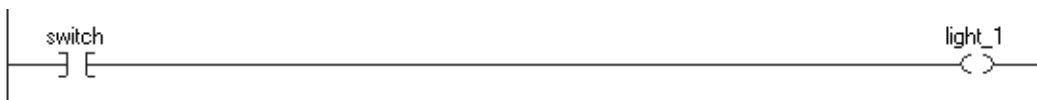
**Fault Conditions:** none

### 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;
```

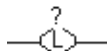
cuu duong than cong. com

cuu duong than cong. com

# Output Latch (OTL)

The OTL instruction sets (latches) the data bit.

## Operands:



## Relay Ladder

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



## Structured Text

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

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

See Appendix B 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

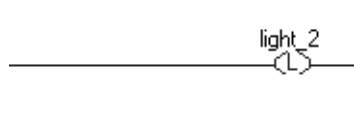
**Fault Conditions:** none

## 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;
```

cuu duong than cong. com

cuu duong than cong. com

## Output Unlatch (OTU)

The OTU instruction clears (unlatches) the data bit.

### Operands:



### Relay Ladder

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



### Structured Text

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

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

See Appendix B 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;
```

cuu duong than cong. com

cuu duong than cong. com

## One Shot (ONS)

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

### Operands:



—[<sup>?</sup>ONS]—

### Relay Ladder

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



### Structured Text

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

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

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

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

**Arithmetic Status Flags:** not affected

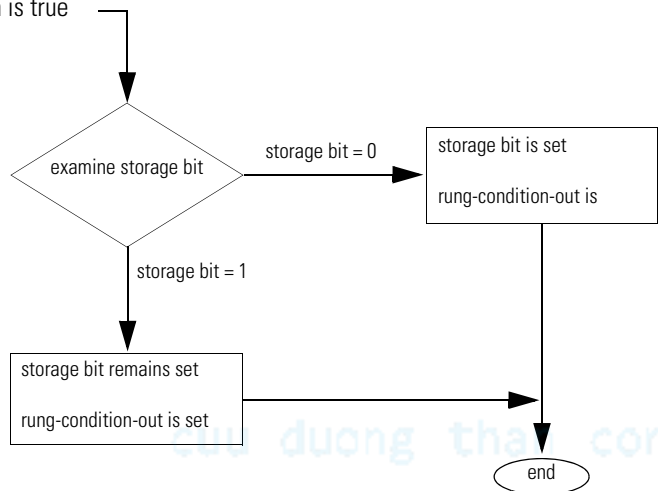
**Fault Conditions:** none



**Execution:**

Condition	Relay Ladder Action
prescan	The storage bit is set to prevent an invalid trigger during the first scan. The rung-condition-out is set to false.
rung-condition-in is false	The storage bit is cleared. The rung-condition-out is set to false.

rung-condition-in is true



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;

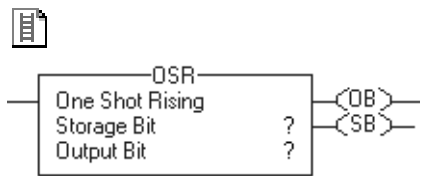
```

# 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 3-89.

## Operands:

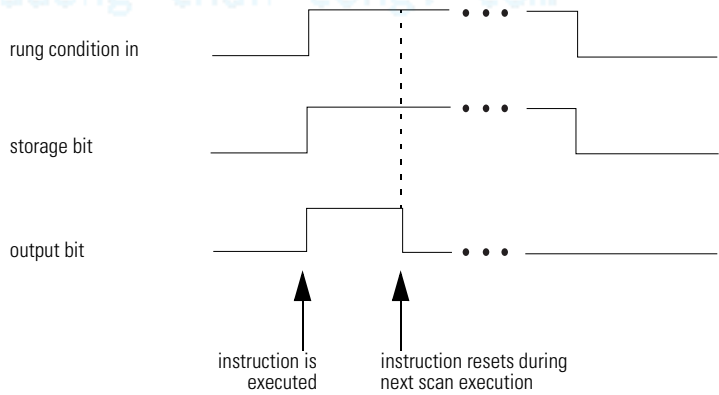


## Relay Ladder

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

cuu duong than cong. com

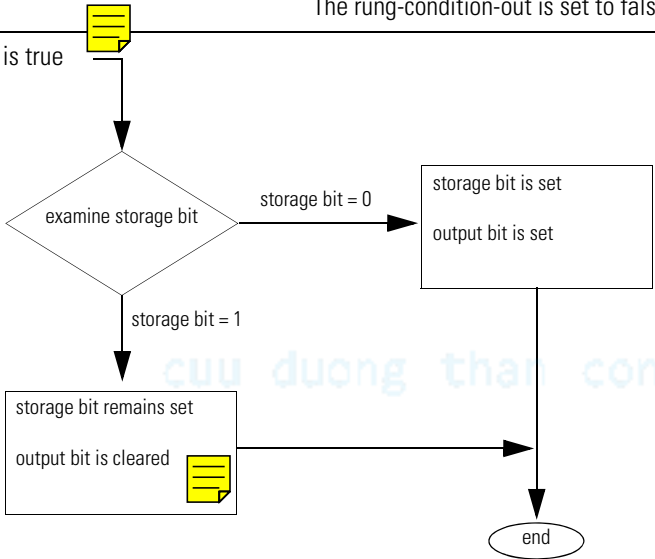


**Arithmetic Status Flags:** not affected

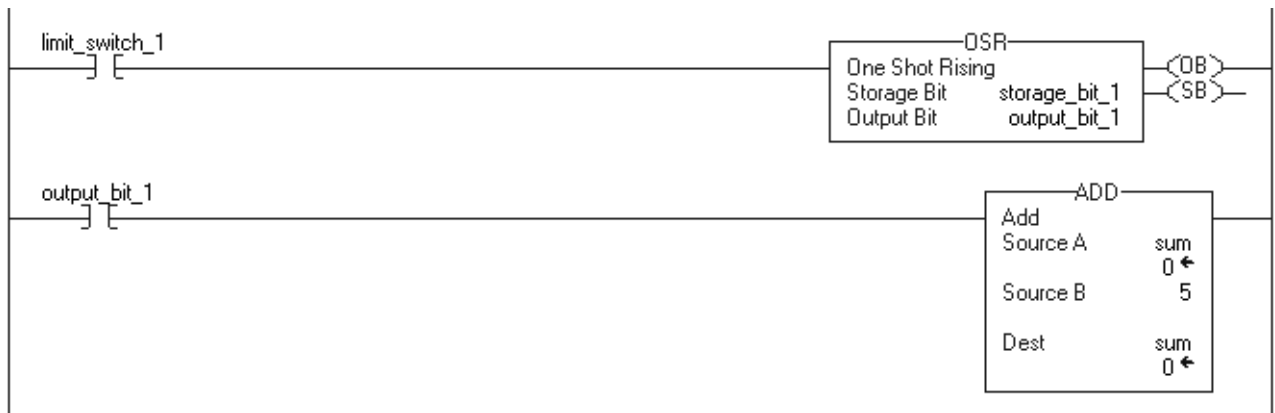
**Fault Conditions:** none

cuu duong than cong. com

**Execution:**

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

**Example:** Each time *limit\_switch\_1* goes from cleared to set, the OSR instruction sets *output\_bit\_1* and the ADD instruction increments *sum* by 5. As long as *limit\_switch\_1* stays set, *sum* stays the same value. The *limit\_switch\_1* must go from cleared to set again for *sum* to be incremented again. You can use *output\_bit\_1* on multiple rungs to trigger other operations



cuu duong than cong. com

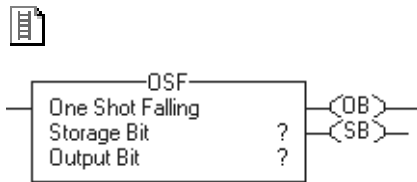
cuu duong than cong. com

# 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 3-92.

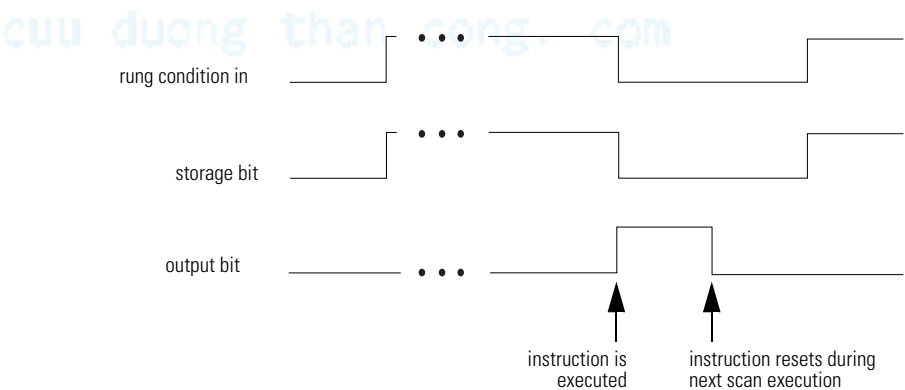
## Operands:



## 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.



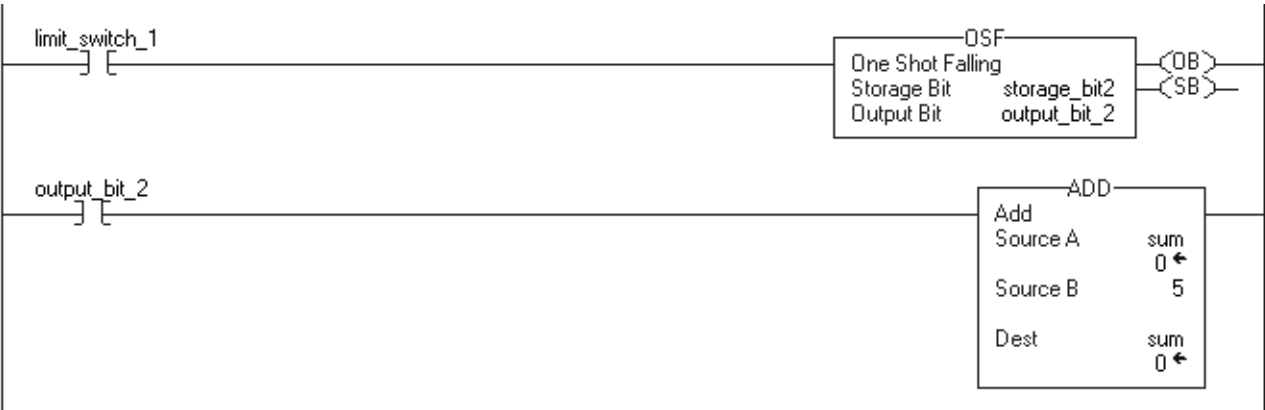
**Arithmetic Status Flags:** not affected

**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 false	<pre>graph TD; Start(( )) --&gt; Exam{examine storage bit}; Exam -- "storage bit = 0" --&gt; Box1[storage bit remains cleared output bit is cleared]; Exam -- "storage bit = 1" --&gt; Box2[storage bit is cleared output bit is set]; Box1 --&gt; End((end)); Box2 --&gt; End;</pre>
rung-condition-in is true	The storage bit is set.  The output bit is cleared.  The rung-condition-out is set to true.
postscan	See rung-condition-in is false above.

**Example:** Each time *limit\_switch\_1* goes from set to cleared, the OSF instruction sets *output\_bit\_2* and the ADD instruction increments *sum* by 5. As long as *limit\_switch\_1* stays cleared, *sum* stays the same value. The *limit\_switch\_1* must go from set to cleared again for *sum* to be incremented again. You can use *output\_bit\_2* on multiple rungs to trigger other operations.



cuu duong than cong. com

cuu duong than cong. com



## 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 3-83.

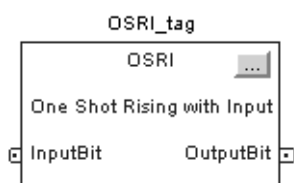
### Operands:



OSRI(OSRI\_tag) ;

### Structured Text

Operand	Type	Format	Description
OSRI tag	FBD_ONESHOT	structure	OSRI structure



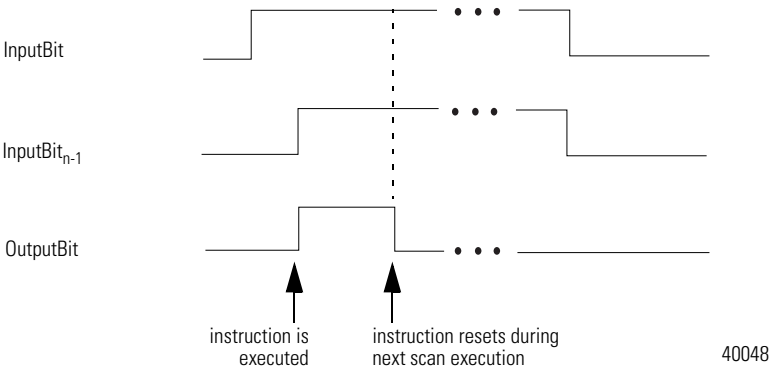
### Function Block

Operand	Type	Format	Description
OSRI tag	FBD_ONESHOT	structure	OSRI structure

### FBD\_ONESHOT Structure

Input Parameter	Data Type	Description
EnableIn	BOOL	<b>Function Block:</b>  If cleared, the instruction does not execute and outputs are not updated.  If set, the instruction executes.  Default is set.
InputBit	BOOL	<b>Structured Text:</b>  No effect. The instruction executes.  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<sub>n-1</sub> is cleared, the OSRI instruction sets OutputBit. When InputBit<sub>n-1</sub> 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 <sub>n-1</sub> is set.	InputBit <sub>n-1</sub> is set.
instruction first run	InputBit <sub>n-1</sub> is set.	InputBit <sub>n-1</sub> is set.
EnableIn is cleared	EnableOut is cleared, the instruction does nothing, and the outputs are not updated.	na
EnableIn is set	On a cleared to set transition of InputBit, the instruction sets InputBit <sub>n-1</sub> .	On a cleared to set transition of InputBit, the instruction sets InputBit <sub>n-1</sub> .
	The instruction executes.	EnableIn is always set.
	EnableOut is 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.

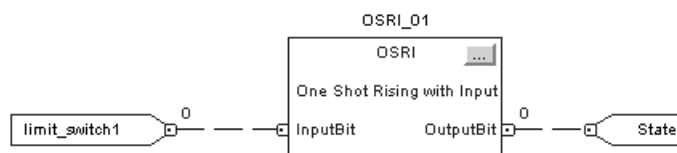
### Structured Text

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

```
OSRI(OSRI_01);
```

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

### Function Block



cuu duong than cong. com

cuu duong than cong. com

# 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 3-86.

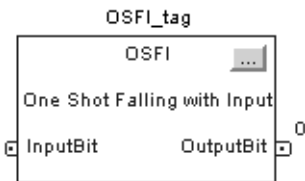
## Operands:



OSFI (OSFI\_tag) ;

## Structured Text

Operand	Type	Format	Description
OSFI tag	FBD_ONESHOT	structure	OSFI structure



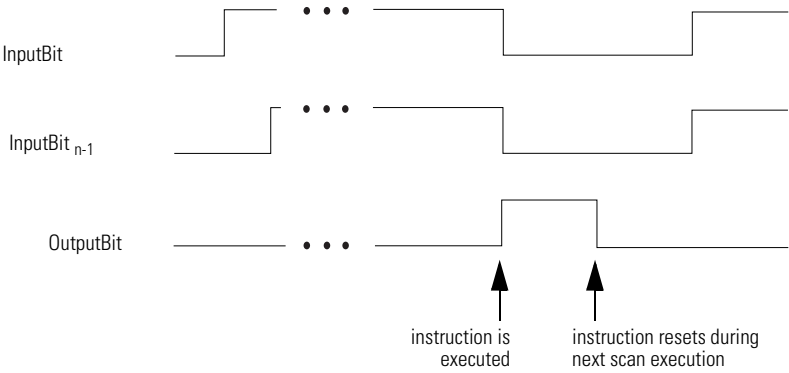
## Function Block

Operand	Type	Format	Description
OSFI tag	FBD_ONESHOT	structure	OSFI structure

## FBD\_ONESHOT Structure

Input Parameter	Data Type	Description
EnableIn	BOOL	<b>Function Block:</b>  If cleared, the instruction does not execute and outputs are not updated.  If set, the instruction executes.  Default is set.
InputBit	BOOL	<b>Structured Text:</b>  No effect. The instruction executes.  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<sub>n-1</sub> is set, the OSFI instruction sets the OutputBit. When InputBit<sub>n-1</sub> is cleared or when InputBit is set, the OSFI instruction clears the OutputBit.



40047

**Arithmetic Status Flags:** not affected

**Fault Conditions:** none

**Execution:**

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

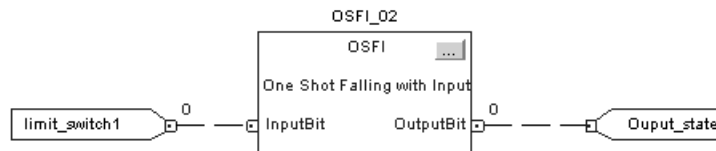
### Structured Text

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

```
OSFI(OSFI_01);
```

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

### Function Block



cuu duong than cong. com

cuu duong than cong. com

## Timer and Counter Instructions

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

### Introduction

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

If You Want To	Use This Instruction	Available In These Languages	See Page
time how long a timer is enabled	TON	relay ladder	96
time how long a timer is disabled	TOF	relay ladder	100
accumulate time	RTO	relay ladder	105
time how long a timer is enabled with built-in reset in function block	TONR	structured text function block	110
time how long a timer is disabled with built-in reset in function block	TOFR	structure text function block	114
accumulate time with built-in reset in function block	RTOR	structured text function block	118
count up	CTU	relay ladder	123
count down	CTD	relay ladder	127
count up and count down in function block	CTUD	structured text function block	131
reset a timer or counter	RES	relay ladder	136

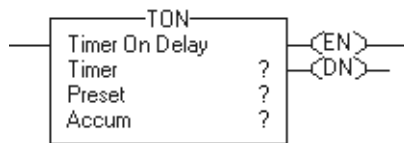
The time base for all timers is 1 msec.

## Timer On Delay (TON)

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

This instruction is available in structured text and function block as TONR, see page 4-110.

### Operands:



### Relay Ladder

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

### TIMER Structure

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

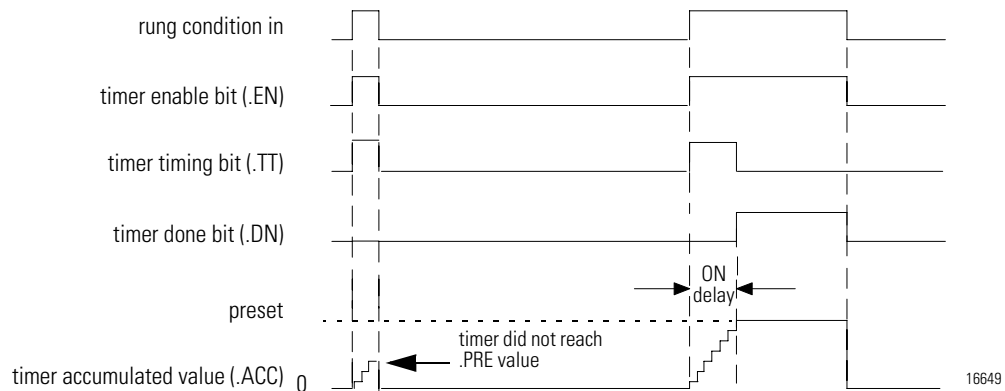
**Description:** The TON instruction accumulates time until:

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

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



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



### How a timer runs

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

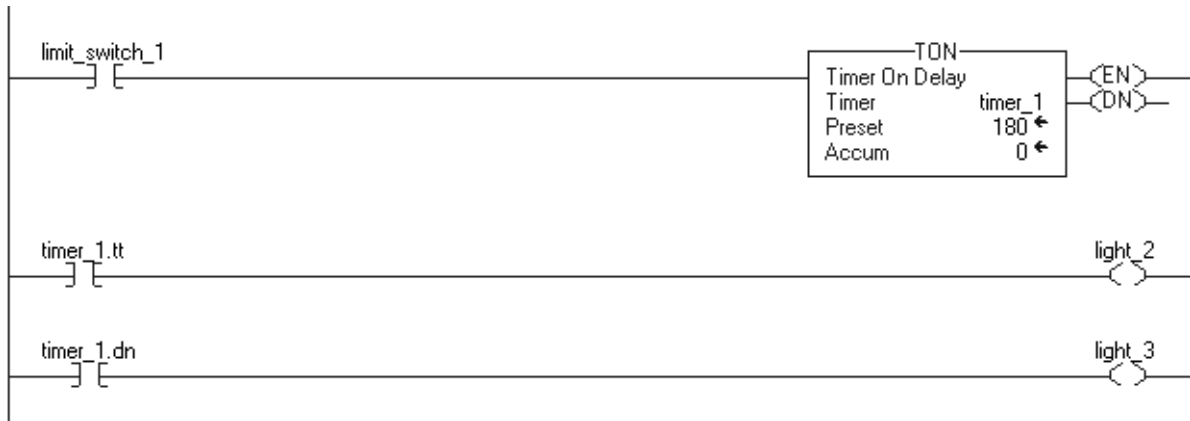
#### Fault Conditions:

A Major Fault Will Occur If	Fault Type	Fault Code
.PRE < 0	4	34
.ACC < 0	4	34

**Execution:**

Condition	Relay Ladder Action
prescan	<p>The .EN, .TT, and .DN bits are cleared.</p> <p>The .ACC value is cleared.</p> <p>The rung-condition-out is set to false.</p>
rung-condition-in is false	<p>The .EN, .TT, and .DN bits are cleared.</p> <p>The .ACC value is cleared.</p> <p>The rung-condition-out is set to false.</p>
rung-condition-in is true	<pre> graph TD     Start([start]) --&gt; DN1{examine .DN bit}     DN1 -- ".DN bit = 1" --&gt; DNSet[.DN is set .TT bit is cleared]     DN1 -- ".DN bit = 0" --&gt; EN1{examine .EN bit}     EN1 -- ".EN bit = 0" --&gt; ENSet[.EN bit is set .TT bit is set]     EN1 -- ".EN bit = 1" --&gt; TTSet[.TT bit is set .ACC = .ACC + (current_time - last_time)]     TTSet --&gt; ACCRoll{.ACC value rolls over}     ACCRoll -- yes --&gt; ACCReset[.ACC = 2,147,483,647]     ACCRoll -- no --&gt; ACC1{examine .ACC}     ENSet --&gt; ACC1     DNSet --&gt; RCOTrue[rung-condition-out is set to true]     ACC1 -- ".ACC ≥ .PRE" --&gt; RCOTrue     ACC1 -- ".ACC &lt; .PRE" --&gt; DNSetTTClear[.DN is set .TT bit is cleared]     DNSetTTClear --&gt; ACC1     RCOTrue --&gt; End([end])   </pre>
postscan	The rung-condition-out is set to false.

**Example:** When *limit\_switch\_1* is set, *light\_2* is on for 180 msec (*timer\_1* is timing). When *timer\_1.acc* reaches 180, *light\_2* goes off and *light\_3* goes on. *Light\_3* remains on until the TON instruction is disabled. If *limit\_switch\_1* is cleared while *timer\_1* is timing, *light\_2* goes off.



cuu duong than cong. com

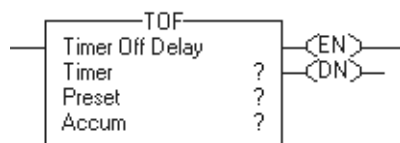
cuu duong than cong. com

## Timer Off Delay (TOF)

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

This instruction is available in structured text and function block as TOFR, see page 114.

### Operands:



### Relay Ladder

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

### TIMER Structure

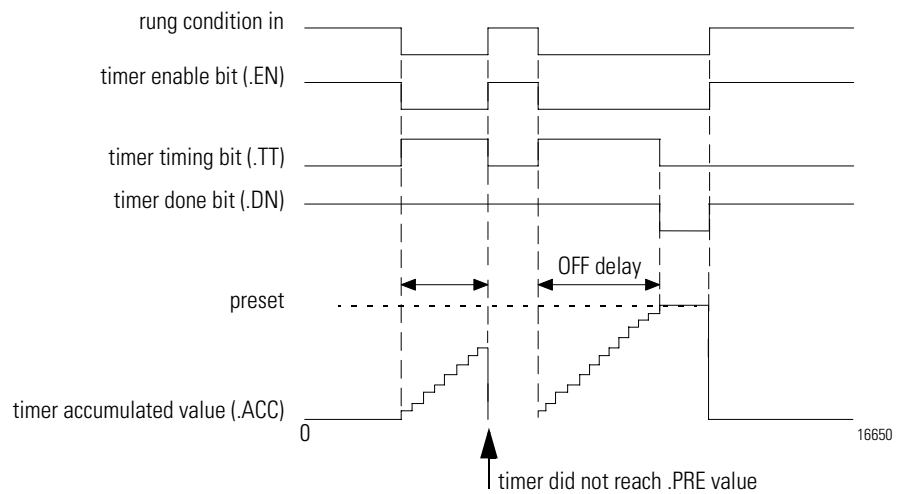
Mnemonic	Data Type	Description
.EN	BOOL	The enable bit indicates that the TOF instruction is enabled.
.TT	BOOL	The timing bit indicates that a timing operation is in process
.DN	BOOL	The done bit is cleared when .ACC ≥ .PRE.
.PRE	DINT	The preset value specifies the value (1 msec units) which the accumulated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

**Description:** The TOF instruction accumulates time until:

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

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

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



### How a timer runs

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:**

A Major Fault Will Occur If	Fault Type	Fault Code
.PRE < 0	4	34
.ACC < 0	4	34

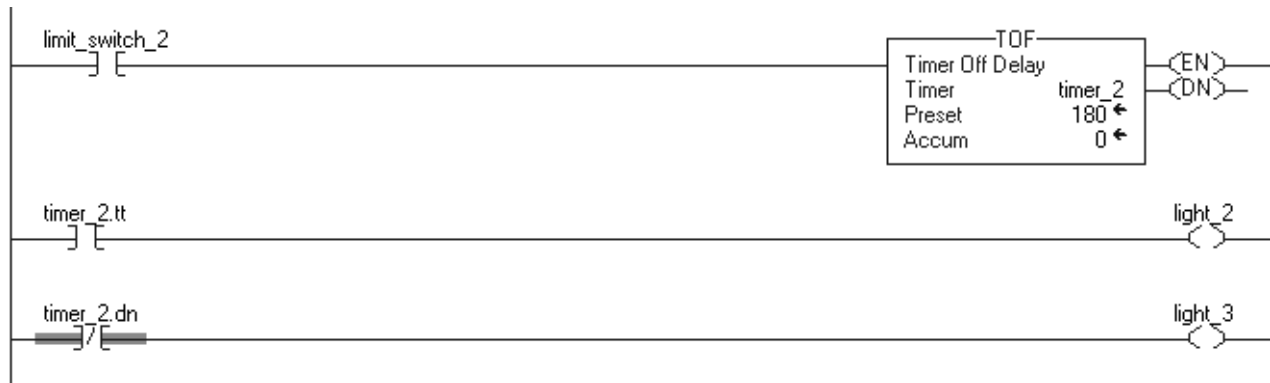
**Execution:**

cuu duong than cong. com

cuu duong than cong. com

Condition	Relay Ladder Action
prescan	<p>The .EN, .TT, and .DN bits are cleared.</p> <p>The .ACC value is set to equal the .PRE value.</p> <p>The rung-condition-out is set to false.</p>
<p>rung-condition-in is false</p> <pre> graph TD     Start([rung-condition-in is false]) --&gt; DN{examine .DN bit}     DN -- ".DN bit = 0" --&gt; ACC{examine .ACC}     DN -- ".DN bit = 1" --&gt; EN{examine .EN bit}     EN -- ".EN bit = 1" --&gt; SetTT[.TT bit is set .EN bit is cleared]     EN -- ".EN bit = 0" --&gt; SetTTCalc[.TT bit is set .ACC = .ACC + (current_time - last_time)]     SetTT --&gt; ACC     SetTTCalc --&gt; ACC     ACC -- ".ACC ≥ .PRE" --&gt; ClearDN[.DN is cleared .TT bit is cleared]     ACC -- ".ACC &lt; .PRE" --&gt; SetFalse[run-condition-out is set to false]     ClearDN --&gt; SetFalse     SetFalse --&gt; End([end])     SetTTCalc --&gt; Rollover{.ACC value rolls over}     Rollover -- yes --&gt; SetMax[.ACC = 2,147,483,647]     Rollover -- no --&gt; ACC     SetMax --&gt; ACC   </pre>	
rung-condition-in is true	<p>The .EN, .TT, and .DN bits are set.</p> <p>The .ACC value is cleared.</p> <p>The rung-condition-out is set to true.</p>
postscan	The rung-condition-out is set to false.

**Example:** When *limit\_switch\_2* is cleared, *light\_2* is on for 180 msec (*timer\_2* is timing). When *timer\_2.acc* reaches 180, *light\_2* goes off and *light\_3* goes on. *Light\_3* remains on until the TOF instruction is enabled. If *limit\_switch\_2* is set while *timer\_2* is timing, *light\_2* goes off.



cuu duong than cong. com

cuu duong than cong. com

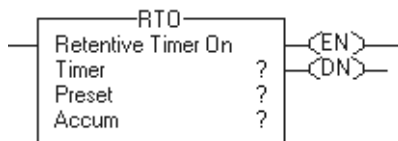


## Retentive Timer On (RTO)

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

This instruction is available in structured text and function block as RTOR, see page 118.

### Operands:



### Relay Ladder

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

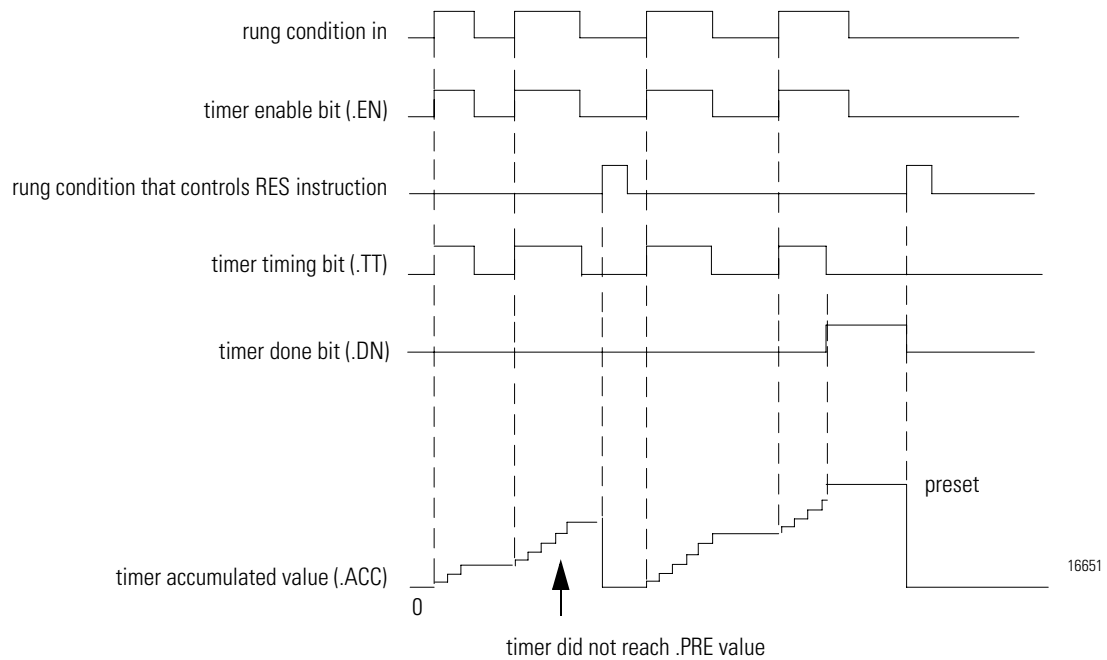
### TIMER Structure

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

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

cuu duong than cong. com

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



### How a Timer Runs

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:**

<b>A Major Fault Will Occur If</b>	<b>Fault Type</b>	<b>Fault Code</b>
.PRE < 0	4	34
.ACC < 0	4	34

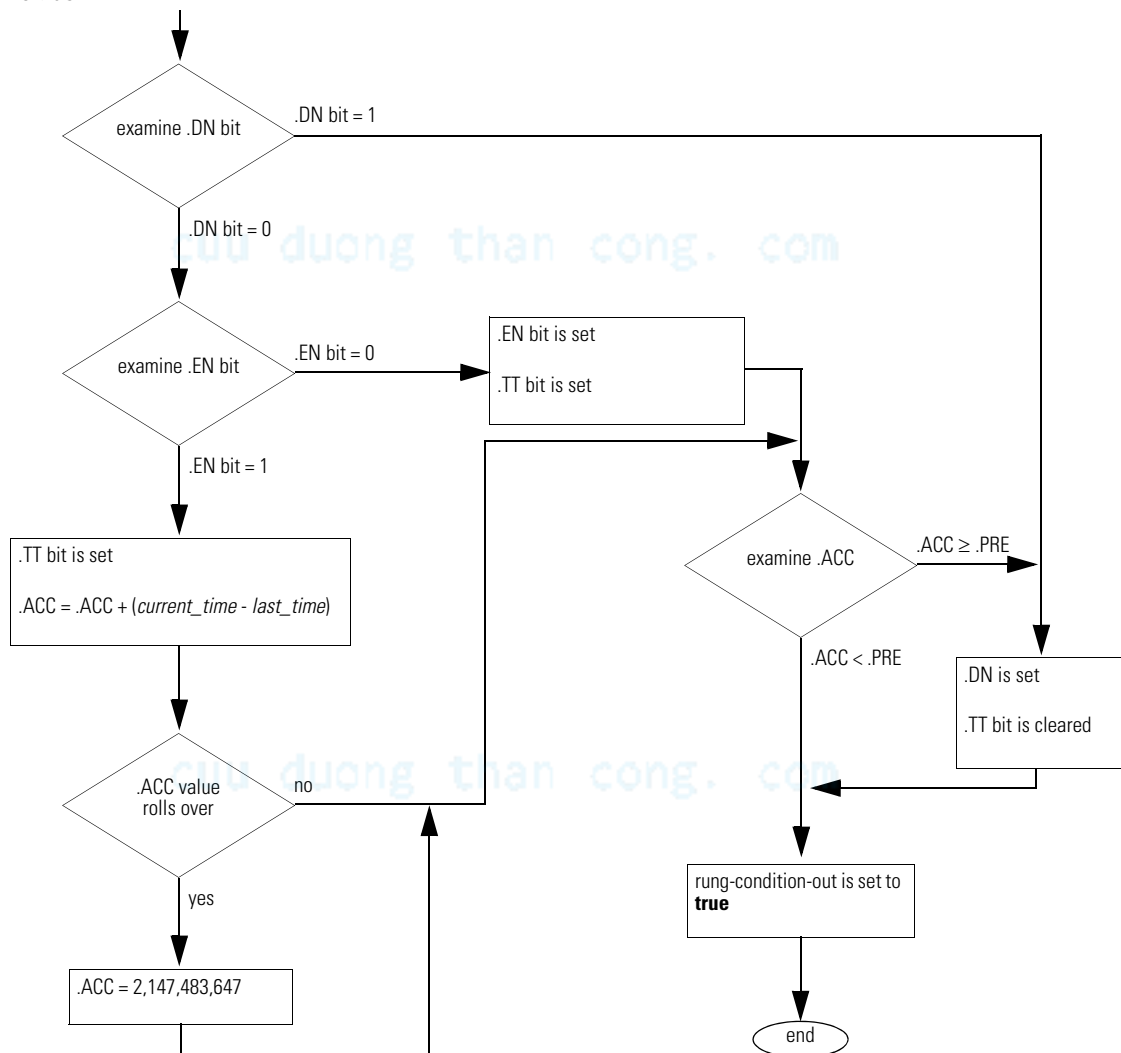
cuu duong than cong. com

cuu duong than cong. com

**Execution:**

Condition	Relay Ladder Action
prescan	<p>The .EN, .TT, and .DN bits are cleared.</p> <p>The .ACC value is not modified.</p> <p>The rung-condition-out is set to false.</p>
rung-condition-in is false	<p>The .EN and .TT bits are cleared.</p> <p>The .DN bit is not modified.</p> <p>The .ACC value is not modified.</p> <p>The rung-condition-out is set to false.</p>

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 msec (*timer\_2* is timing). When *timer\_3.acc* reaches 180, *light\_1* goes off and *light\_2* goes on. *Light\_2* remains until *timer\_3* is reset. If *limit\_switch\_2* is cleared while *timer\_3* is timing, *light\_1* remains on. When *limit\_switch\_2* is set, the RES instruction resets *timer\_3* (clears status bits and .ACC value).



cuu duong than cong. com

cuu duong than cong. com

## Timer On Delay with Reset (TONR)

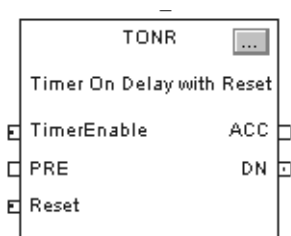
The TONR instruction is a non-retentive timer that accumulates time when TimerEnable is set.

This instruction is available in relay ladder as two separate instructions: TON (see page 4-96) and RES (see page 136).

### 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

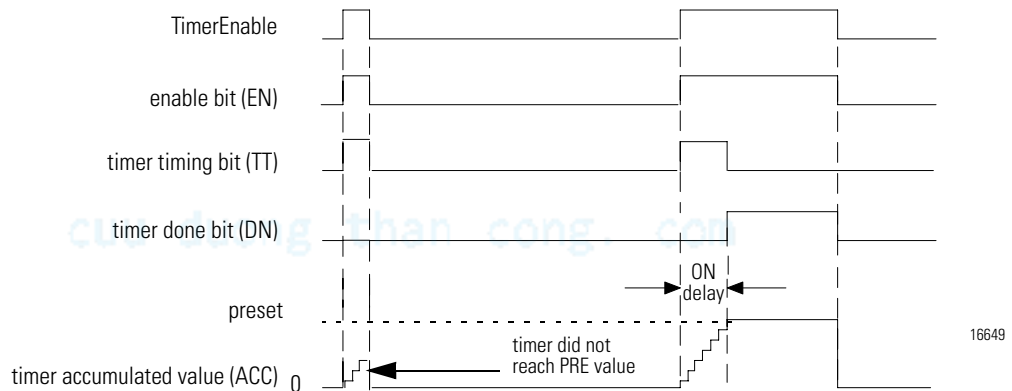
Input Parameter	Data Type	Description
EnableIn	BOOL	<b>Function Block:</b>  If cleared, the instruction does not execute and outputs are not updated.  If set, the instruction executes.  Default is set.
TimerEnable	BOOL	<b>Structured Text:</b>  No effect. The instruction executes.  If set, this enables the timer to run and accumulate time.  Default is cleared.
PRE	DINT	Timer preset value. This is the value in 1msec units that ACC must reach before timing is finished. If invalid, the instruction sets the appropriate bit in Status and the timer does not execute.  Valid = 0 to maximum positive integer
Reset	BOOL	Request to reset the timer. When set, the timer resets.  Default is cleared.
Output Parameter	Data Type	Description
EnableOut	BOOL	The instruction produced a valid result.
ACC	BOOL	Accumulated time in milliseconds.
EN	BOOL	Timer enabled output. Indicates the timer instruction is enabled.
TT	BOOL	Timer timing output. When set, a timing operation is in progress.

Input Parameter	Data Type	Description
DN	BOOL	Timing done output. Indicates when the accumulated time is greater than or equal to the preset value.
Status	DINT	Status of the function block.
InstructFault (Status.0)	BOOL	The instruction detected one of the following execution errors. This is not a minor or major controller error. Check the remaining status bits to determine what occurred.
PresetInv (Status.1)	BOOL	The preset value is invalid.

**Description:** The TONR instruction accumulates time until the:

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

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



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

### How a Timer Runs

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



**Example:** Each scan that *limit\_switch1* is set, the TONR instruction increments the ACC value by elapsed time until the ACC value reaches the PRE value. When  $ACC \geq PRE$ , the DN parameter is set, and *timer\_state* is set.

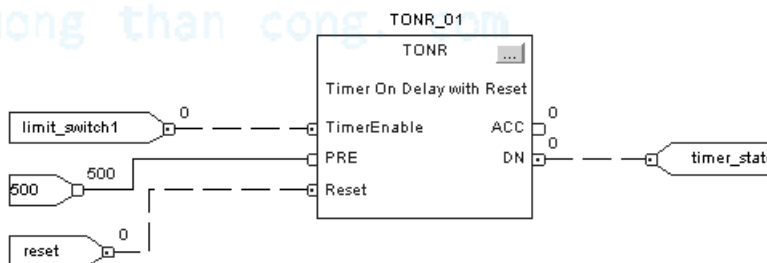
### Structured Text

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

TONR(TONR_01);

timer_state := TONR_01.DN;
```

### Function Block Example



## Timer Off Delay with Reset (TOFR)

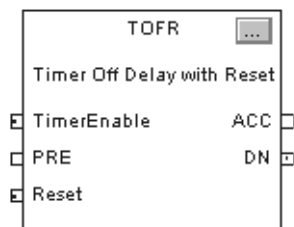
The TOFR instruction is a non-retentive timer that accumulates time when TimerEnable is cleared.

This instruction is available in relay ladder as two separate instructions: TOF (see page 4-100) and RES (see page 136).

### 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

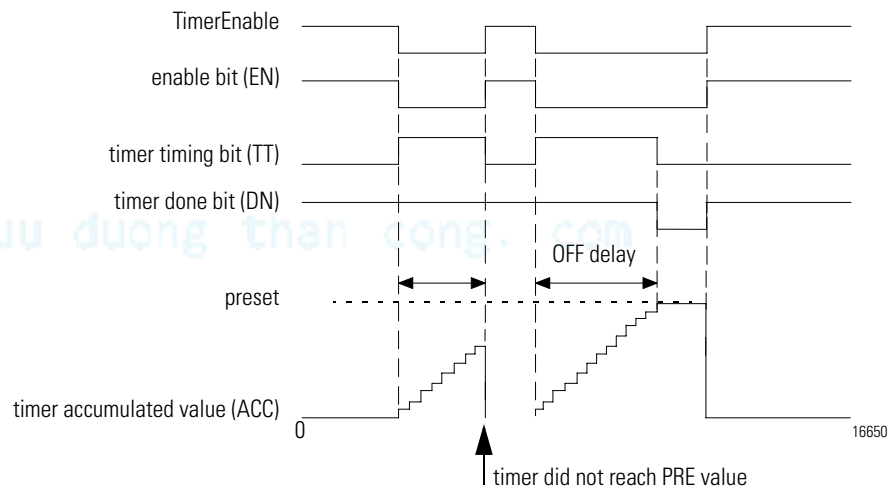
Input Parameter	Data Type	Description
EnableIn	BOOL	<b>Function Block:</b>  If cleared, the instruction does not execute and outputs are not updated.  If set, the instruction executes.  Default is set.
TimerEnable	BOOL	<b>Structured Text:</b>  No effect. The instruction executes.  If cleared, this enables the timer to run and accumulate time.  Default is cleared.
PRE	DINT	Timer preset value. This is the value in 1msec units that ACC must reach before timing is finished. If invalid, the instructions sets the appropriate bit in Status and the timer does not execute.  Valid = 0 to maximum positive integer
Reset	BOOL	Request to reset the timer. When set, the timer resets.  Default is cleared.
Output Parameter	Data Type	Description
EnableOut	BOOL	The instruction produced a valid result.
ACC	BOOL	Accumulated time in milliseconds.
EN	BOOL	Timer enabled output. Indicates the timer instruction is enabled.

Input Parameter	Data Type	Description
TT	BOOL	Timer timing output. When set, a timing operation is in progress.
DN	BOOL	Timing done output. Indicates when accumulated time is greater than or equal to preset.
Status	DINT	Status of the function block.
InstructFault (Status.0)	BOOL	The instruction detected one of the following execution errors. This is not a minor or major controller error. Check the remaining status bits to determine what occurred.
PresetInv (Status.1)	BOOL	The preset value is invalid.

**Description:** The TOFR instruction accumulates time until the:

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

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



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

### How a Timer Runs

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

**Example:** Each scan after *limit\_switch1* is cleared, the TOFR instruction increments the ACC value by elapsed time until the ACC value reaches the PRE value. When  $ACC \geq PRE$ , the DN parameter is cleared, and *timer\_state2* is set.

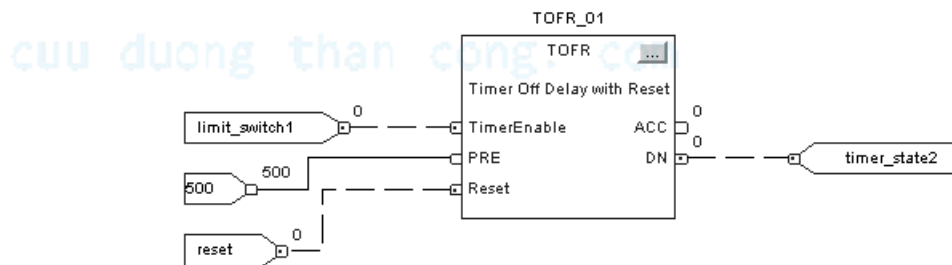
### Structured Text

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

TOFR(TOFR_01);

timer_state2 := TOFR_01.DN;
```

### Function Block



## Retentive Timer On with Reset (RTOR)

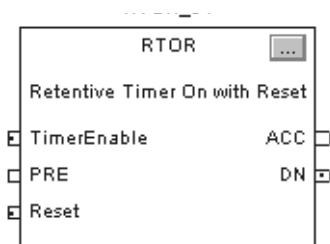
The RTOR instruction is a retentive timer that accumulates time when TimerEnable is set.

This instruction is available in relay ladder as two separate instructions: RTO (see page 4-105) and RES (see page 136).

### 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

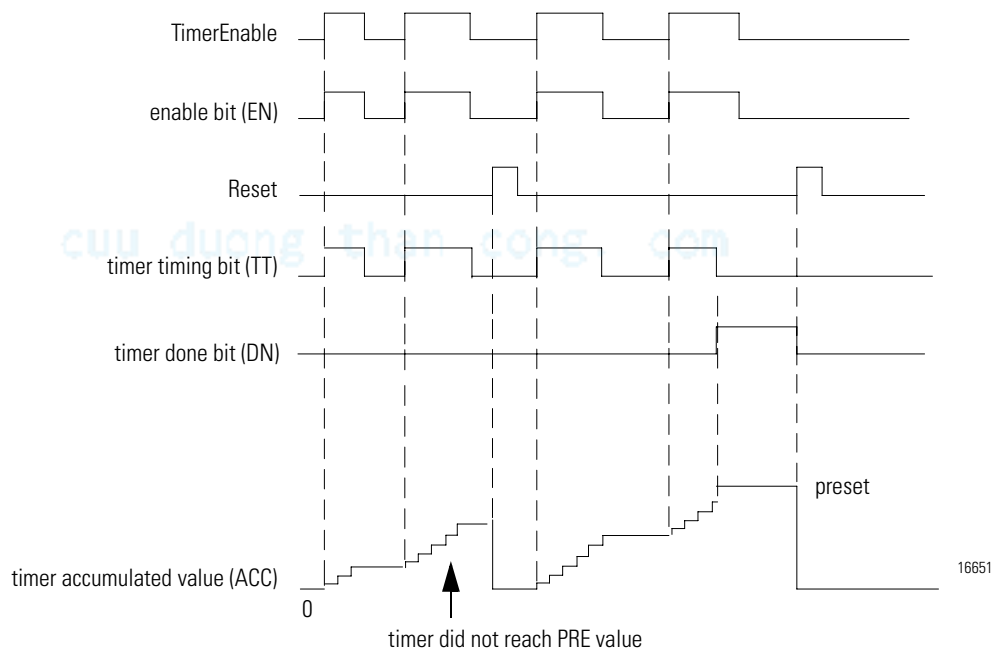
### FBD\_TIMER Structure

Input Parameter	Data Type	Description
EnableIn	BOOL	<b>Function Block:</b>  If cleared, the instruction does not execute and outputs are not updated.  If set, the instruction executes.  Default is set.
TimerEnable	BOOL	<b>Structured Text:</b>  No effect. The instruction executes.
PRE	DINT	If set, this enables the timer to run and accumulate time.  Default is cleared.  Timer preset value. This is the value in 1msec units that ACC must reach before timing is finished. If invalid, the instruction sets the appropriate bit in Status and the timer does not execute.  Valid = 0 to maximum positive integer
Reset	BOOL	Request to reset the timer. When set, the timer resets.
Output Parameter	Data Type	Description
EnableOut	BOOL	The instruction produced a valid result.
ACC	DINT	Accumulated time in milliseconds. This value is retained even while the TimerEnable input is cleared. This makes the behavior of this block different than the TONR block.
EN	BOOL	Timer enabled output. Indicates the timer instruction is enabled.
TT	BOOL	Timer timing output. When set, a timing operation is in progress.

Input Parameter	Data Type	Description
DN	BOOL	Timing done output. Indicates when accumulated time is greater than or equal to preset.
Status	DINT	Status of the function block.
InstructFault (Status.0)	BOOL	The instruction detected one of the following execution errors. This is not a minor or major controller error. Check the remaining status bits to determine what occurred.
PresetInv (Status.1)	BOOL	The preset value is invalid.

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

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



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

### How a Timer Runs

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

cuu duong than cong. com

cuu duong than cong. com



**Execution:**

Condition	Function Block Action	Structured Text Action
prescan	No action taken.	No action taken.
instruction first scan	EN, TT and DN are cleared ACC value is not modified	EN, TT and DN are cleared ACC value is not modified
instruction first run	EN, TT and DN are cleared ACC value is not modified	EN, TT and DN are cleared ACC value is not modified
EnableIn is cleared	EnableOut is cleared, the instruction does nothing, and the outputs are not updated.	na
EnableIn is set	<b>Function Block:</b>  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.

cuu duong than cong. com

**Example:** Each scan that *limit\_switch1* is set, the RTOR instruction increments the ACC value by elapsed time until the ACC value reaches the PRE value. When  $ACC \geq PRE$ , the DN parameter is set, and *timer\_state3* is set.

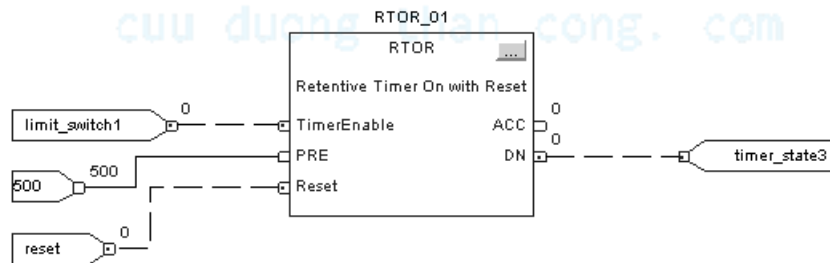
### Structured Text

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

RTOR(RTOR_01);

timer_state3 := RTOR_01.DN;
```

### Function Block

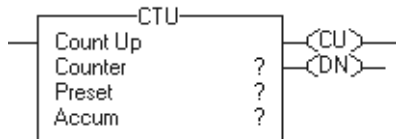


## Count Up (CTU)

The CTU instruction counts upward.

This instruction is available in structured text and function block as CTUD, see page 131.

### Operands:



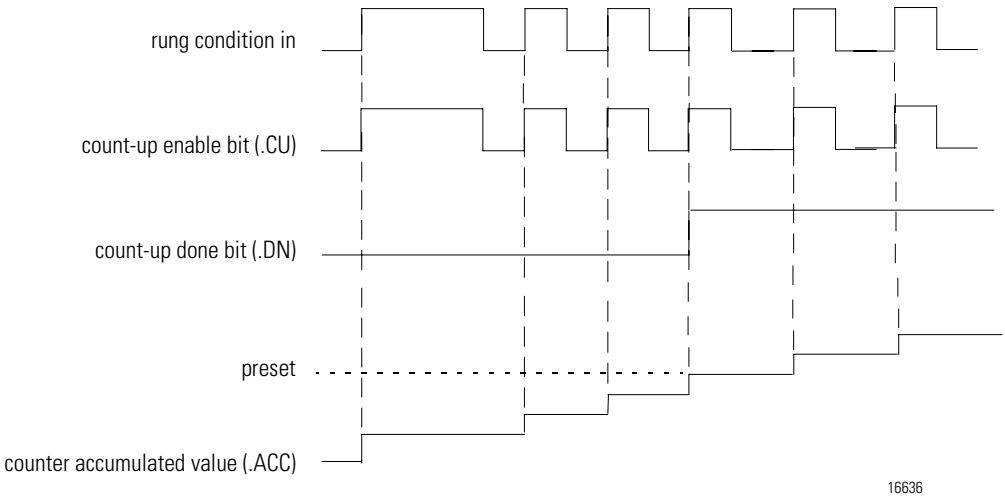
### Relay Ladder

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

### COUNTER Structure

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

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

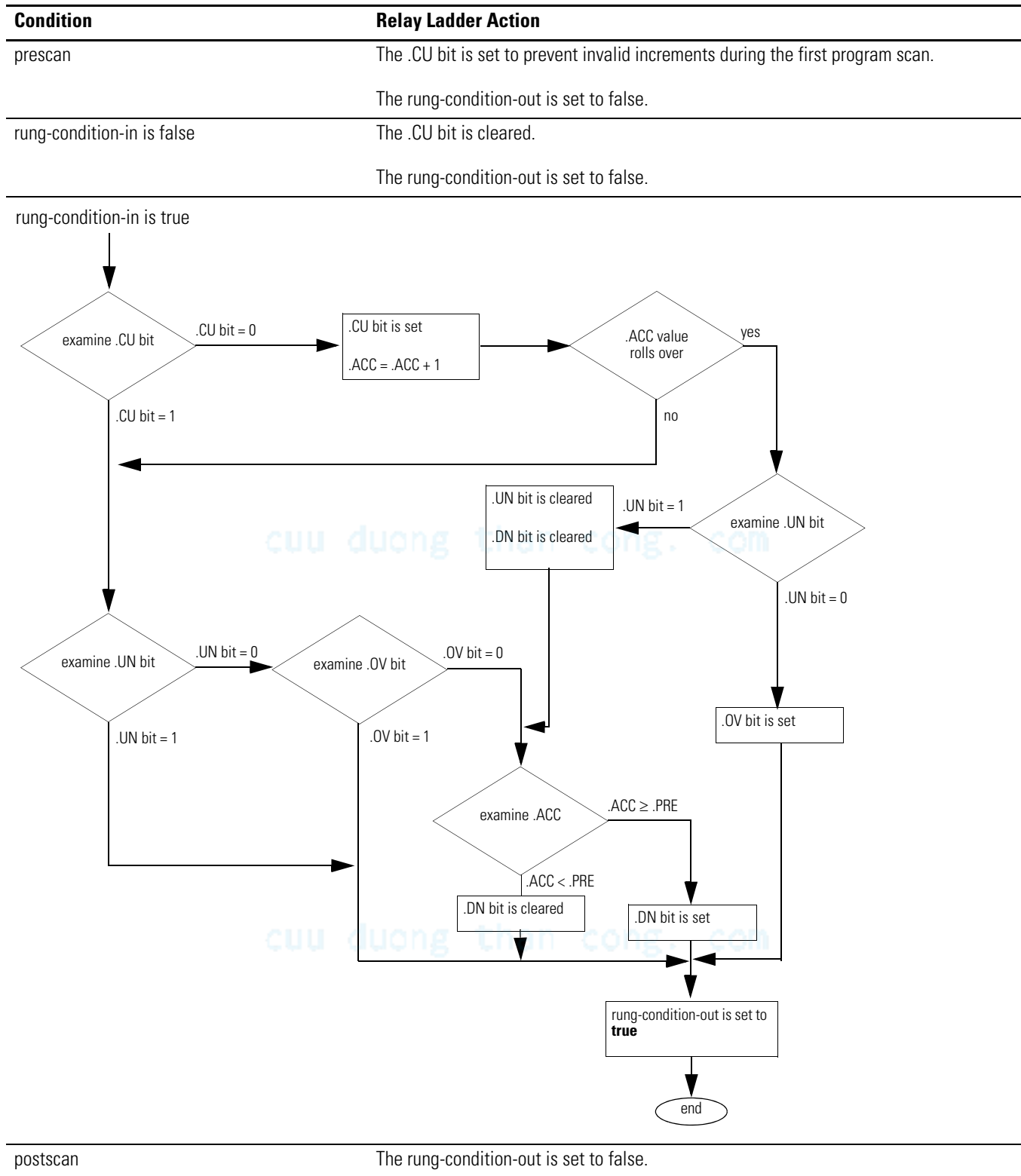


16636

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

**Arithmetic Status Flags:** not affected

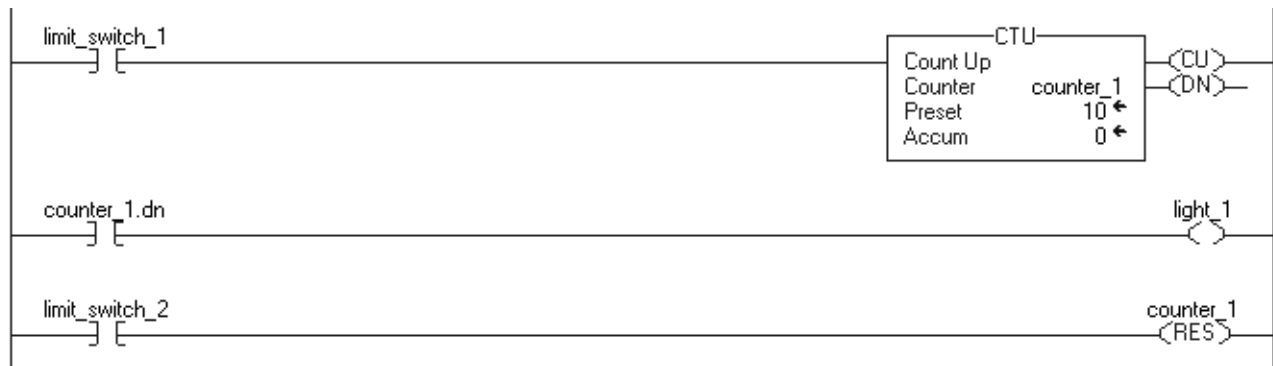
**Fault Conditions:** none

**Execution:**

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.



cuu duong than cong. com

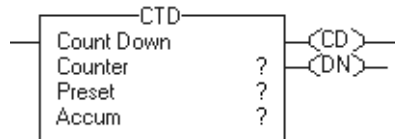
cuu duong than cong. com

## Count Down (CTD)

The CTD instruction counts downward.

This instruction is available in structured text and function block as CTUD, see page 131.

### Operands:



### Relay Ladder

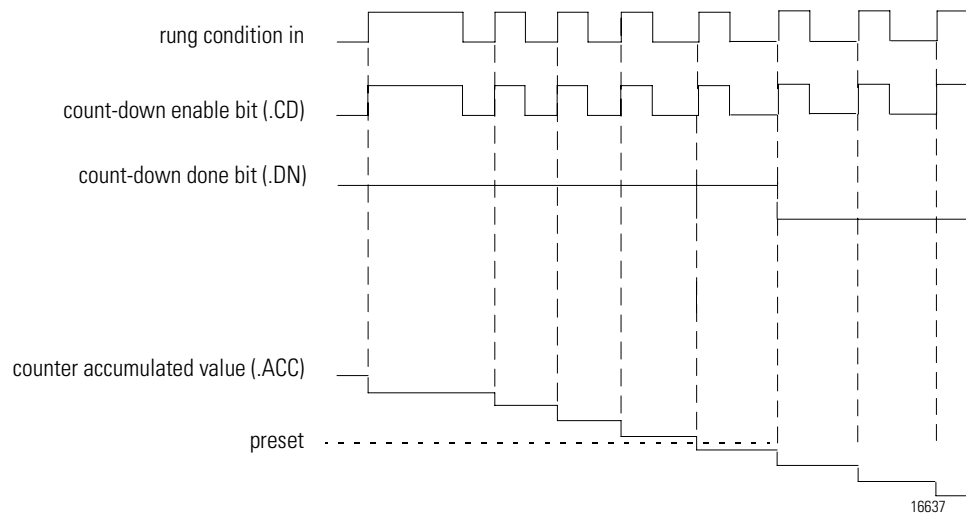
Operand	Type	Format	Description
Counter	COUNTER	tag	counter structure
Preset	DINT	immediate	how low to count
Accum	DINT	immediate	number of times the counter has counted
initial value is typically 0			

### COUNTER Structure

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

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

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



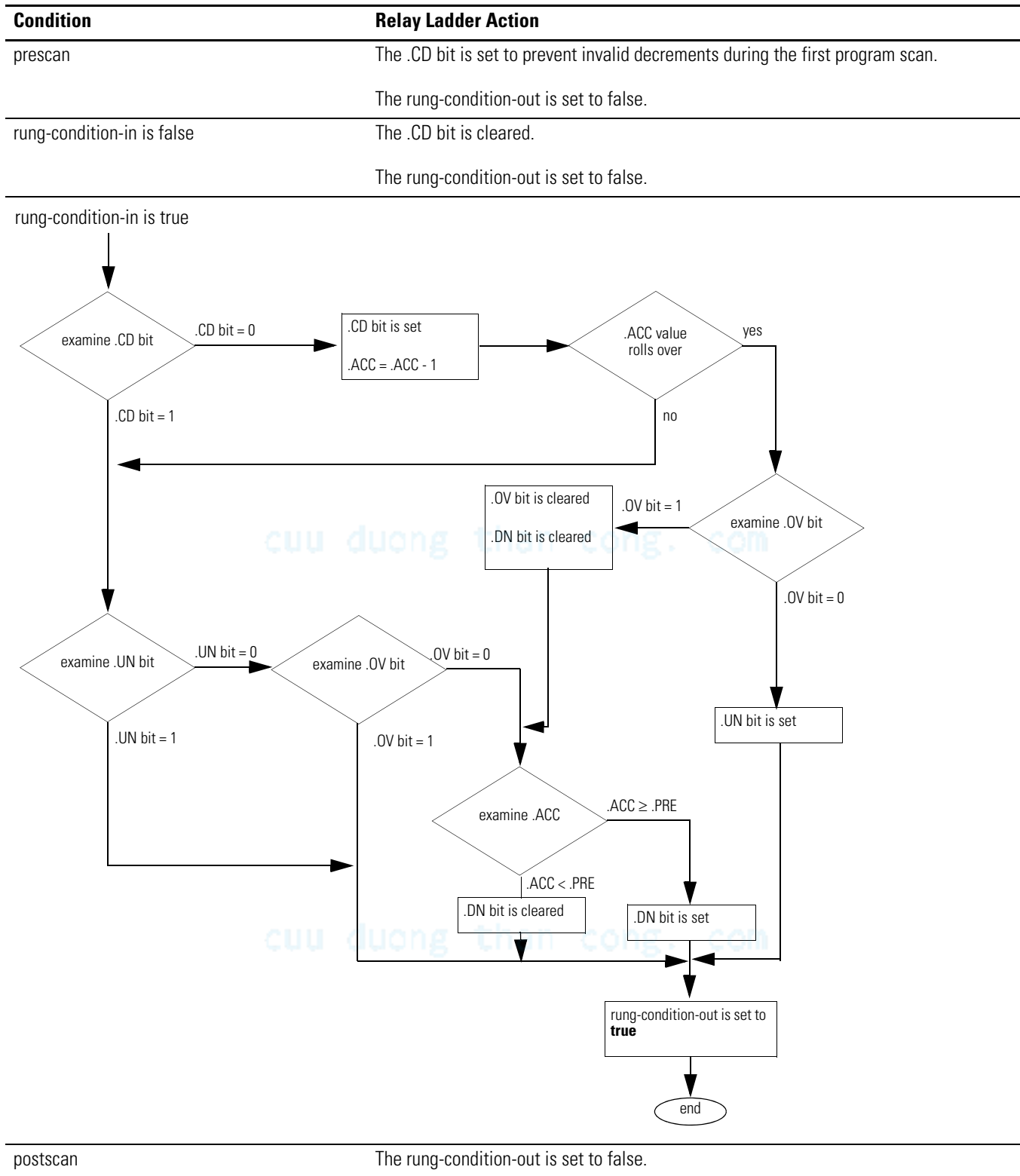
The accumulated value continues decrementing, even after the .DN bit is set. To clear the accumulated value, use a RES instruction that references the counter structure or write 0 to the accumulated value.

**Arithmetic Status Flags:** not affected

**Fault Conditions:** none



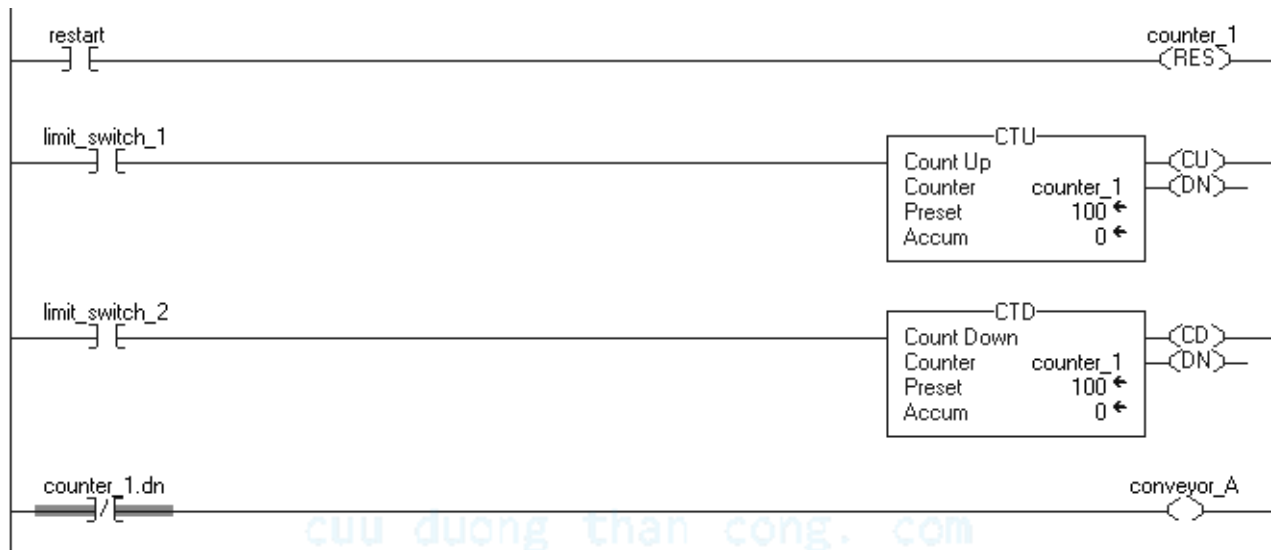
### Execution:



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 1. Each time a part leaves, *limit\_switch\_2* is enabled and *counter\_1* decrements by 1. If there are 100 parts in the buffer zone (*counter\_1.dn* is set), *conveyor\_a* turns on and stops the conveyor from bringing in any more parts until the buffer has room for more parts.



## Count Up/Down (CTUD)

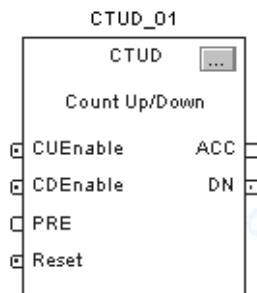
The CTUD instruction counts up by one when CUEnable transitions from clear to set. The instruction counts down by one when CDEnable transitions from clear to set.

This instruction is available in relay ladder as three separate instructions: CTU (see page 4-123), CTD (see page 4-127), and RES (see page 136).

### Operands:



CTUD(CTUD\_tag) ;



### Structured Text

Variable	Type	Format	Description
CTUD tag	FBD_COUNTER	structure	CTUD structure

### Function Block

Operand	Type	Format	Description
CTUD tag	FBD_COUNTER	structure	CTUD structure

### FBD\_COUNTER Structure

Input Parameter	Data Type	Description
EnableIn	BOOL	<b>Function Block:</b>  If cleared, the instruction does not execute and outputs are not updated.  If set, the instruction executes.  Default is set.  <b>Structured Text:</b>  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.

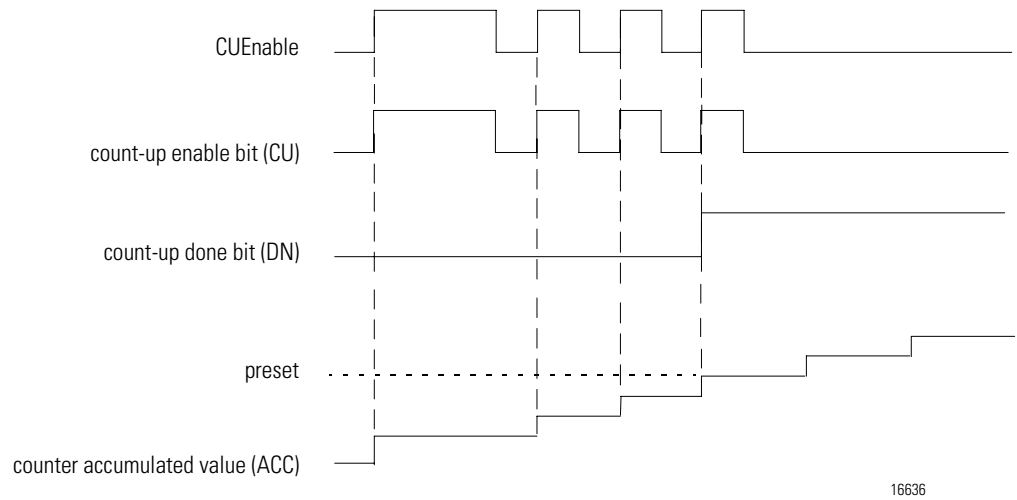
Input Parameter	Data Type	Description
CDEnable	BOOL	Enable down count. When input toggles from clear to set, accumulator counts down by one. Default is cleared.
PRE	DINT	Counter preset value. This is the value the accumulated value must reach before DN is set. Valid = any integer Default is 0.
Reset	BOOL	Request to reset the timer. When set, the counter resets. Default is cleared.

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

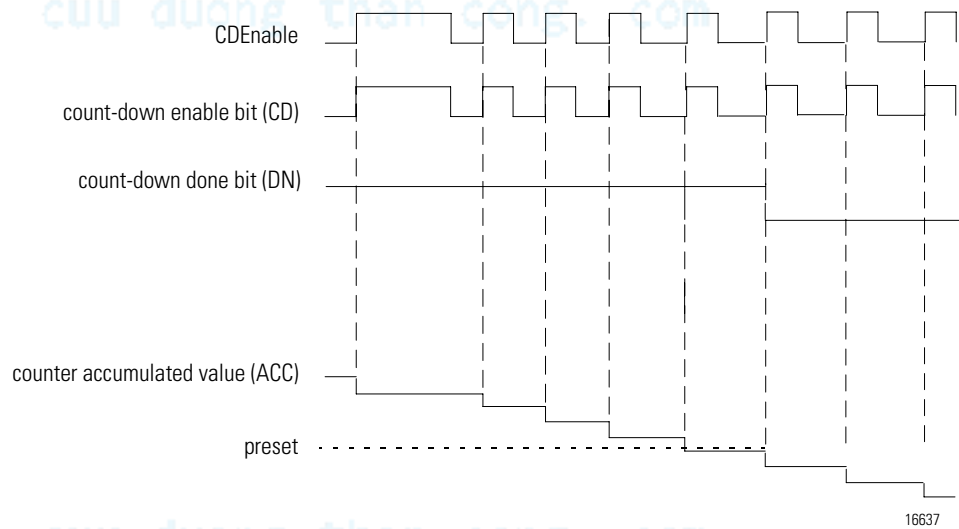
**Description** When enabled and CUEnable is set, the CTUD instructions increments the counter by one. When enabled and CDEnable is set, the CTUD instruction decrements the counter by one.

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

### Counting Up



### 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 <sub>n-1</sub> and CDEnable <sub>n-1</sub> are set.	CUEnable <sub>n-1</sub> and CDEnable <sub>n-1</sub> are set.
instruction first run	CUEnable <sub>n-1</sub> and CDEnable <sub>n-1</sub> are set.	CUEnable <sub>n-1</sub> and CDEnable <sub>n-1</sub> are set.
EnableIn is cleared	EnableOut is cleared, the instruction does nothing, and the outputs are not updated.	na
EnableIn is set	The instruction sets CUEnable <sub>n-1</sub> and CDEnable <sub>n-1</sub> .	The instruction sets CUEnable <sub>n-1</sub> and CDEnable <sub>n-1</sub> .
	On a cleared to set transition of EnableIn:	EnableIn is always set.
	<ul style="list-style-type: none"> <li>• The instruction executes.</li> <li>• EnableOut is set.</li> </ul>	The instruction executes.
reset	When set, the instruction clears CUEnable <sub>n-1</sub> , CDEnable <sub>n-1</sub> , CU, CD, DN, OV, and UN and sets ACC = zero.	When set, the instruction clears CUEnable <sub>n-1</sub> , CDEnable <sub>n-1</sub> , CU, CD, DN, OV, and UN and sets ACC = zero.
postscan	No action taken.	No action taken.

**Example:** When *limit\_switch1* goes from cleared to set, CUEnable is set for one scan and the CTUD instruction increments the ACC value by 1. When  $ACC \geq PRE$ , the DN parameter is set, which enables the function block instruction following the CTUD instruction.

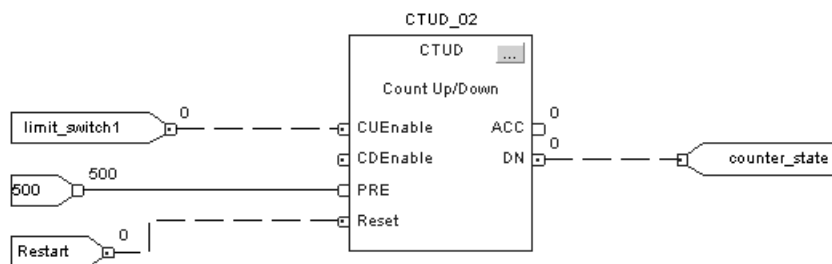
**Structured Text**

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

```
CTUD(CTUD_01);
```

```
counter_state := CTUD_01.DN;
```

**Function Block**



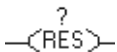
cuu duong than cong. com

cuu duong than cong. com

## Reset (RES)

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

### Operands:



### Relay Ladder

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

**Description:** When enabled the RES instruction clears these elements:

When Using a Res Instruction For a	The Instruction Clears
TIMER	.ACC value control status bits
COUNTER	.ACC value control status bits
CONTROL	.POS value control status bits

### ATTENTION



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

**Arithmetic Status Flags:** not affected

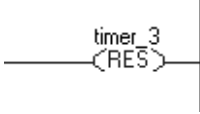

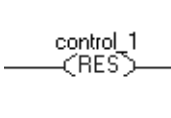
**Fault Conditions:** none



### Execution:

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

### Examples:

Example	Description
	When enabled, reset <i>timer_3</i> .
	When enabled, reset <i>counter_1</i> .
	When enabled, reset <i>control_1</i> .

## Notes:

cuu duong than cong. com

cuu duong than cong. com

## Input/Output Instructions

### (MSG, GSV, SSV, IOT)

### Introduction

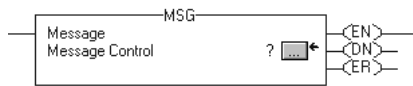
The input/output instructions read or write data to or from the controller or a block of data to or from another module on another network.

If You Want To	Use This Instruction	Available In These Languages	See Page
send data to or from another module	MSG	relay ladder	140
		structured text	
get controller status information	GSV	relay ladder	173
		structured text	
set controller status information	SSV	relay ladder	173
		structured text	
<ul style="list-style-type: none"> <li>send output values to an I/O module or consuming controller at a specific point in your logic</li> <li>trigger an event task in another controller</li> </ul>	IOT	relay ladder	200
		structured text	

# Message (MSG)

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

## Operands:



## Relay Ladder

Operand	Type	Format	Description
Message control	MESSAGE	tag	message structure



MSG(MessageControl) ;

## Structured Text

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

## MESSAGE Structure

### ATTENTION



### If you check the status bits more than once

The controller changes the DN, ER, EW, and ST bits asynchronous to the scan of your logic. Use a copy of the bits if you check them in more than one place in your logic. Otherwise, the bits may change during the scan and your logic won't work as you expect it.



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 the following status bits of a MSG instruction:

- DN
- EN
- ER
- EW
- ST

Do not change those bits either by themselves or as part of the FLAGS word. If you do, the controller may have a non-recoverable fault. The controller clears the project from its memory when it has a non-recoverable fault.

Mnemonic	Data Type	Description
.FLAGS	INT	The FLAGS member provides access to the status members (bits) in one 16-bit word.
		<b>This bit: Is this member:</b>
		2 .EW
		4 .ER
		5 .DN
		6 .ST
		7 .EN
		8 .TO
		9 .EN_CC
		<b>Important:</b> Do not change the 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.
.EXERR	INT	The extended error code word specifies additional error code information for some error codes.
.REQ_LEN	INT	The requested length specifies how many words the message instruction will attempt to transfer.
.DN_LEN	INT	The done length identifies how many words actually transferred.
.EW	BOOL	 The enable waiting bit is set when the controller detects that a message request has entered the queue. The controller resets the .EW bit when the .ST bit is set. <b>Important:</b> Do not change the EW bit. The controller ignores the change and uses the internally-stored value of the bit.
.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. <b>Important:</b> Do not change the ER bit.
.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. <b>Important:</b> Do not change the DN bit.
.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. <b>Important:</b> Do not change the ST bit. The controller ignores the change and uses the internally-stored value of the bit.
.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. <b>Important:</b> Do not change the EN bit.
.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 5-170 Connections for MSG instructions going out the serial port are not cached, even if the .EN_CC bit is set.
.ERR_SRC	SINT	Used by RSLogix 5000 software to show the error path on the Message Configuration dialog box
.DestinationLink	INT	To change the Destination Link of a DH+ or CIP with Source ID message, set this member to the required value.

Mnemonic	Data Type	Description
.DestinationNode	INT	To change the Destination Node of a DH+ or CIP with Source ID message, set this member to the required value.
.SourceLink	INT	To change the Source Link of a DH+ or CIP with Source ID message, set this member to the required value.
.Class	INT	To change the Class parameter of a CIP Generic message, set this member to the required value.
.Attribute	INT	To change the Attribute parameter of a CIP Generic message, set this member to the required value.
.Instance	DINT	To change the Instance parameter of a CIP Generic message, set this member to the required value.
.LocalIndex	DINT	If you use an asterisk [*] to designate the element number of the local array, the LocalIndex provides the element number. To change the element number, set this member to the required value.
		<b>If the message:</b> <b>Then the local array is the:</b>
		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.
		<b>If the message goes over this network:</b> <b>Then specify the slot number in:</b>
		universal remote I/O      octal
		ControlNet                      decimal (0-15)
.Path	STRING	To send the message to a different controller, set this member to the new path.
		<ul style="list-style-type: none"><li>• Enter the path as hexadecimal values.</li><li>• Omit commas [,]</li></ul>
		For example, for a path of 1, 0, 2, 42, 1, 3, enter \$01\$00\$02\$2A\$01\$03.
		To browse to a device and automatically create a portion or all of the new string, right-click a string tag and choose <i>Go to Message Path Editor</i> .
.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.
		<b>If the message:</b> <b>Then the remote array is the:</b>
		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.
		<b>If the message:</b> <b>Then the remote array is the:</b>
		reads data                      Source element
		writes data                      Destination element

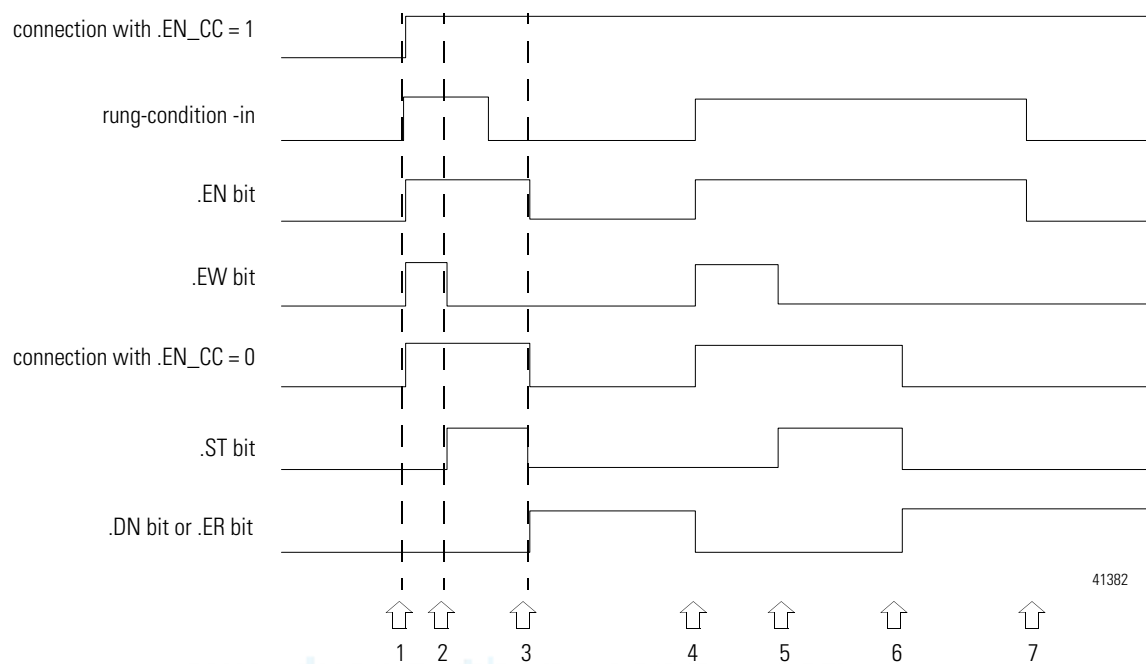
Mnemonic	Data Type	Description						
.UnconnectedTimeout	DINT	Time out for an unconnected message or for making a connection. The default value is 30 seconds.						
		<table><tr><th>If the message is</th><th>Then</th></tr><tr><td>unconnected</td><td>The ER bit turns on if the controller doesn't get a response within the UnconnectedTimeout time.</td></tr><tr><td>connected</td><td>The ER bit turns on if the controller doesn't get a response for making the connection within the UnconnectedTimeout time.</td></tr></table>	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.
		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.						
.TimeoutMultiplier	SINT	<ul style="list-style-type: none"><li>• This time out applies only after the connection is made.</li><li>• The time out = ConnectionRate x TimeoutMultiplier.</li><li>• The default ConnectionRate is 7.5 seconds.</li><li>• The default TimeoutMultiplier is 0 (which is a multiplication factor of 4).</li><li>• The default time out for connected messages is 30 seconds (7.5 seconds x 4 = 30 seconds).</li><li>• To change the time out, change the ConnectionRate and leave the TimeoutMultiplier at the default value.</li></ul>						

**Description** The MSG instruction transfers elements of data.

This is a transitional instruction:

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

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



41382



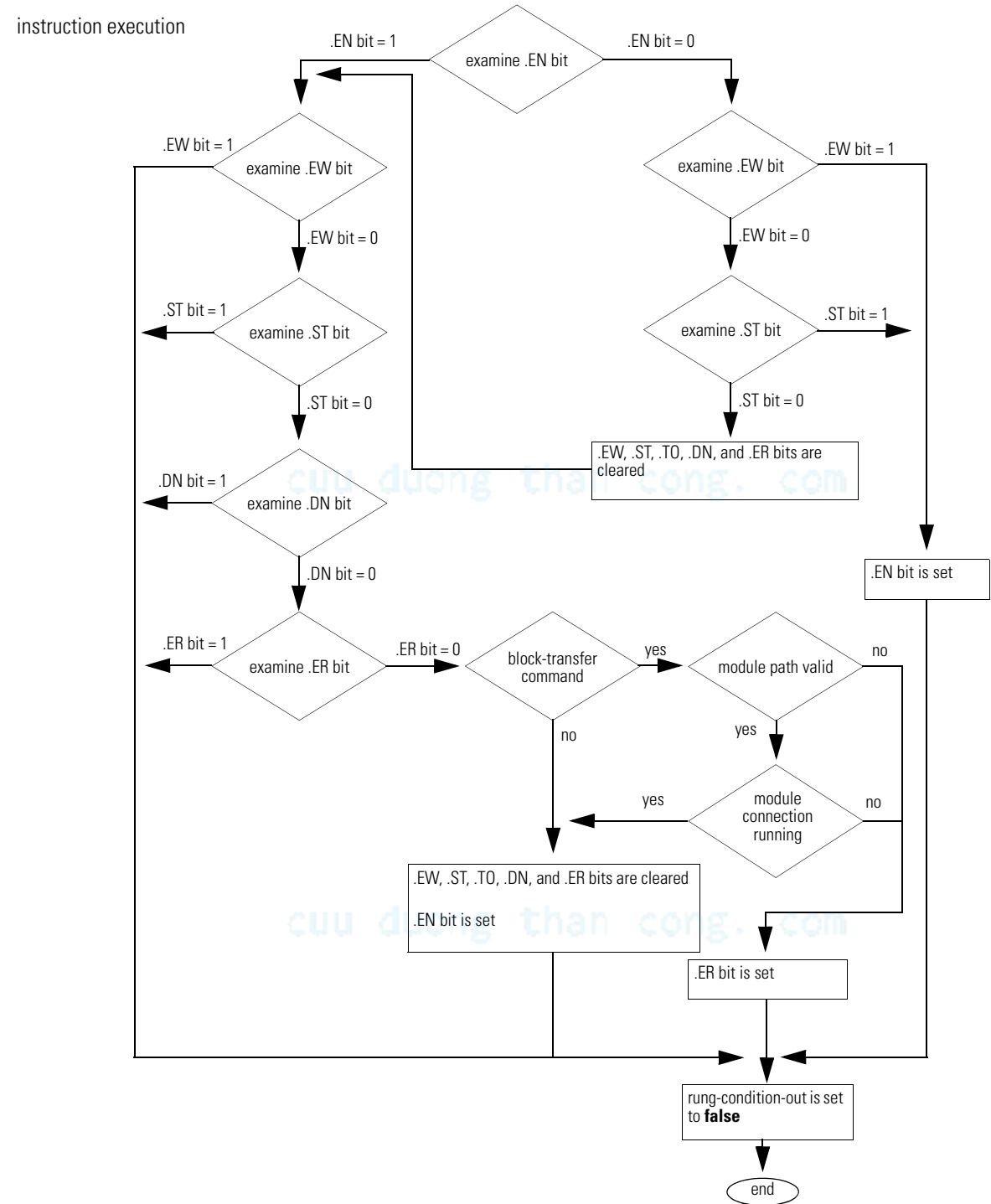
Where	Description	Where	Description
1	rung-condition-in is true .EN is set .EW is set connection is opened*	5	message is sent .ST is set .EW is cleared
2	message is sent .ST is set .EW is cleared	6	message is done or errored rung-condition-in is still true .DN or .ER is set .ST is cleared connection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

**Execution:**

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

Condition	Relay Ladder Action	Structured Text Action
rung-condition-in is false	<pre> graph TD     Start([Start]) --&gt; EN{examine .EN bit}     EN -- ".EN bit = 1" --&gt; EW{examine .EW bit}     EN -- ".EN bit = 0" --&gt; End([end])     EW -- ".EW bit = 1" --&gt; ST{examine .ST bit}     EW -- ".EW bit = 0" --&gt; DN1{examine .DN bit}     ST -- ".ST bit = 1" --&gt; DN1     ST -- ".ST bit = 0" --&gt; ER1{examine .ER bit}     DN1 -- ".DN bit = 1" --&gt; ER1     DN1 -- ".DN bit = 0" --&gt; DN2{examine .DN bit}     ER1 -- ".ER bit = 1" --&gt; BTC{block-transfer command}     ER1 -- ".ER bit = 0" --&gt; DN2     DN2 -- ".DN bit = 1" --&gt; ENCLR[.EN bit is cleared]     DN2 -- ".DN bit = 0" --&gt; ER2{examine .ER bit}     ENCLR --&gt; ER2     ER2 -- ".ER bit = 1" --&gt; BTC     ER2 -- ".ER bit = 0" --&gt; End     BTC -- yes --&gt; MPV{module path valid}     BTC -- no --&gt; EMR[execute message request]     MPV -- yes --&gt; MCR{module connection running}     MPV -- no --&gt; ERSET[.ER bit is set]     MCR -- yes --&gt; ERSET     MCR -- no --&gt; ERSET     EMR --&gt; EWSET[.EW bit is set]     ERSET --&gt; End     EWSET --&gt; End     </pre>	<p>na</p> <p>The rung-condition-out is set to true.</p>

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



postscan	The rung-condition-out is set to false.	No action taken.
----------	---	------------------

**Arithmetic Status Flags:** not affected

**Fault Conditions:** none

## MSG Error Codes

The error codes depend on the type of MSG instruction.

### Error Codes

RSLogix 5000 software does not always display the full description.

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

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

cuu duong than cong. com

cuu duong than cong. com

## Extended Error Codes

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

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

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

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

Extended Error Code (Hex)	Description
0203	Connection timeout

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

Extended Error Code (Hex)	Description
0000	extended status out of memory
0001	extended status out of instances

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

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

## PLC and SLC Error Codes (.ERR)

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

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

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

**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



## PLC and SLC Error Codes (hex) (Continued)

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

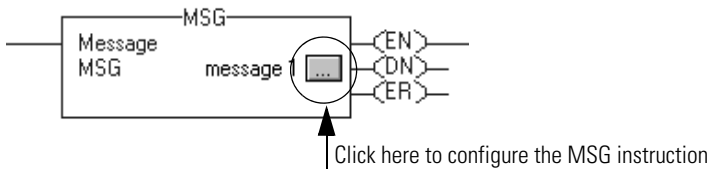
## Block-Transfer Error Codes

These are the Logix5000 block-transfer specific error codes.

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

## Specify the Configuration Details

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



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

42976

If The Target Device Is a	Select One Of These Message Types	See Page
Logix5000 controller	CIP Data Table Read	156
	CIP Data Table Write	
I/O module that you configure using RSLogix 5000 software	Module Reconfigure	157
	CIP Generic	158
PLC-5 controller	PLC5 Typed Read	159
	PLC5 Typed Write	
	PLC5 Word Range Read	
	PLC5 Word Range Write	
SLC controller	SLC Typed Read	161
MicroLogix controller	SLC Typed Write	
Block-transfer module	Block-Transfer Read	161
	Block-Transfer Write	
PLC-3 processor	PLC3 typed read	162
	PLC3 typed write	
	PLC3 word range read	
	PLC3 word range write	
PLC-2 processor	PLC2 unprotected read	163
	PLC2 unprotected write	

You must specify this configuration information.

For This Property	Specify
Source Element	<ul style="list-style-type: none"> <li>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.</li> <li>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.</li> </ul>
Number of Elements	The number of elements you read/write depends on the type of data you are using. An element refers to one "chunk" of related data. For example, tag <i>timer1</i> is one element that consists of one timer control structure.
Destination Element	<ul style="list-style-type: none"> <li>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.</li> <li>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.</li> </ul>

### 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:

- 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.

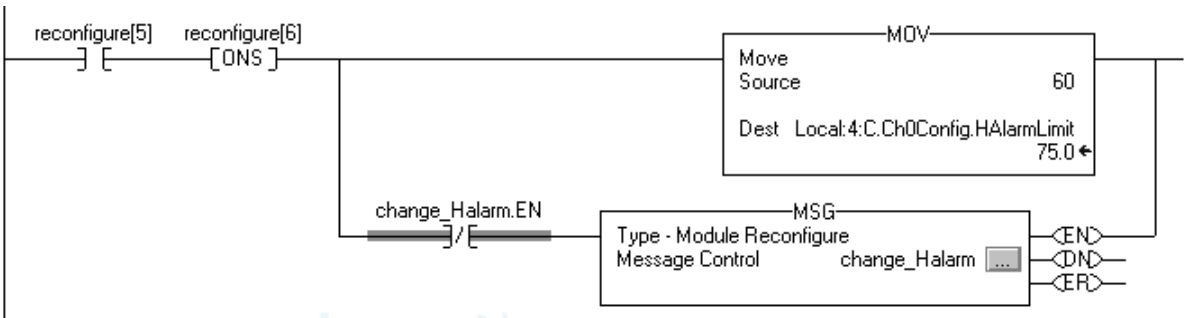
In This Property	Select
Message Type	Module Reconfigure

**Example:** To reconfigure an I/O module:

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

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

### Relay Ladder



### Structured Text

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

## Specify CIP Generic messages

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

If You Want To	In This Property	Type Or Select
Perform a pulse test on a digital output module	Message Type	CIP Generic
	Service Type	Pulse Test
	Source	<i>tag_name</i> of type INT [5]
		This array contains:
	<i>tag_name</i> [0]	bit mask of points to test (test only one point at a time)
	<i>tag_name</i> [1]	reserved, leave 0
	<i>tag_name</i> [2]	pulse width (hundreds of $\mu$ secs, usually 20)
	<i>tag_name</i> [3]	zero cross delay for ControlLogix I/O (hundreds of $\mu$ secs, usually 40)
	<i>tag_name</i> [4]	verify delay
	Destination	leave blank
Reset electronic fuses on a digital output module	Message Type	CIP Generic
	Service Type	Reset Electronic Fuse
	Source	<i>tag_name</i> of type DINT
		This tag represents a bit mask of the points to reset fuses on.
	Destination	leave blank
Reset latched diagnostics on a digital input module	Message Type	CIP Generic
	Service Type	Reset Latched Diagnostics (I)
	Source	<i>tag_name</i> of type DINT
		This tag represents a bit mask of the points to reset diagnostics on.
Reset latched diagnostics on a digital output module	Message Type	CIP Generic
	Service Type	Reset Latched Diagnostics (O)
	Source	<i>tag_name</i> of type DINT
		This tag represents a bit mask of the points to reset diagnostics on.

If You Want To	In This Property	Type Or Select
Unlatch the alarm of an analog input module	Message Type	CIP Generic
	Service Type	Select which alarm that you want to unlatch: <ul style="list-style-type: none"> <li>• Unlatch All Alarms (I)</li> <li>• Unlatch Analog High Alarm (I)</li> <li>• Unlatch Analog High High Alarm (I)</li> <li>• Unlatch Analog Low Alarm (I)</li> <li>• Unlatch Analog Low Low Alarm (I)</li> <li>• Unlatch Rate Alarm (I)</li> </ul>
	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: <ul style="list-style-type: none"> <li>• Unlatch All Alarms (O)</li> <li>• Unlatch High Alarm (O)</li> <li>• Unlatch Low Alarm (O)</li> <li>• Unlatch Ramp Alarm (O)</li> </ul>
	Instance	Channel of the alarm that you want to unlatch

## 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 160.
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 160
PLC5 Word Range Read	Read a contiguous range of 16-bit words in PLC-5 memory regardless of data type.
	<p>This command starts at the address specified as the Source Element and reads sequentially the number of 16-bit words requested.</p> <p>The data from the Source Element is stored, starting at the address specified as the Destination Tag.</p>
PLC5 Word Range Write	Write a contiguous range of 16-bit words from Logix5000 memory regardless of data type to PLC-5 memory.
	<p>This command starts at the address specified as the Source Tag and reads sequentially the number of 16-bit words requested.</p> <p>The data from the Source Tag is stored, starting at the address specified as the Destination Element in the PLC-5 processor.</p>

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

**Data types for PLC5 Typed Read and Typed Write messages**

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

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

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

**Typed read command**

16-bit words in  
PLC-5 processor

1
2
3
4



32-bit words in  
Logix5000 controller

1
2
3
4

The typed commands maintain data structure and value.

**Word-range read command**

16-bit words in  
PLC-5 processor

1
2
3
4



32-bit words in  
Logix5000 controller

2	1
4	3

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 to 64 integers.

If You Want The	Then Specify
Block-transfer module to determine how many 16-bit integers to send (BTR).	0 for the number of elements
Controller to send 64 integers (BTW).	

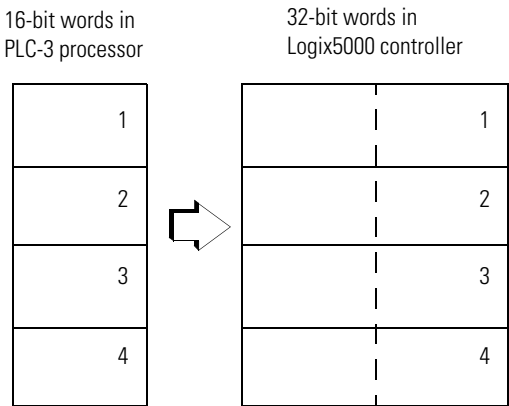
## Specify PLC-3 messages

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

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

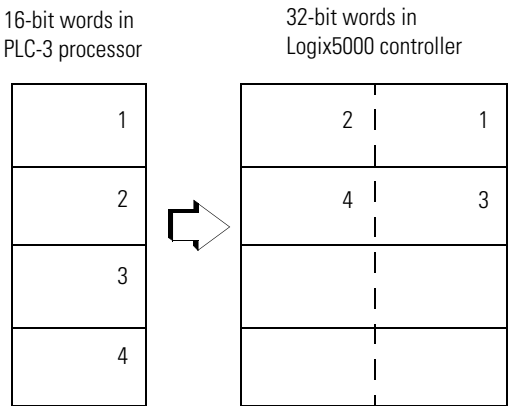
The following diagrams show how the typed and word-range commands differ. The example uses read commands from a PLC-3 processor to a Logix5000 controller.

**Typed read command**



The typed commands maintain data structure and value.

**Word-range read command**



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

cuu duong than cong. com

**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).

cuu duong than cong. com

## MSG Configuration Examples

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

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

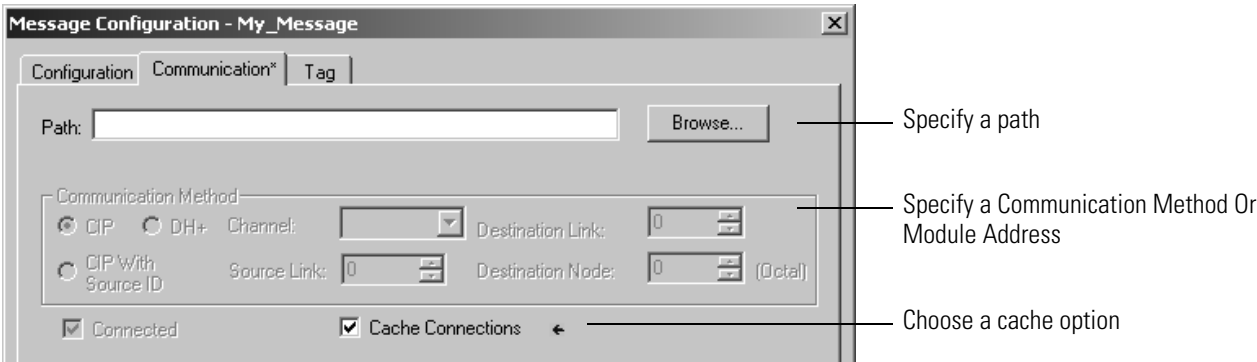
Message Path	Example Source and Destination	
Logix5000 → Logix5000	source tag	<i>array_1[0]</i>
	destination tag	<i>array_2[0]</i>
	You can use an alias tag for the source tag (in originating Logix5000 controller). You <b>cannot</b> use an alias for the destination tag. The destination must be a base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

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

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

# Specify the Communication Details

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.

If	Then
The I/O configuration of the controller has the module that gets the message.	Use the <i>Browse</i> button to select the module.
The I/O configuration of the controller has only the local communication module.	1. Use the <i>Browse</i> button 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.

Click the Browse button and select the module.

Path: Peer\_Controller

Peer\_Controller

Browse...

The I/O configuration of the controller has only the local communication module.

Go to the local communication module.

Go out the EtherNet/IP port...

to the address of 10.10.10.10.

Go across the backplane...

to the module in slot 0.

Path: LocalENB, 2, 10.10.10.10, 1, 0

LocalENB, 2, 10.10.10.10, 1, 0

Browse...

The I/O configuration of the controller doesn't have any of the modules that you need for the message.

Go across the backplane...

to the local communication module on slot 1

Go out the ControlNet port....

to node 4

Go across the backplane...

to the module in slot 0.

Path: 1, 1, 2, 4, 1, 0

1, 1, 2, 4, 1, 0

Browse...

To type a path, use this format:

*port, next\_address, port, next\_address, ...*

Where	Is	
	For this network	Type
<i>port</i>	backplane	1
	DF1 (serial, serial channel 0)	2
	ControlNet	
	EtherNet/IP	
	DH+ channel A	
	DH+ channel B	3
	DF1 channel 1 (serial channel 1)	
<i>next_address</i>	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 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)

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

cuu duong than cong. com

cuu duong than cong. com



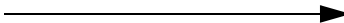
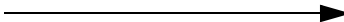
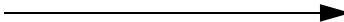
## Specify a Communication Method Or Module Address

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

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

## Choose a cache option

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

This Type Of Message	And This Communication Method	Uses A Connection
CIP data table read or write		3
PLC2, PLC3, PLC5, or SLC (all types)	CIP	3
	CIP with Source ID	
	DH+	
CIP generic		your option <sup>(1)</sup>
block-transfer read or write		3

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

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

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

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

If You Have This Software And Firmware Revision	Then You Can Cache
11.x or earlier	<ul style="list-style-type: none"> <li>• block transfer messages for up to 16 connections</li> <li>• other types of messages for up to 16 connections</li> </ul>
12.x or later	up to 32 connections

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

IF THE MSG Instructions Are To	And They Are	Then
different devices	→	Each MSG instruction uses 1 connection.
same device	enabled at the same time	Each MSG instruction uses 1 connection.
	NOT enabled at the same time	The MSG instructions share the connection. (that is, Together they count as 1 connection.)

**EXAMPLE****Share a Connection**

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

cuu duong than cong. com

cuu duong than cong. com

## Guidelines

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

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

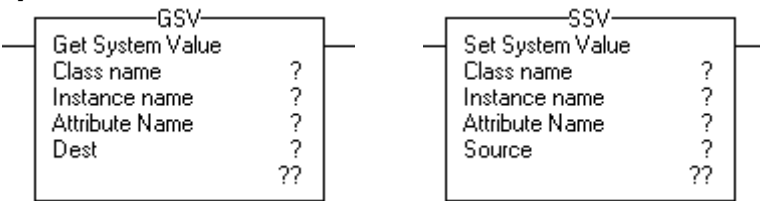
# Get System Value (GSV) and Set System Value (SSV)

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	tag	destination for attribute data
	INT		
	DINT		
	REAL		
	structure		
Source (SSV)	SINT	tag	tag that contains data you want to copy to the attribute
	INT		
	DINT		
	REAL		
	structure		



### 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**

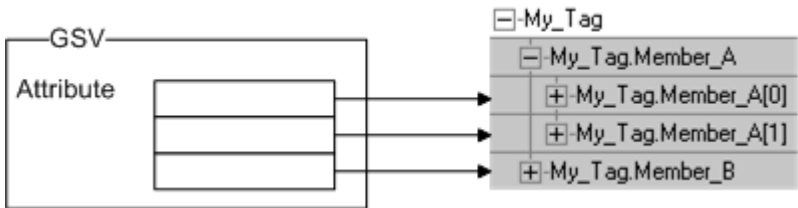


Use the GSV and SSV instructions carefully. Making changes to objects may cause unexpected controller operation or injury to personnel.

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

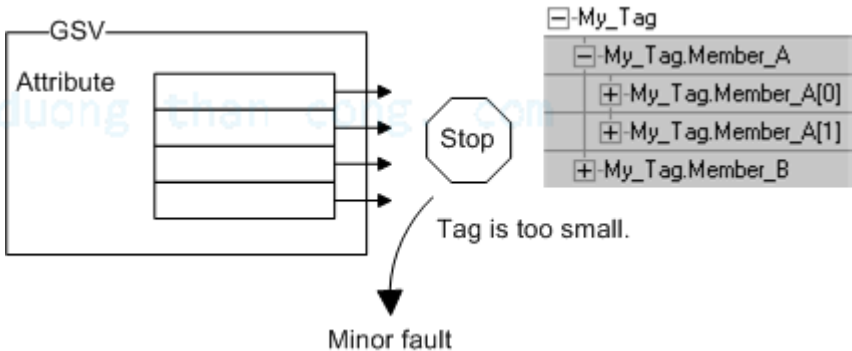
The GSV and SSV instructions write or read past a member into other members of a tag. If the tag is too small, the instructions don't write or read the data. They log a minor fault instead.

**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.	na
rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	na
EnableIn is set	na	EnableIn is always set. The instruction executes.
instruction executes	Get or set the specified value.	Get or set the specified value.
postscan	The rung-condition-out is set to false.	No action taken.

## GSV/SSV Objects

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

### ATTENTION



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

You can access these objects:

For Information About This Object	See This Page Or Publication
AXIS	<i>ControlLogix Motion Module Setup and Configuration Manual</i> , publication 1756-UM006
CONTROLLER	5-177
CONTROLLERDEVICE	5-177
CST	5-181
DF1	5-182
FAULTLOG	5-185
MESSAGE	5-186
MODULE	5-188
MOTIONGROUP	5-189
PROGRAM	5-190
ROUTINE	5-192
SERIALPORT	5-192
TASK	5-194
WALLCLOCKTIME	5-196



## Access the CONTROLLER object

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

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

## Access the CONTROLLERDEVICE object

The CONTROLLERDEVICE object identifies the physical hardware of the controller.

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

Attribute	Data Type	Instruction	Description
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.

cuu duong than cong. com

cuu duong than cong. com

Attribute	Data Type	Instruction	Description
Status	INT	GSV	<p>Bits identify status:</p> <p>Bits 3-0 are reserved</p> <p><b>Device Status Bits</b></p> <p><b>Bits 7-4:</b>      <b>Meaning:</b></p> <p>0000              reserved</p> <p>0001              flash update in progress</p> <p>0010              reserved</p> <p>0011              reserved</p> <p>0100              flash is bad</p> <p>0101              faulted</p> <p>0110              run</p> <p>0111              program</p> <p><b>Fault Status Bits</b></p> <p><b>Bits 11-8:</b>      <b>Meaning:</b></p> <p>0001              recoverable minor fault</p> <p>0010              unrecoverable minor fault</p> <p>0100              recoverable major fault</p> <p>1000              unrecoverable major fault</p> <p><b>Logix5000 Specific Status Bits</b></p> <p><b>Bits 13-12:</b>      <b>Meaning:</b></p> <p>01                  keyswitch in run</p> <p>10                  keyswitch in program</p> <p>11                  keyswitch in remote</p> <p><b>Bits 15-14</b>      <b>Meaning</b></p> <p>01                  controller is changing modes</p> <p>10                  debug mode if controller is in run mode</p>

Attribute	Data Type	Instruction	Description
Type	INT	GSV	Identifies the device as a controller. Controller = 14
Vendor	INT	GSV	Identifies the vendor of the device. Allen-Bradley = 0001

cuu duong than cong. com

cuu duong than cong. com

## Access the CST object

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

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

## Access the DF1 object

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

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

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

Attribute	Data Type	Instruction	Description
PendingDuplicateDetection	SINT	SSV	Pending value for the DuplicateDetection attribute.
PendingEmbeddedResponse Enable	SINT	SSV	Pending value for the EmbeddedResponse attribute.
PendingENQTransmitLimit	SINT	SSV	Pending value for the ENQTransmitLimit attribute.
PendingEOTSuppression	SINT	SSV	Pending value for the EOTSuppression attribute.
PendingErrorDetection	SINT	SSV	Pending value for the ErrorDetection attribute.
PendingNormalPollGroupSize	INT	SSV	Pending value for the NormalPollGroupSize attribute.
PendingMasterMessage Transmit	SINT	SSV	Pending value for the MasterMessageTransmit attribute.
PendingNAKReceiveLimit	SINT	SSV	Pending value for the NAKReceiveLimit attribute.
PendingPollingMode	SINT	SSV	Pending value for the PollingMode attribute.
PendingReplyMessageWait	DINT	SSV	Pending value for the ReplyMessageWait attribute.
PendingStationAddress	INT	SSV	Pending value for the StationAddress attribute.
PendingSlavePollTimeout	DINT	SSV	Pending value for the SlavePollTimeout attribute.
PendingTransmitRetries	SINT	SSV	Pending value for the TransmitRetries attribute.

To apply values for any of the DF1 pending attributes:

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

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

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

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



## Access the FAULTLOG object

The FAULTLOG object provides fault information about the controller.

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

## Access The MESSAGE Object

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

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

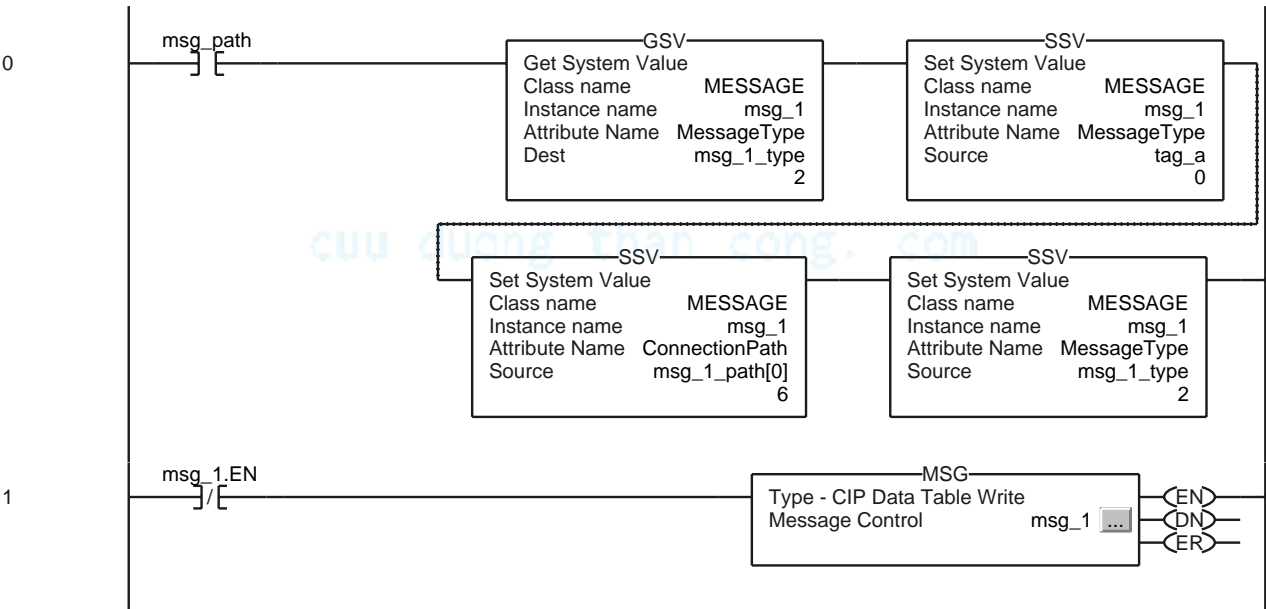
To change a MESSAGE attribute, follow these steps:

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

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

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

Relay Ladder



Structured Text

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

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

## Access The MODULE Object

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

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

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

## Access The MOTIONGROUP Object

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

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

## Access The PROGRAM Object

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

Attribute	Data Type	Instruction	Description
DisableFlag	SINT	GSV	Controls this program's execution.
		SSV	<b>Value:</b> 0 1 <b>Meaning:</b> execution enabled execution disabled
Instance	DINT	GSV	Provides the instance number of this PROGRAM object.
LastScanTime	DINT	GSV	Time it took to execute this program the last time it was executed.
		SSV	Time is in microseconds.
MajorFaultRecord	DINT[11]	GSV	Records major faults for this program
		SSV	We recommend that you create a user-defined structure to simplify access to the MajorFaultRecord attribute:
<b>Name:</b>	<b>Data Type:</b>	<b>Style:</b>	<b>Description:</b>
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, etc.)
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)
MaxScanTime	DINT	GSV	Maximum recorded execution time for this program. Time is in microseconds.
		SSV	

Attribute	Data Type	Instruction	Description
MinorFaultRecord	DINT[11]	GSV	Records minor faults for this program
		SSV	We recommend that you create a user-defined structure to simplify access to the MinorFaultRecord attribute:
<b>Name:</b>	<b>Data Type:</b>	<b>Style:</b>	<b>Description:</b>
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, etc.)
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)
SFCRestart	INT	GSV	unused - reserved for future use
		SSV	

cuu duong than cong. com

cuu duong than cong. com

## Access The Routine object

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

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

## Access The SERIALPORT Object

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

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



Attribute	Data Type	Instruction	Description
PendingBaudRate	DINT	SSV	Pending value for the BaudRate attribute.
PendingDataBits	SINT	SSV	Pending value for the DataBits attribute.
PendingParity	SINT	SSV	Pending value for the Parity attribute.
PendingRTSOffDelay	INT	SSV	Pending value for the RTSOffDelay attribute.
PendingRTSSendDelay	INT	SSV	Pending value for the RTSSendDelay attribute.
PendingStopBits	SINT	SSV	Pending value for the StopBits attribute.

To apply values for any of the SERIALPORT pending attributes:

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

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

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

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

## Access The TASK Object

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

Attribute	Data Type	Instruction	Description
DisableUpdateOutputs	DINT	GSV	Enables or disables the processing of outputs at the end of a task
		SSV	<b>To:</b> enable the processing of outputs at the end of the task <b>Set the attribute to:</b> 0
			<b>To:</b> disable the processing of outputs at the end of the task <b>Set the attribute to:</b> 1 (or any non-zero value)
EnableTimeOut	DINT	GSV	Enables or disables the timeout function of an event task.
		SSV	<b>To:</b> disable the timeout function <b>Set the attribute to:</b> 0
			<b>To:</b> enable the timeout function <b>Set the attribute to:</b> 1 (or any non-zero value)
InhibitTask	DINT	GSV	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.
		SSV	<b>To:</b> enable the task <b>Set the attribute to:</b> 0 (default)
			<b>To:</b> inhibit (disable) the task <b>Set the attribute to:</b> 1 (or any non-zero value)
Instance	DINT	GSV	Provides the instance number of this TASK object.
			Valid values are 0-31.
LastScanTime	DINT	GSV	Time it took to execute this task the last time it was executed. Time is in microseconds.
		SSV	
MaxInterval	DINT[2]	GSV	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.
		SSV	A value of 0 indicates 1 or less executions of the task.
MaxScanTime	DINT	GSV	Maximum recorded execution time for this program. Time is in microseconds.
		SSV	
MinInterval	DINT[2]	GSV	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.
		SSV	A value of 0 indicates 1 or less executions of the task.
OverlapCount	DINT	GSV	Number of times that the task was triggered while it was still executing. Valid for an event or a periodic task.
		SSV	To clear the count, set the attribute to 0.

Attribute	Data Type	Instruction	Description
Priority	INT	GSV	Relative priority of this task as compared to the other tasks.
		SSV	Valid values 0-15.
Rate	DINT	GSV	<b>If the task type is:</b>
		SSV	<b>Then the Rate attribute specifies the:</b>
			periodic
			event
StartTime	DINT[2]	GSV	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.
		SSV	
Status	DINT	GSV	Provides status information about the task. Once the controller sets one of these bits, you must manually clear the bit.
		SSV	<b>To determine if:</b>
			<b>Examine this bit:</b>
			An EVNT instruction triggered the task (event task only).
			A timeout triggered the task (event task only).
Watchdog	DINT	GSV	Time limit for execution of all programs associated with this task. Time is in microseconds.
		SSV	If you enter 0, these values are assigned:
			<b>Time:</b>
			<b>Task Type:</b>
			0.5 sec
			5.0 sec
			periodic or event
			continuous

## Access The WALLCLOCKTIME Object

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

Attribute	Data Type	Instruction	Description
CSTOffset	DINT[2]	GSV	Positive offset from the CurrentValue of the CST object (coordinated system time, see page 5-181). DINT[0] contains the lower 32 bits of the value; DINT[1] contains the upper 32 bits of the value.
		SSV	
			Value in $\mu$ secs. The default is 0.
CurrentValue	DINT[2]	GSV	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.
		SSV	
			The value is the number of microseconds that have elapsed since 0000 hrs 1 January 1972.
			The CST and WALLCLOCKTIME objects are mathematically related in the controller. For example, if you add the CST CurrentValue and the WALLCLOCKTIME CSTOffset, the result is the WALLCLOCKTIME CurrentValue.
DateTime	DINT[7]	GSV	The date and time in a readable format.
		SSV	DINT[0]    year
			DINT[1]    integer representation of month (1-12)
			DINT[2]    integer representation of day (1-31)
			DINT[3]    hour (0-23)
			DINT[4]    minute (0-59)
			DINT[5]    seconds (0-59)
			DINT[6]    microseconds (0-999,999)

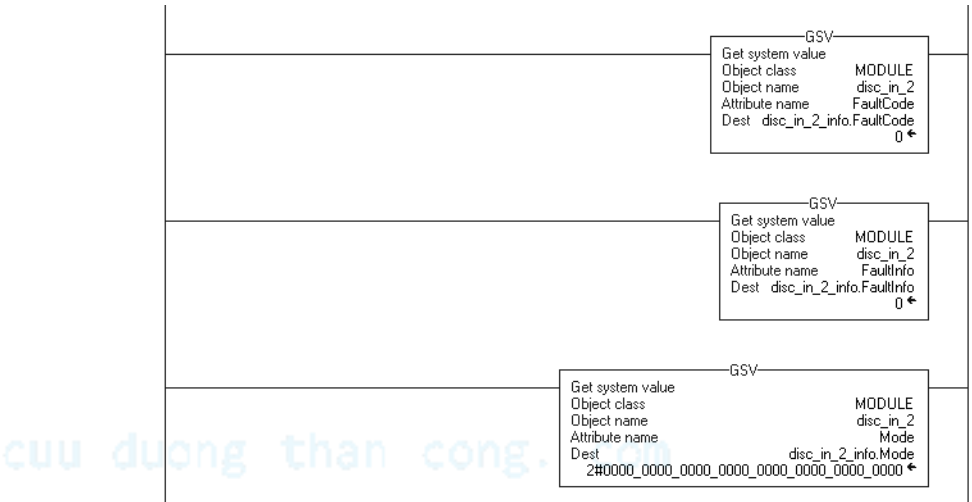
# GSV/SSV Programming Example

## 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

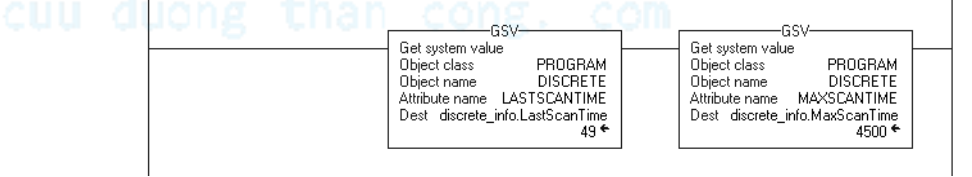


### Structured Text

```
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

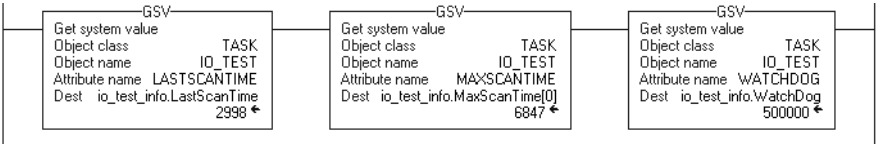


### Structured Text

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

**Example 3:** This example gets status information about task *IO\_test* and places the data in a user-defined structure *io\_test\_info*.

### Relay Ladder



### Structured Text

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

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

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

cuu duong than cong. com

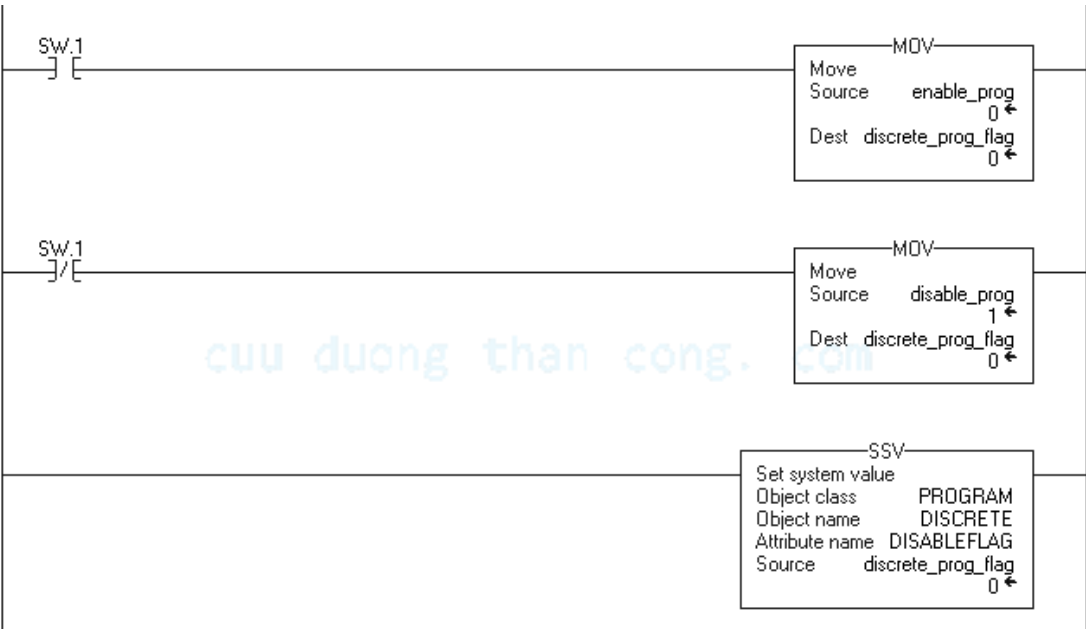
cuu duong than cong. com

### 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 a similar to using inhibit bits with a PLC-5 processor.

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

#### Relay Ladder



#### Structured Text

```

IF SW.1 THEN
    discrete_prog_flag := enable_prog;
ELSE
    discrete_prog_flag := disable_prog;
END_IF;

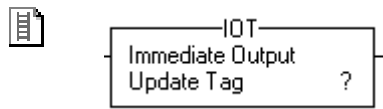
SSV(PROGRAM,DISCRETE,DISABLEFLAG,discrete_prog_flag);

```

# Immediate Output (IOT)

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

## Operands:



## Relay Ladder

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

```
IOT(output_tag);
```

## Structured Text

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

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

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

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

Check this box. →

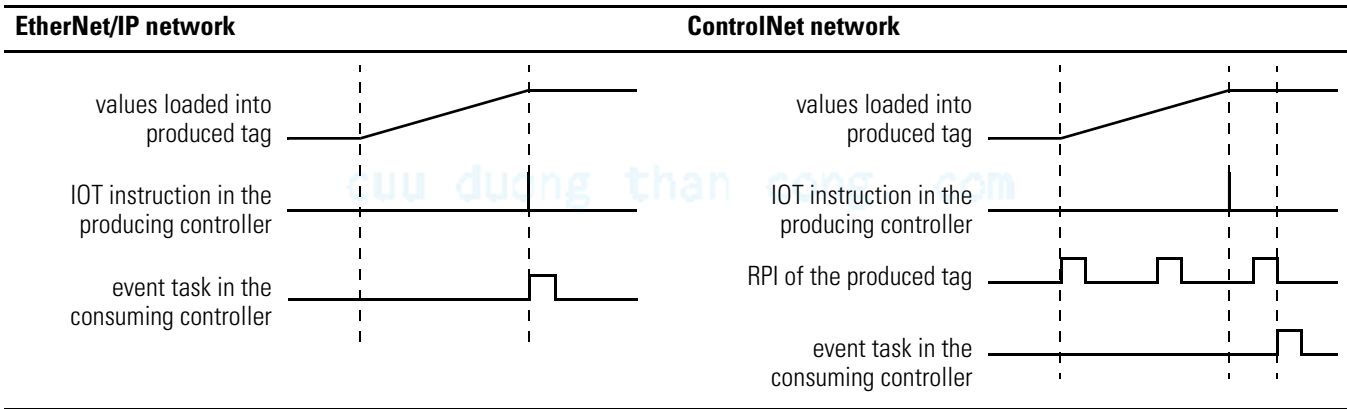
This configures the tag to update its event trigger only via an IOT instruction.



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.



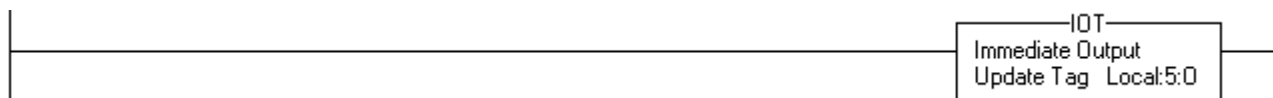
**Arithmetic Status Flags:** not affected

**Fault Conditions:** none

**Execution:**

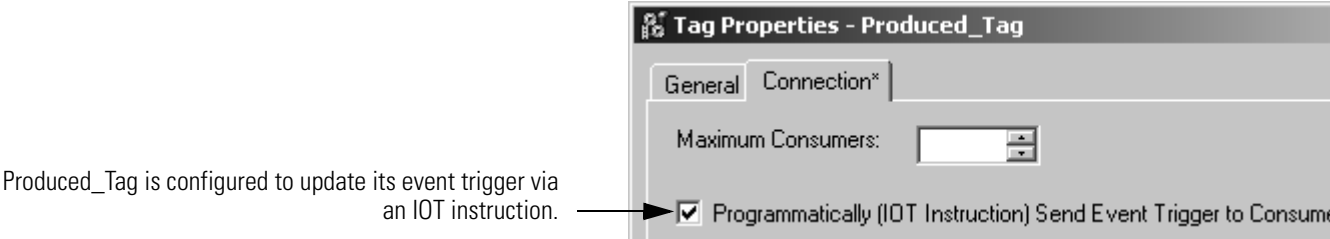
Condition	Relay Ladder Action	Structured Text Action
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	na
EnableIn is set	na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

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

**Relay Ladder****Structured Text**

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

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

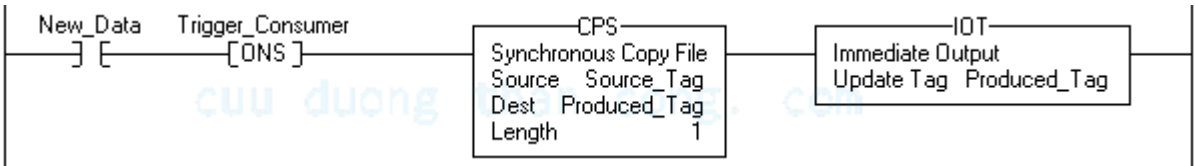


**Relay Ladder**

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:

cuu duong than cong. com

cuu duong than cong. com

## Compare Instructions

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

### Introduction

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

If You Want To	Use This Instruction	Available In These Languages	See Page
compare values based on an expression	CMP	relay ladder	207
		structured text <sup>(1)</sup>	
test whether two values are equal	EQU	relay ladder	212
		structured text <sup>(2)</sup>	
		function block	
test whether one value is greater than or equal to a second value	GEQ	relay ladder	216
		structured text <sup>(1)</sup>	
		function block	
test whether one value is greater than a second value	GRT	relay ladder	220
		structured text <sup>(1)</sup>	
		function block	
test whether one value is less than or equal to a second value	LEQ	relay ladder	224
		structured text <sup>(1)</sup>	
		function block	
test whether one value is less than a second value	LES	relay ladder	228
		structured text <sup>(1)</sup>	
		function block	

If You Want To	Use This Instruction	Available In These Languages	See Page
test whether one value is between two other values	LIM	relay ladder	232
		structured text <sup>(1)</sup>	
		function block	
pass two values through a mask and test whether they are equal	MEQ	relay ladder	238
		structured text <sup>(1)</sup>	
		function block	
test whether one value is not equal to a second value	NEQ	relay ladder	243
		structured text <sup>(1)</sup>	
		function block	

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

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

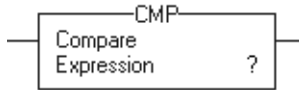
You can compare values of different data types, such as floating point and integer.

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

## Compare (CMP)

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

### Operands:



### Relay Ladder

Operand	Type	Format	Description
Expression	SINT	immediate	an expression consisting of tags and/or immediate values separated by operators
	INT	tag	
	DINT		
	REAL		
	string		
	A SINT or INT tag converts to a DINT value by sign-extension.		



### Structured Text

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

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

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

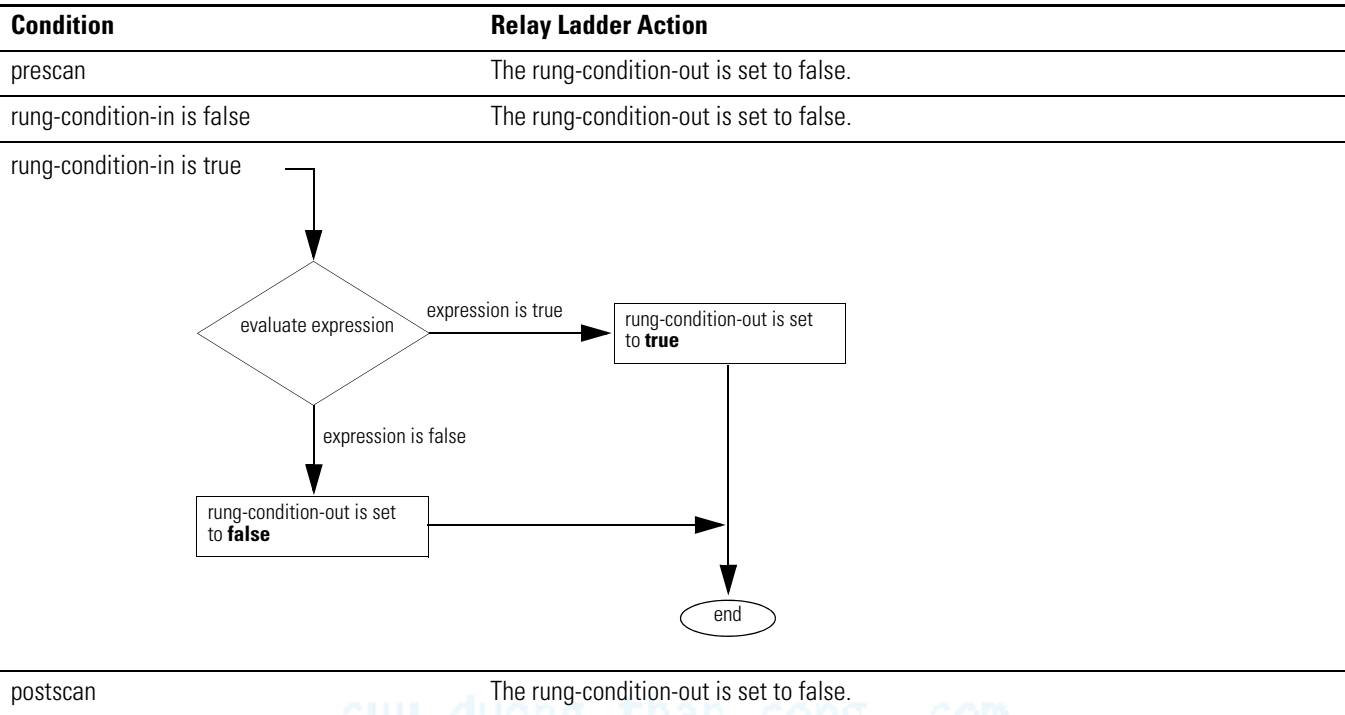
**Description:** Define the CMP expression using operators, tags, and immediate values. Use parentheses ( ) to define sections of more complex expressions.

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

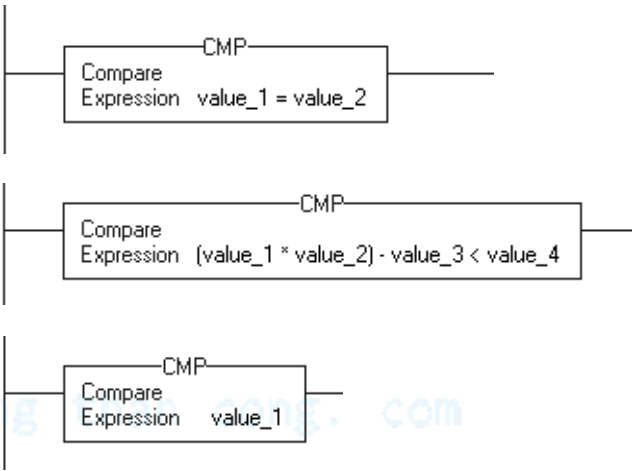
**Arithmetic Status Flags:** The CMP instruction only affects the arithmetic status flags if the expression contains an operator (for example, +, −, \*, /) that affects the arithmetic status flags.

**Fault Conditions:** none

**Execution:**



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



If you enter an expression without a comparison operator, such as *value\_1 + value\_2*, or *value\_1*, the instruction evaluates the expression as:

If The Expression	The Rung-condition-out Is Set To
non zero	true
zero	false



## CMP expressions

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

## Valid operators

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

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

## 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	<ul style="list-style-type: none"> <li><math>tag\_b + 5</math></li> <li><math>tag\_c \text{ AND } tag\_d</math></li> <li><math>(tag\_e ** 2) \text{ MOD } (tag\_f / tag\_g)</math></li> </ul>

## 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

less

greater

↑

↓

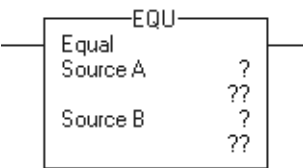
AB < B

a > B

# Equal to (EQU)

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

## Operands:



## Relay Ladder

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

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

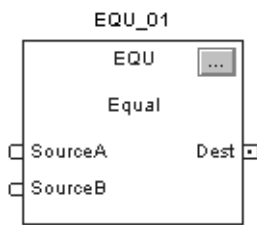


```
IF sourceA = sourceB THEN
    <statements>;
```

## Structured Text

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

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



### Function Block

Operand	Type	Format	Description
EQU tag	FBD_COMPARE	structure	EQU structure

### FBD\_COMPARE Structure

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

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

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

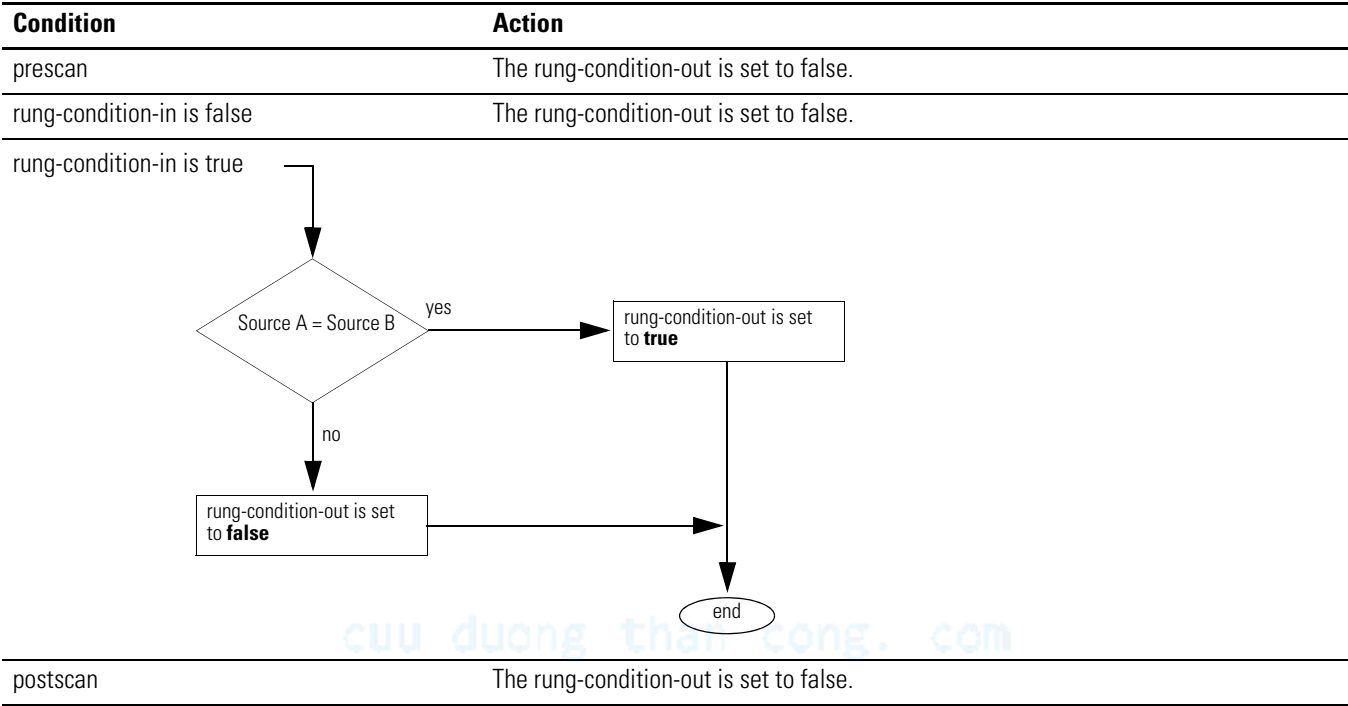
**Arithmetic Status Flags:** not affected

**Fault Conditions:** none

**Execution:**



**Relay Ladder**

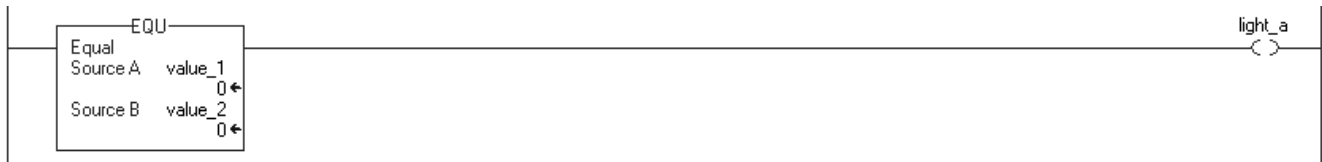


**Function Block**

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

**Example:** If *value\_1* is equal to *value\_2*, set *light\_a*. If *value\_1* is not equal to *value\_2*, clear *light\_a*.

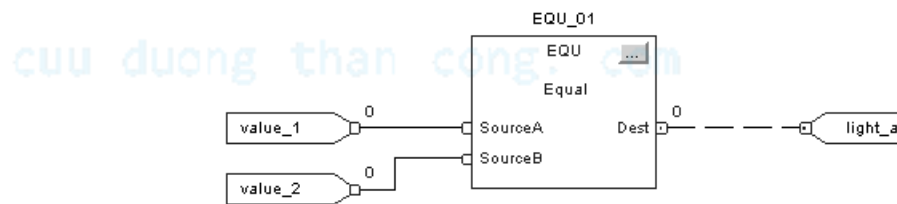
### Relay Ladder



### Structured Text

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

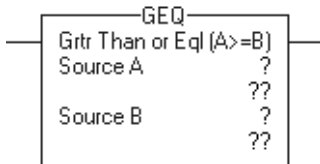
### Function Block



## Greater than or Equal to (GEQ)

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

### Operands:



### Relay Ladder

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

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



```

IF sourceA >= sourceB THEN
    <statements>;

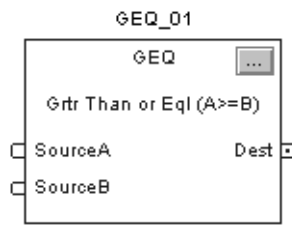
```

### Structured Text

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

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





### Function Block

Operand	Type	Format	Description
GEQ tag	FBD_COMPARE	structure	GEQ structure

### FBD\_COMPARE Structure

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

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

When you compare strings:

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

cuu duong than

l  
e  
s  
s  
e  
r

g  
r  
e  
a  
t  
e  
r

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

— AB < B  
— a > B

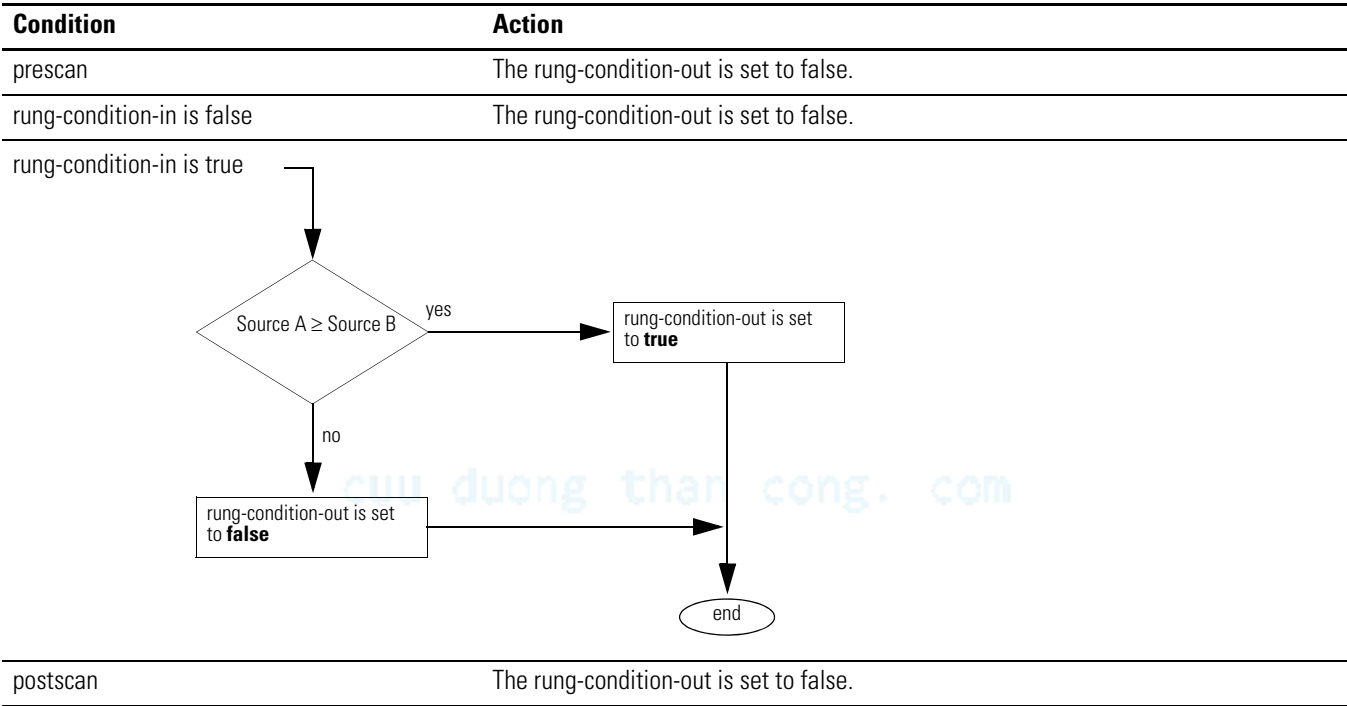
**Arithmetic Status Flags:** not affected

**Fault Conditions:** none

**Execution:**



**Relay Ladder**



**Function Block**

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

**Example:** If *value\_1* is greater than or equal to *value\_2*, set *light\_b*. If *value\_1* is less than *value\_2*, clear *light\_b*.

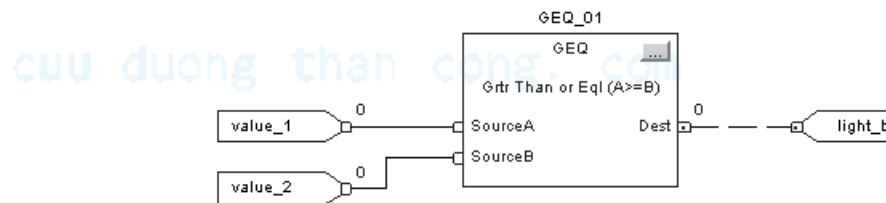
### Relay Ladder



### Structured Text

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

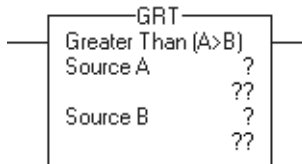
### Function Block



## Greater Than (GRT)

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

### Operands:



### Relay Ladder

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

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

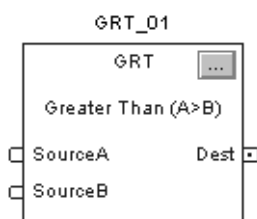


```
IF sourceA > sourceB THEN
    <statements>;
```

### Structured Text

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

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



### Function Block

Operand	Type	Format	Description
GRT tag	FBD_COMPARE	structure	GRT structure

**FBD\_COMPARE Structure**

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

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

When you compare strings:

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

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

↑ less  
 ↓ greater

— AB < B  
 — a > B

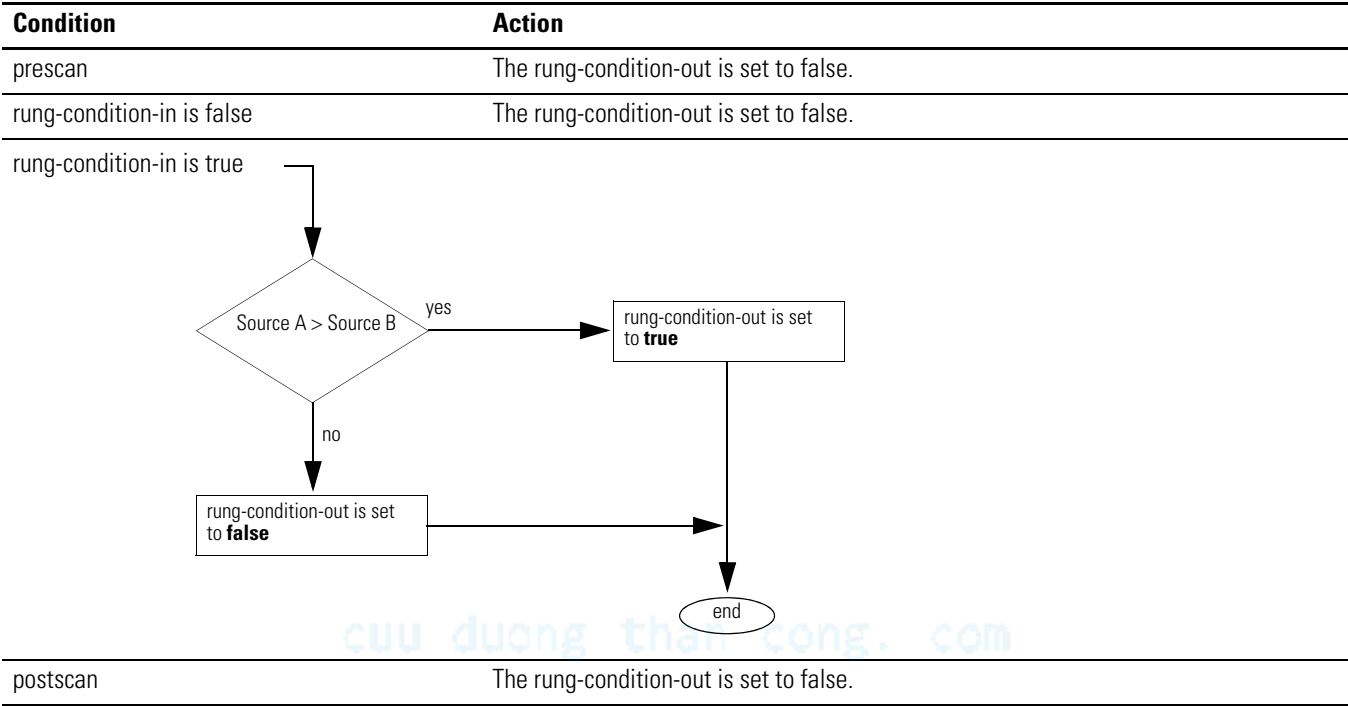
**Arithmetic Status Flags:** not affected

**Fault Conditions:** none

**Execution:**



**Relay Ladder**



**Function Block**

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

**Example:** If *value\_1* is greater than *value\_2*, set *light\_1*. If *value\_1* is less than or equal to *value\_2*, clear *light\_1*.

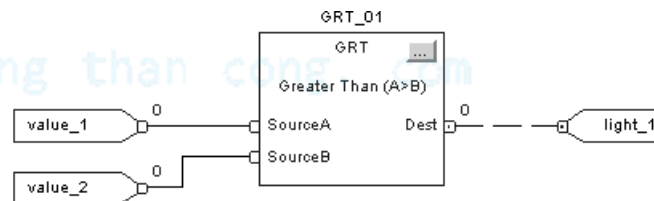
### Relay Ladder



### Structured Text

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

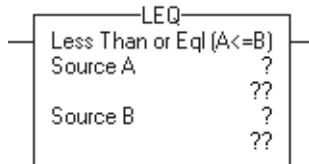
### Function Block



## Less Than or Equal to (LEQ)

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

### Operands:



### Relay Ladder

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

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



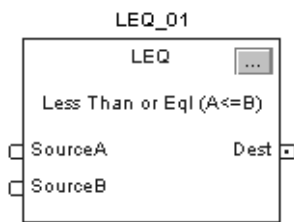
```
IF sourceA <= sourceB THEN
  <statements>;
```

### Structured Text

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

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





### Function Block

Operand	Type	Format	Description
LEQ tag	FBD_COMPARE	structure	LEQ structure

### FBD\_COMPARE Structure

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

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

When you compare strings:

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

cuu duong than

l  
e  
s  
s  
e  
r

g  
r  
e  
a  
t  
e  
r

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

— AB < B  
— a > B

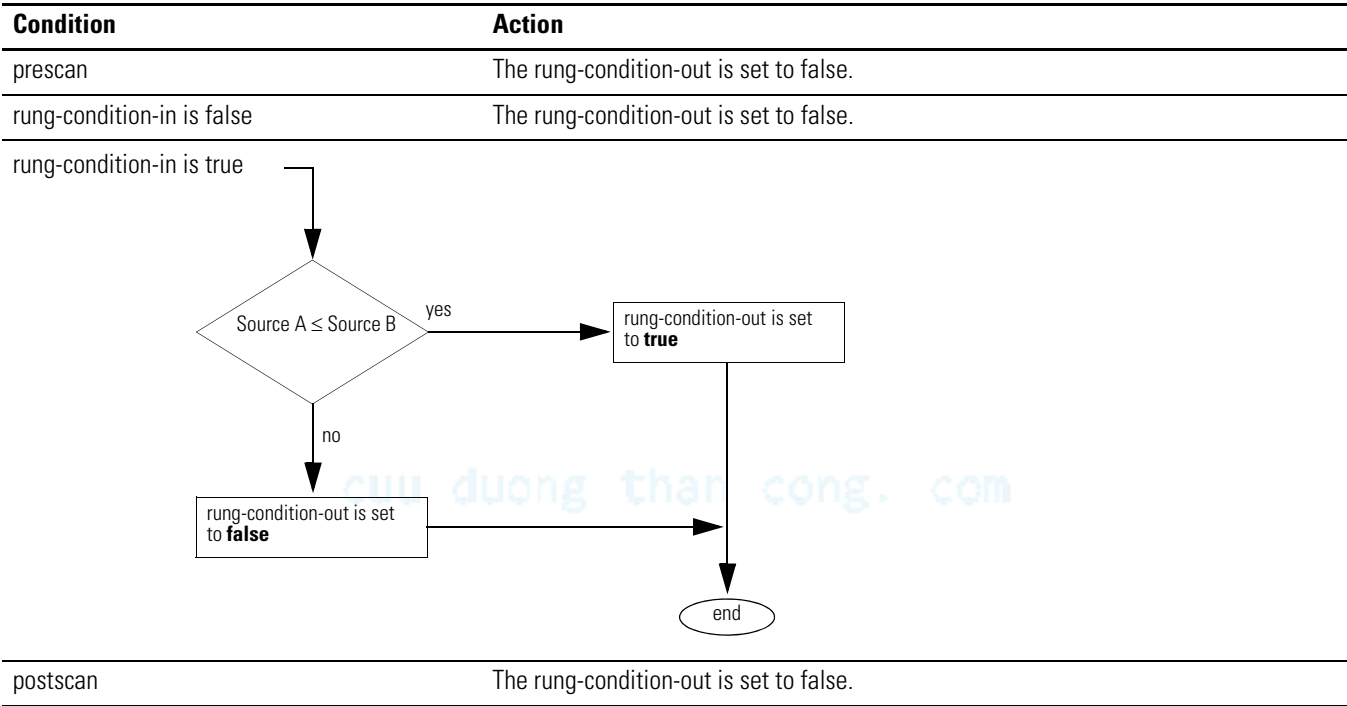
**Arithmetic Status Flags:** not affected

**Fault Conditions:** none

**Execution:**



**Relay Ladder**



**Function Block**

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

**Example:** If *value\_1* is less than or equal to *value\_2*, set *light\_2*. If *value\_1* is greater than *value\_2*, clear *light\_2*.

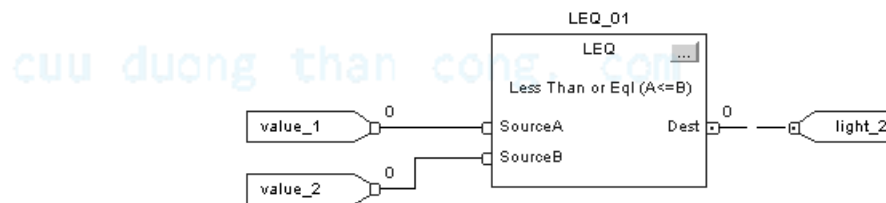
### Relay Ladder



### Structured Text

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

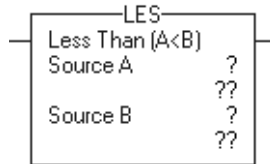
### Function Block



## 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	immediate	value to test against Source B
	INT	tag	
	DINT		
	REAL		
Source B	string		
	SINT	immediate	value to test against Source A
	INT	tag	
	DINT		
	REAL		
	string		

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

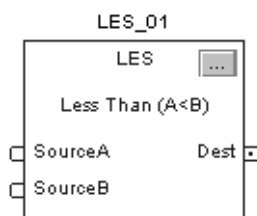


```
IF sourceA < sourceB THEN
    <statements>;
```

### Structured Text

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

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



### Function Block

Operand	Type	Format	Description
LES tag	FBD_COMPARE	structure	LES structure

**FBD\_COMPARE Structure**

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

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

When you compare strings:

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

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

↑ less  
 ↓ greater

— AB < B  
 — a > B

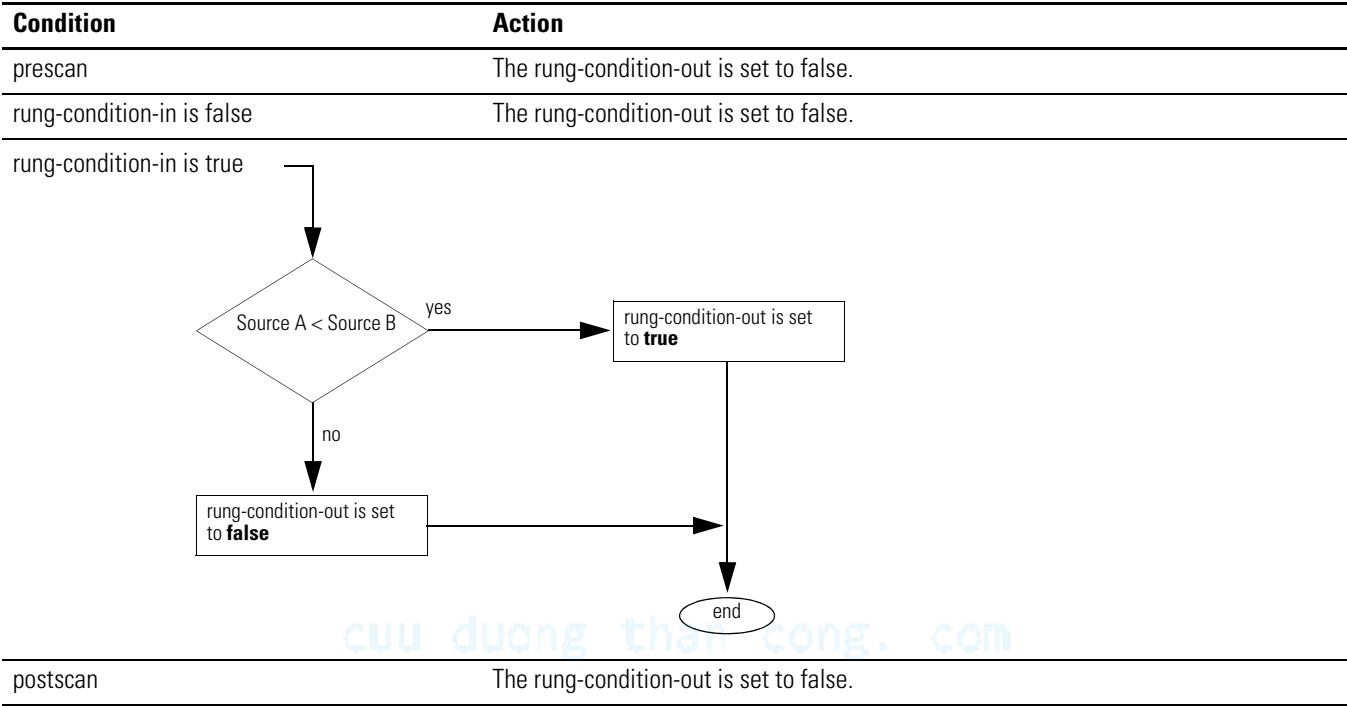
**Arithmetic Status Flags:** not affected

**Fault Conditions:** none

**Execution:**



**Relay Ladder**



**Function Block**

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

**Example:** If *value\_1* is less than *value\_2*, set *light\_3*. If *value\_1* is greater than or equal to *value\_2*, clear *light\_3*.

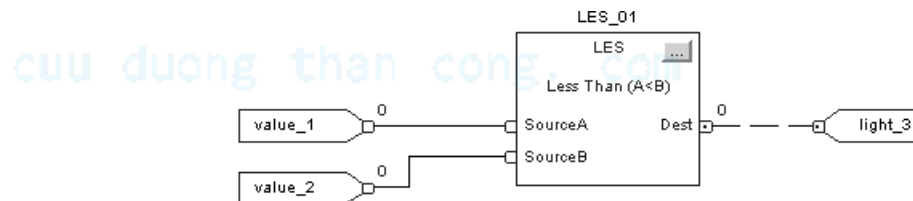
### Relay Ladder



### Structured Text

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

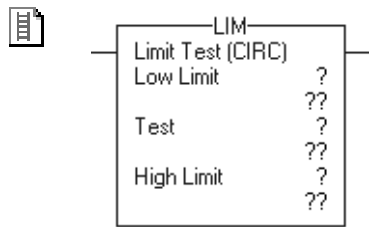
### Function Block



# Limit (LIM)

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

## Operands:



## Relay Ladder

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



## Structured Text

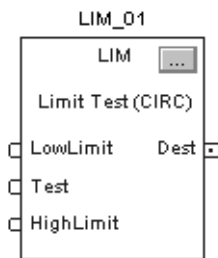
Structured text does not have a LIM instruction, but you can achieve the same results using structured text.

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

    <statement>;

END_IF;
```





## Function Block

Operand	Type	Format	Description
LIM tag	FBD_LIMIT	structure	LIM structure

## FBD\_LIMIT Structure

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

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

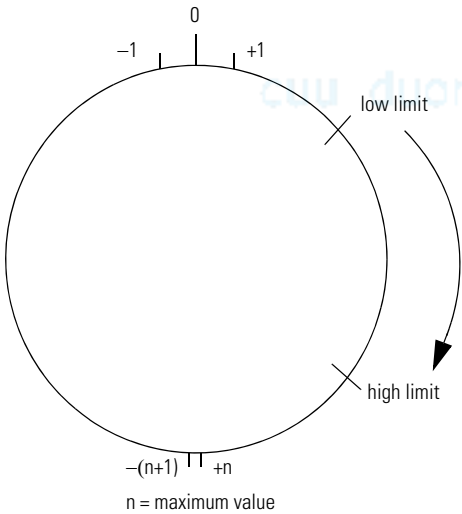
If Low Limit	And Test Value Is	The Rung-condition-out Is
$\leq$ High Limit	equal to or between limits	true
	not equal to or outside limits	false
$\geq$ High Limit	equal to or outside limits	true
	not equal to or inside limits	false

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

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

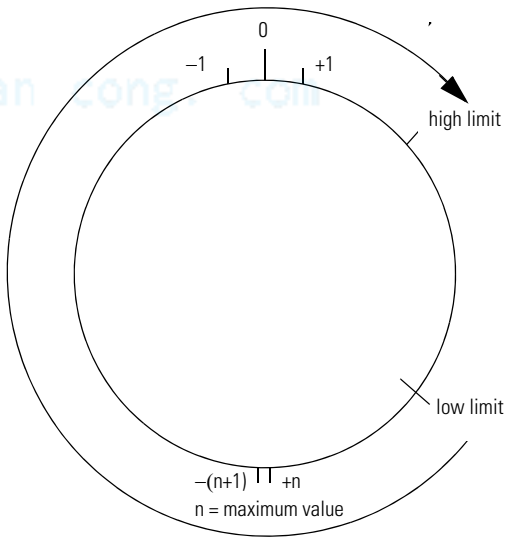
**Low Limit ≤ High Limit**

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



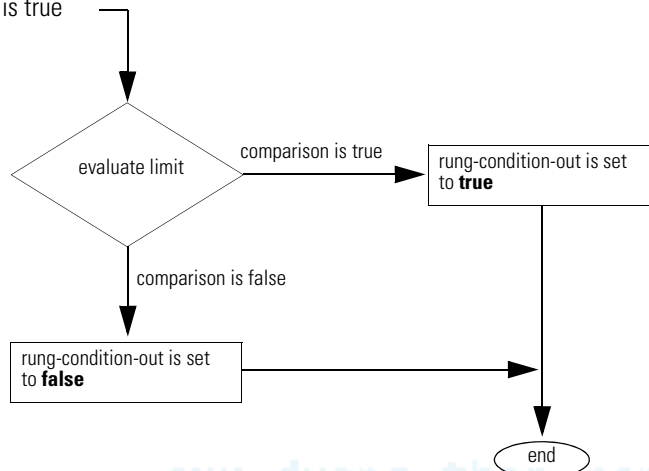
**Arithmetic Status Flags:** not affected

**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.

rung-condition-in is 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, the instruction does nothing, and the outputs are not updated.
EnableIn is set	The instruction executes. EnableOut is set.
postscan	No action taken.

**Example 1: Low Limit ≤ High Limit:**

When  $0 \leq \text{value} \leq 100$ , set *light\_1*. If *value* < 0 or *value* > 100, clear *light\_1*.

**Relay Ladder**



**Structured Text**

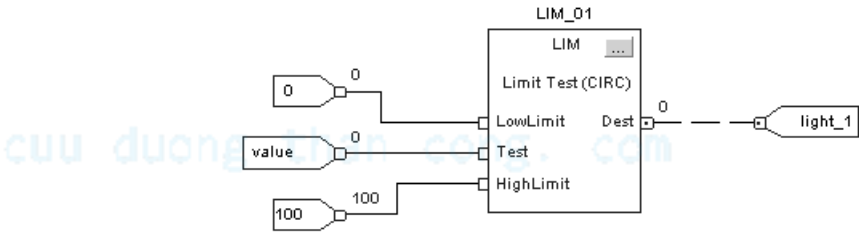
```
IF (value <= 100 AND (value >= 0 AND value <= 100)) OR
   (value >= 100 AND value <= 0 OR value >= 100)) THEN

    light_1 := 1;
ELSE

    light_1 := 0;

END_IF;
```

**Function Block**



**Example 2: Low Limit  $\geq$  High Limit:**

When  $value \geq 0$  or  $value \leq -100$ , set *light\_1*. If  $value < 0$  or  $value > -100$ , clear *light\_1*.

**Relay Ladder****Structured Text**

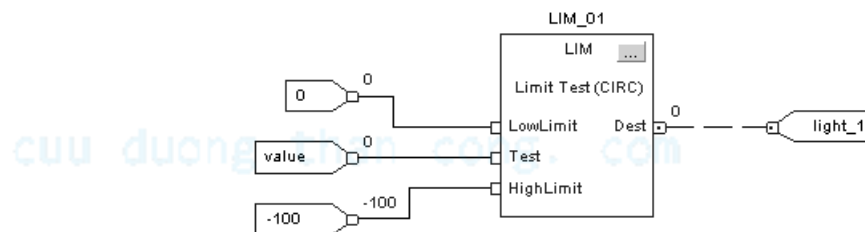
```
IF (0 <= -100 AND value >= 0 AND value <= -100)) OR
   (0 >= -100 AND (value <= 0 OR value >= -100)) THEN
```

```
    light_1 := 1;
```

```
ELSE
```

```
    light_1 := 0;
```

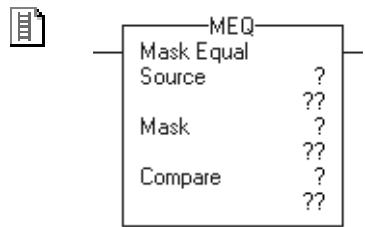
```
END_IF;
```

**Function Block**

# Mask Equal to (MEQ)

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

## Operands:



## Relay Ladder

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



## Structured Text

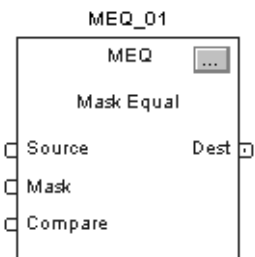
Structured text does not have an MEQ instruction, but you can achieve the same results using structured text.

```
IF (Source AND Mask) = (Compare AND Mask) THEN
```

```
    <statement>;
```

```
END_IF;
```

## Function Block



Operand	Type	Format	Description
MEQ tag	FBD_MASK_EQUAL	structure	MEQ structure

**FBD\_MASK\_EQUAL Structure**

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

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

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

**Entering an Immediate Mask Value**

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

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

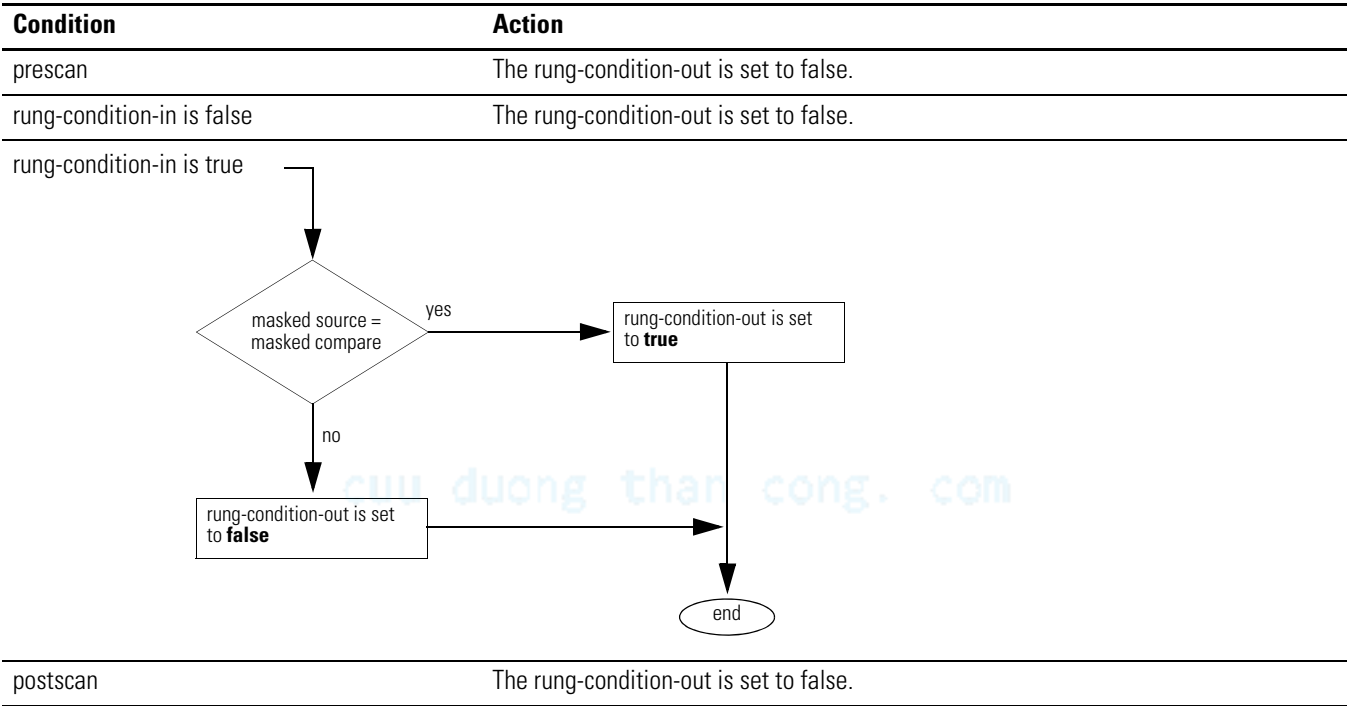
**Arithmetic Status Flags:** not affected

**Fault Conditions:** none

**Execution:**



**Relay Ladder**



**Function Block**

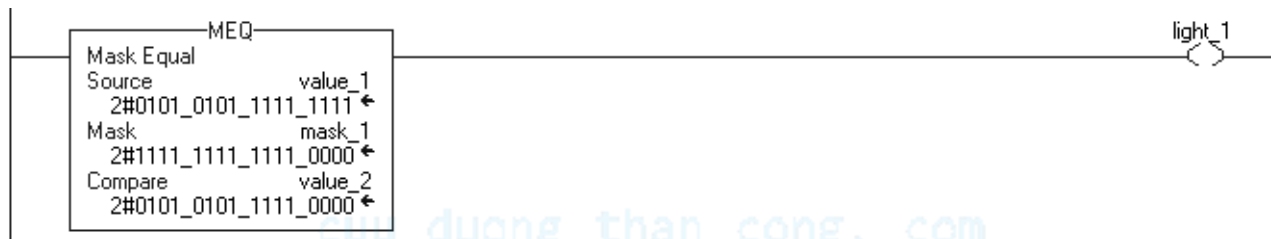
Condition	Action
prescan	No action taken.
instruction first scan	No action taken.
instruction first run	No action taken.
EnableIn is cleared	EnableOut is cleared, the instruction does nothing, and the outputs are not updated.
EnableIn is set	The instruction executes.
	EnableOut is set.
postscan	No action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



**Example 1:** If the masked *value\_1* is equal to the masked *value\_2*, set *light\_1*. If the masked *value\_1* is not equal to the masked *value\_2*, clear *light\_1*. This example shows that the masked values are equal. A 0 in the mask restrains the instruction from comparing that bit (shown by x in the example).

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

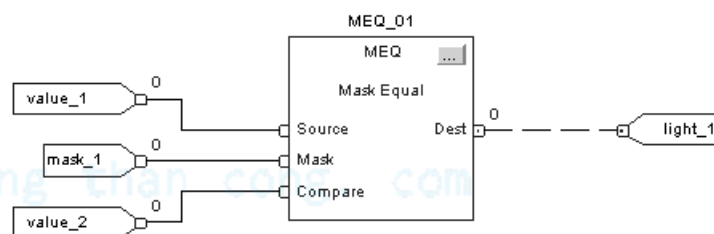
### Relay Ladder



### Structured Text

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

### Function Block

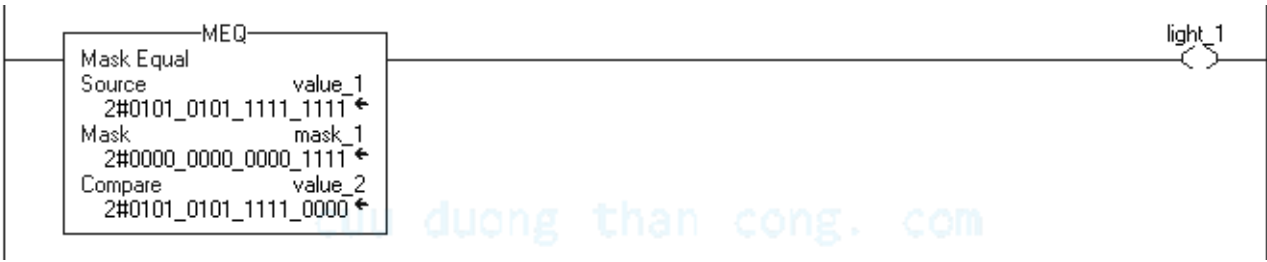


**Example 2:** If the masked *value\_1* is equal to the masked *value\_2*, set *light\_1*. If the masked *value\_1* is not equal to the masked *value\_2*, clear *light\_1*. This example shows that the masked values are not equal. A 0 in the mask restrains the instruction from comparing that bit (shown by x in the example).

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

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

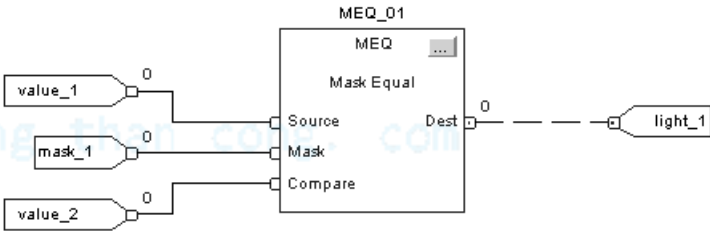
### Relay Ladder



### Structured Text

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

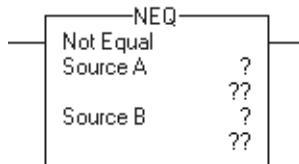
### Function Block



## 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	immediate	value to test against Source B
	INT	tag	
	DINT		
	REAL		
	string		
Source B	SINT	immediate	value to test against Source A
	INT	tag	
	DINT		
	REAL		
	string		

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

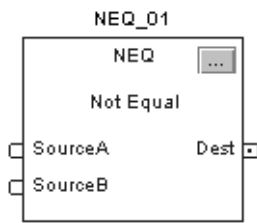


```
IF sourceA <> sourceB THEN
  <statements>;
```

### Structured Text

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

See Appendix 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

lesser

↑

greater

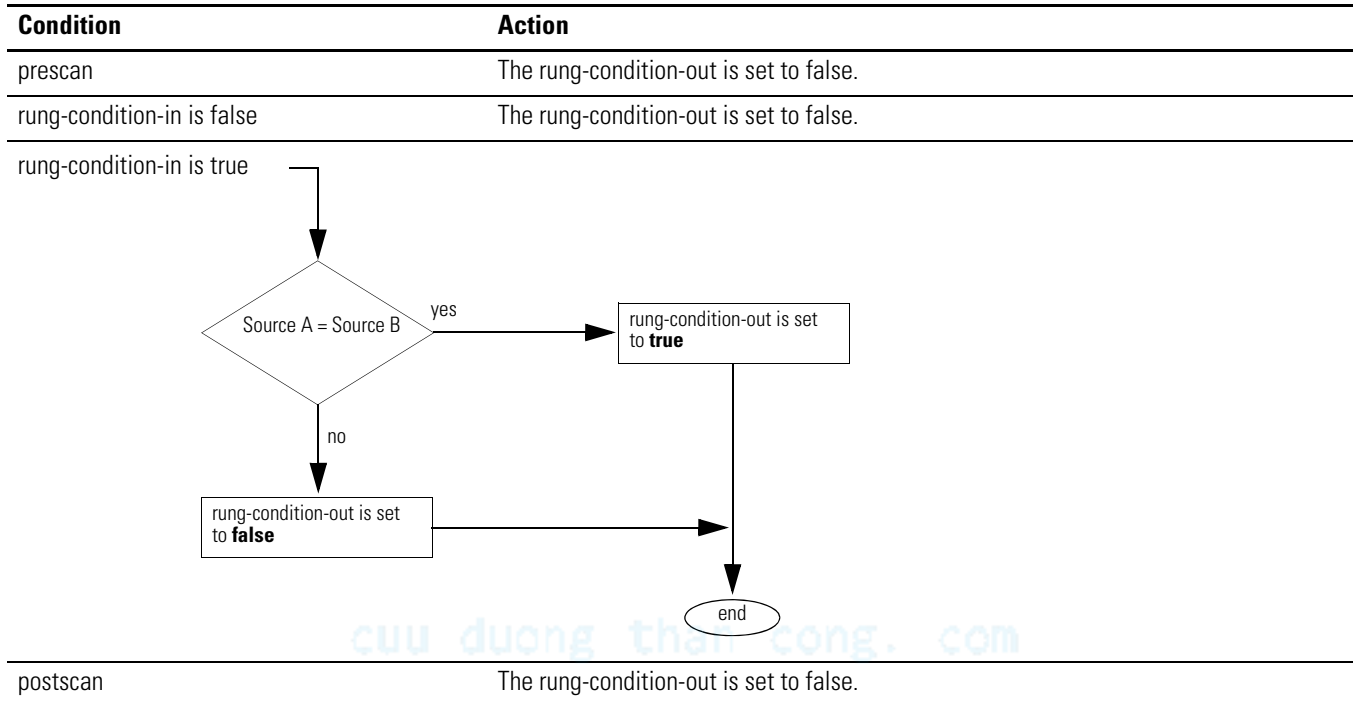
↓

— AB < B

— a > B

**Arithmetic Status Flags:** not affected

**Fault Conditions:** none

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

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

**Example:** If *value\_1* is not equal to *value\_2*, set *light\_4*. If *value\_1* is equal to *value\_2*, clear *light\_4*.

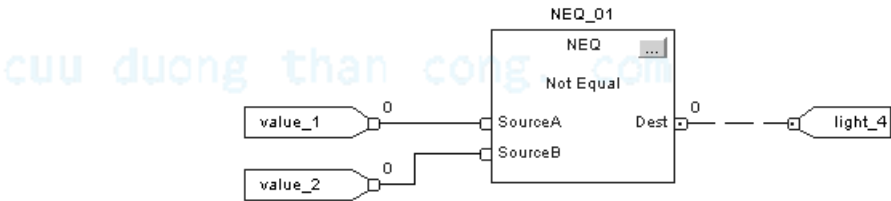
**Relay Ladder**



**Structured Text**

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

**Function Block**



## Compute/Math Instructions

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

### Introduction

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

If You Want To	Use This Instruction	Available In These Languages	See Page
evaluate an expression	CPT	relay ladder structured text <sup>(1)</sup>	249
add two values	ADD	relay ladder structured text <sup>(2)</sup> function block	253
subtract two values	SUB	relay ladder structured text <sup>(2)</sup> function block	257
multiply two values	MUL	relay ladder structured text <sup>(2)</sup> function block	260
divide two values	DIV	relay ladder structured text <sup>(2)</sup> function block	263
determine the remainder after one value is divided by another	MOD	relay ladder structured text <sup>(2)</sup> function block	268

If You Want To	Use This Instruction	Available In These Languages	See Page
calculate the square root of a value	SQR	relay ladder	272
	SQRT <sup>(3)</sup>	structured text	
		function block	
take the opposite sign of a value	NEG	relay ladder	276
		structured text <sup>(2)</sup>	
		function block	
take the absolute value of a value	ABS	relay ladder	279
		structured text	
		function block	

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

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

<sup>(3)</sup> Structured text only.

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

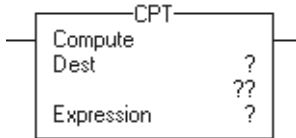
For relay ladder instructions, **bold** data types indicate optimal data types. An instruction executes faster and requires less memory if all the operands of the instruction use the same optimal data type, typically DINT or REAL.



## Compute (CPT)

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

### Operands:



### Relay Ladder

Operand	Type	Format:	Description
Destination	SINT INT <b>DINT</b> <b>REAL</b>	tag	tag to store the result
Expression	SINT INT <b>DINT</b> <b>REAL</b>	immediate tag	an expression consisting of tags and/or immediate values separated by operators
A SINT or INT tag converts to a DINT value by sign-extension.			



### Structured Text

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

```
destination := numeric_expresion;
```

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

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

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

#### TIP

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

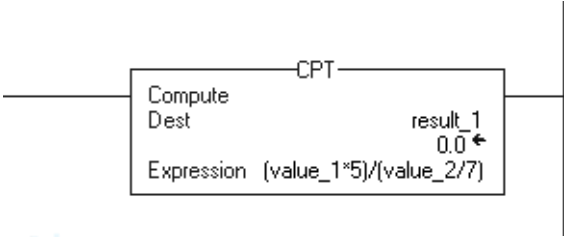
**Arithmetic Status Flags:** Arithmetic status flags are affected.

**Fault Conditions:** none

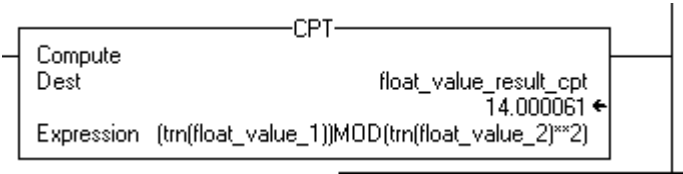
**Execution:**

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

**Example 1:** When enabled, the CPT instruction evaluates *value\_1* multiplied by 5 and divides that result by the result of *value\_2* divided by 7 and places the final result in *result\_1*.



**Example 2:** When enabled, the CPT instruction truncates *float\_value\_1* and *float\_value\_2*, raises the truncated *float\_value\_2* to the power of two and divides the truncated *float\_value\_1* by that result, and stores the remainder after the division in *float\_value\_result\_cpt*.



## Valid operators

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

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

## 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( <i>tag_a</i> )
two operands	operand_a operator operand_b	<ul style="list-style-type: none"> <li><i>tag_b</i> + 5</li> <li><i>tag_c</i> AND <i>tag_d</i></li> <li>(<i>tag_e</i> ** 2) MOD (<i>tag_f</i> / <i>tag_g</i>)</li> </ul>

## 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.

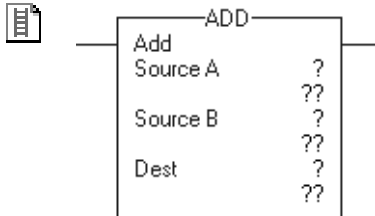
Operations of equal order are performed from left to right.

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

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



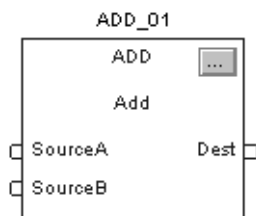
```
dest := sourceA + sourceB;
```

### Structured Text

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

See Appendix 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**

<b>Input Parameter:</b>	<b>Data Type:</b>	<b>Description:</b>
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
<b>Output Parameter:</b>	<b>Data Type:</b>	<b>Description:</b>
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**

<b>Condition:</b>	<b>Action:</b>
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

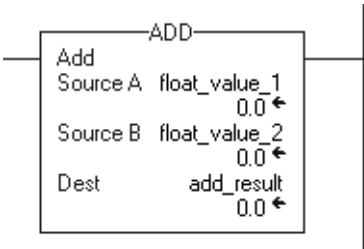
Condition:	Action:
prescan	No action taken.
instruction first scan	No action taken.
instruction first run	No action taken.
EnableIn is cleared	EnableOut is cleared.
EnableIn is set	The instruction executes.
	EnableOut is set.
postscan	No action taken.

cuu duong than cong. com

cuu duong than cong. com

**Example:** Add *float\_value\_1* to *float\_value\_2* and place the result in *add\_result*.

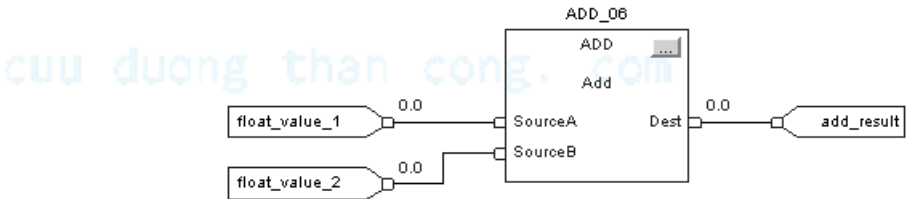
**Relay Ladder**



**Structured Text**

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

**Function Block**

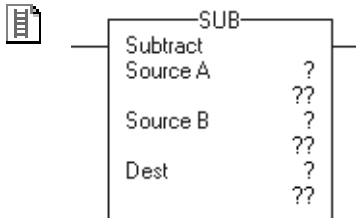




## Subtract (SUB)

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



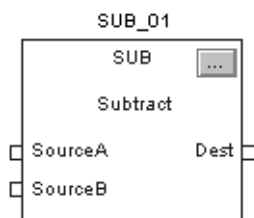
```
dest := sourceA - sourceB;
```

### Structured Text

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

See Appendix B 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**

<b>Input Parameter:</b>	<b>Data Type:</b>	<b>Description:</b>
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
<b>Output Parameter:</b>	<b>Data Type:</b>	<b>Description:</b>
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**

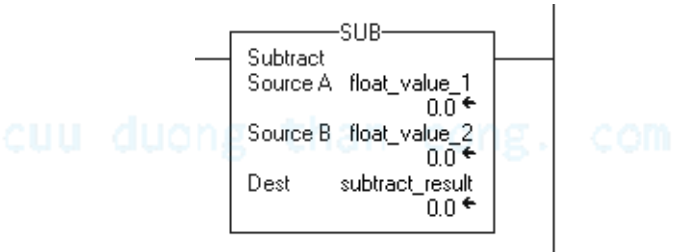
<b>Condition:</b>	<b>Action:</b>
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	Destination = Source B - Source A  The rung-condition-out is set to true.
postscan	The rung-condition-out is set to false.

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*.

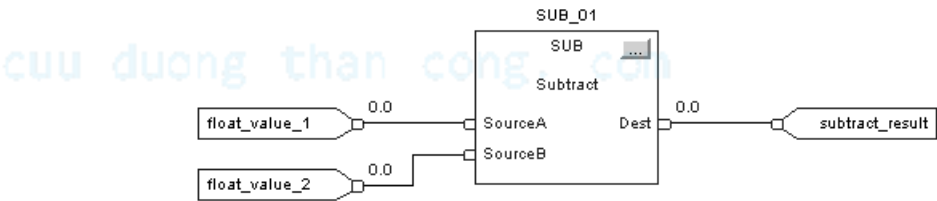
Relay Ladder



Structured Text

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

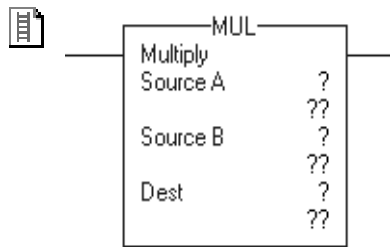
Function Block



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

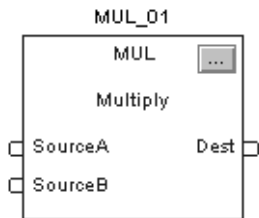


```
dest := sourceA * sourceB;
```

## Structured Text

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

See Appendix B 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**

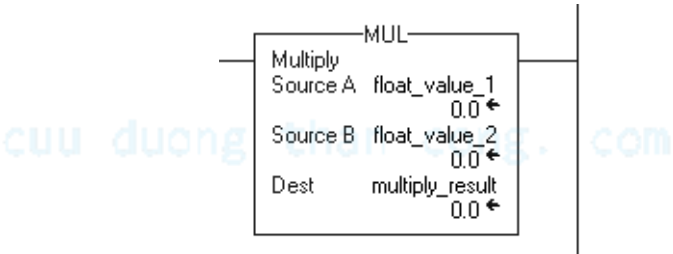
Condition	Action
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	Destination = Source B x Source A The rung-condition-out is set to true.
postscan	The rung-condition-out is set to false.

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*.

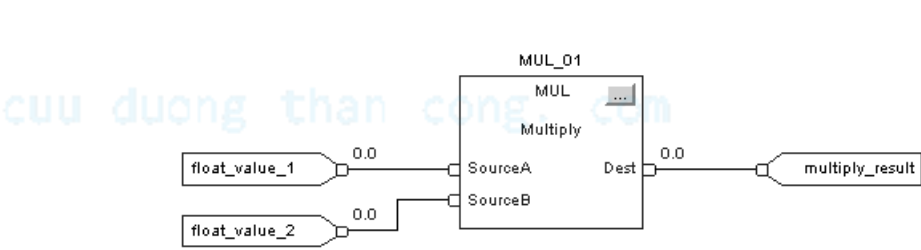
Relay Ladder



Structured Text

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

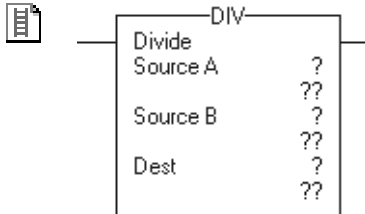
Function Block



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

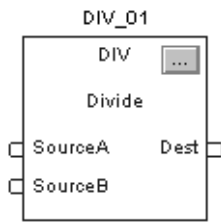


```
dest := sourceA / sourceB;
```

### Structured Text

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

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



### Function Block

Operand	Type	Format	Description
DIV tag	FBD_MATH	structure	DIV structure

### FBD\_MATH Structure

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

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

If Source A	Then The Fractional Portion Of The Result	Example
and Source B are <i>not</i> REALs	truncates	Source A     DINT     5
		Source B     DINT     3
		Destination   DINT     1
or Source B is a REAL	rounds	Source A     REAL     5.0
		Source B     DINT     3
		Destination   DINT     2



If Source B (the divisor) is zero:

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

If Source B Is Zero And:	And The Destination Is a:	And The Result Is:	Then The Destination Is Set To:
all operands are integers (SINT, INT, or DINT)	—————→	—————→	Source A
at least one operand is a REAL	SINT, INT, or DINT	positive	-1
		negative	0
	REAL	positive	1.\$ (positive infinity)
		negative	-1.\$ (negative infinity)

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

**Arithmetic Status Flags:** Arithmetic status flags are affected.

**Fault Conditions:**

A Minor Fault Occurs If	Fault Type	Fault Code
the divisor is zero	4	4

**Execution:**

### Relay Ladder

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

### Function Block

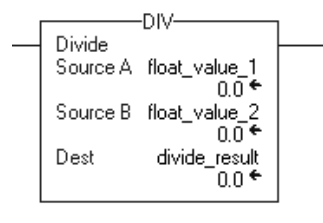
Condition	Action
prescan	No action taken.
instruction first scan	No action taken.
instruction first run	No action taken.
EnableIn is cleared	EnableOut is cleared.
EnableIn is set	The instruction executes.
	EnableOut is set.
postscan	No action taken.

cuu duong than cong. com

cuu duong than cong. com

**Example 1:** Divide *float\_value\_1* by *float\_value\_2* and place the result in *divide\_result*.

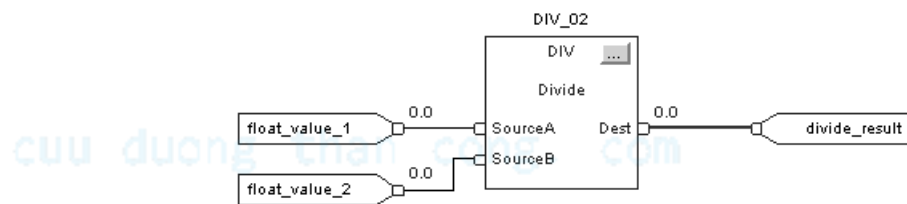
### Relay Ladder



### Structured Text

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

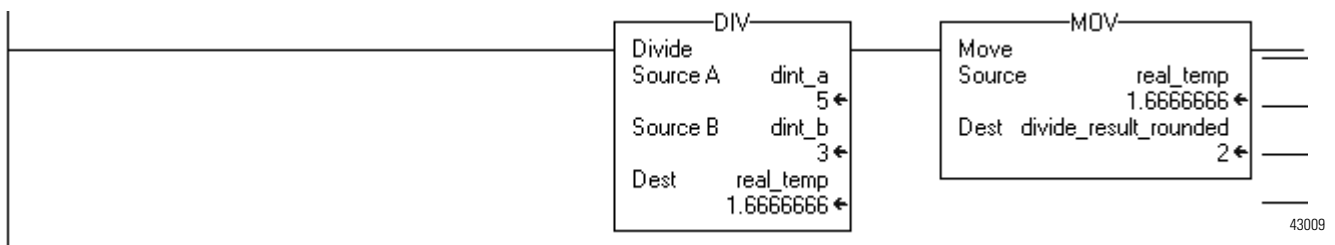
### Function Block



**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.

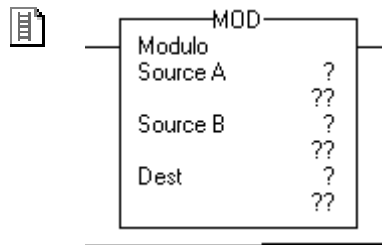
### Relay Ladder



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

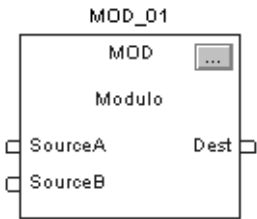


`dest := sourceA MOD sourceB;`

## Structured Text

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

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



## Function Block

Operand	Type	Format	Description
MOD tag	FBD_MATH	structure	MOD structure

**FBD\_MATH Structure**

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

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

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

If Source B Is Zero And	And The Destination Is a	And The Result Is	Then The Destination Is Set To
all operands are integers (SINT, INT, or DINT)	—————→	—————→	Source A
at least one operand is a REAL	SINT, INT, or DINT	positive	-1
		negative	0
	REAL	positive	1.\$ (positive infinity)
		negative	-1.\$ (negative infinity)

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

**Arithmetic Status Flags:** Arithmetic status flags are affected.

**Fault Conditions:**

A Minor Fault Occurs If	Fault Type	Fault Code
the divisor is zero	4	4

**Execution:**



**Relay Ladder**

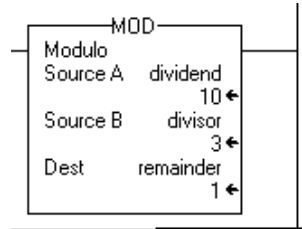
Condition	Action
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	Destination = Source A – ( TRN ( Source A / Source B ) * Source B ) The rung-condition-out is set to true.
postscan	The rung-condition-out is set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

**Function Block**

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

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

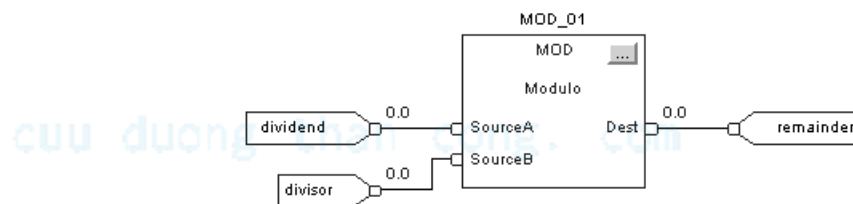
### Relay Ladder



### Structured Text

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

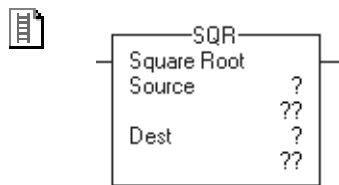
### Function Block



# Square Root (SQR)

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

## Operands:



## Relay Ladder

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



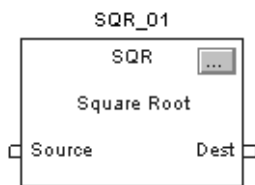
dest := SQR(source);

## Structured Text

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

See Appendix B 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**

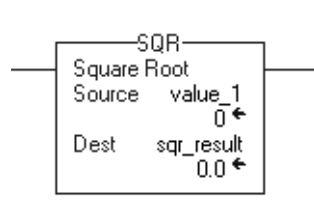
Condition	Action
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	$Destination = \sqrt{Source}$
	The rung-condition-out is set to true.
postscan	The rung-condition-out is set to false.

**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*.

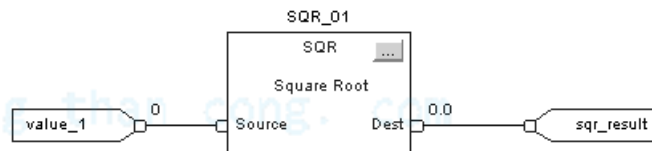
### Relay Ladder



### Structured Text

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

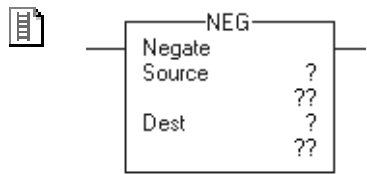
### Function Block



# 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	immediate	value to negate
	INT	tag	
	DINT		
Destination	REAL		
	A SINT or INT tag converts to a DINT value by sign-extension.		
	SINT	tag	tag to store the result
	INT		
	DINT		
	REAL		

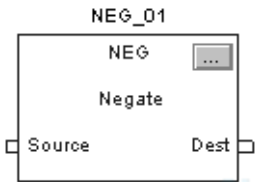


dest := -source;

## Structured Text

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

See Appendix B 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:** [fb.com/tailieudientucntt](https://fb.com/tailieudientucntt)



### Relay Ladder

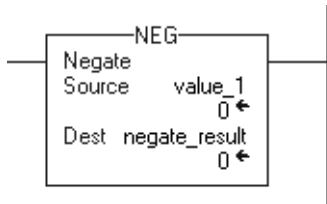
Condition	Action
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	Destination = 0 – Source The rung-condition-out is set to true.
postscan	The rung-condition-out is set to false.

### 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*.

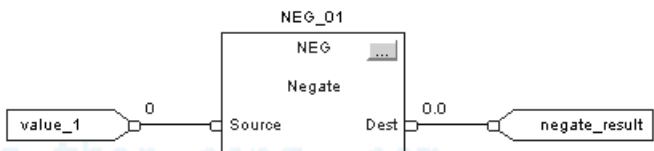
**Relay Ladder**



**Structured Text**

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

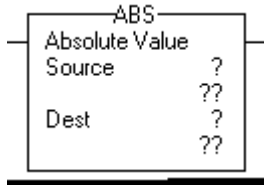
**Function Block**



## 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	immediate	value of which to take the absolute value
	INT	tag	
	DINT		
	REAL		
A SINT or INT tag converts to a DINT value by sign-extension.			
Destination	SINT	tag	tag to store the result
	INT		
	DINT		
	REAL		



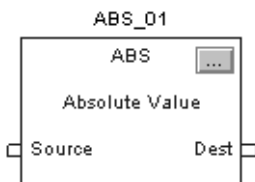
```
dest := ABS(source);
```

### Structured Text

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

See Appendix B 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:** [cuu duong than cong. com](https://fb.com/tailieudientucntt)

**Relay Ladder**

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

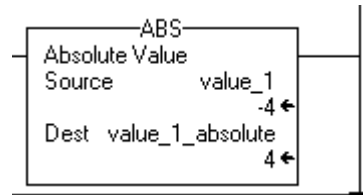
**Function Block**

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



**Example:** Place the absolute value of *value\_1* into *value\_1\_absolute*. In this example, the absolute value of negative four is positive four.

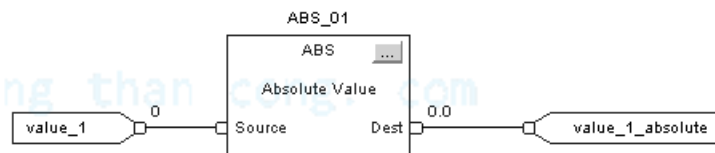
### Relay Ladder



### Structured Text

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

### Function Block



## Notes:

cuu duong than cong. com

cuu duong than cong. com

## Move/Logical Instructions

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

### Introduction

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

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

The move instructions modify and move bits.

If you want to	Use this instruction	Available in these languages	See page
copy a value	MOV	relay ladder structured text <sup>(1)</sup>	285
copy a specific part of an integer	MVM	relay ladder	287
copy a specific part of an integer in function block	MVMT	structured text function block	290
move bits within an integer or between integers	BT	relay ladder	293
move bits within an integer or between integers in function block	BTDT	structured text function block	296
clear a value	CLR	structured text <sup>(1)</sup> relay ladder	299
rearrange the bytes of a INT, DINT, or REAL tag	SWPB	relay ladder structured text	301

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

The logical instructions perform operations on bits.

If you want to:	Use this instruction:	Available in these languages	See page
bitwise AND operation	Bitwise AND & <sup>(1)</sup>	relay ladder	305
		structured text <sup>(2)</sup>	
		function block	
bitwise OR operation	Bitwise OR	relay ladder	308
		structured text <sup>(2)</sup>	
		function block	
bitwise, exclusive OR operation	Bitwise XOR	relay ladder	311
		structured text <sup>(2)</sup>	
		function block	
bitwise NOT operation	Bitwise NOT	relay ladder	315
		structured text <sup>(2)</sup>	
		function block	
logically AND as many as eight boolean inputs.	Boolean AND (BAND)	structured text <sup>(2)</sup>	319
		function block	
logically OR as many as eight boolean inputs.	Boolean OR (BOR)	structured text <sup>(2)</sup>	322
		function block	
perform an exclusive OR on two boolean inputs.	Boolean Exclusive OR (BXOR)	structured text <sup>(2)</sup>	325
		function block	
complement a boolean input.	Boolean NOT (BNOT)	structured text <sup>(2)</sup>	328
		function block	

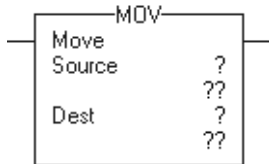
<sup>(1)</sup> Structured text only.

<sup>(2)</sup> 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	immediate	value to move (copy)
	INT	tag	
	DINT		
	REAL		
A SINT or INT tag converts to a DINT value by sign-extension.			
Destination	SINT	tag	tag to store the result
	INT		
	DINT		
	REAL		



```
dest := source;
```

### Structured Text

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

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

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

**Arithmetic Status Flags:** Arithmetic status flags are affected.

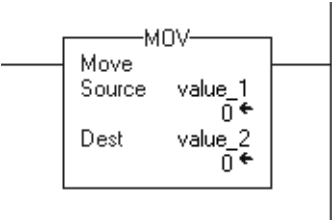
**Fault Conditions:** none

**Execution:**

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

**Example:** Move the data in *value\_1* to *value\_2*.

**Relay Ladder**



**Structured Text**

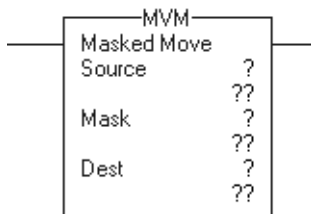
```
value_2 := value _1;
```

## Masked Move (MVM)

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

This instruction is available in structured text and function block as MVMT, see page 290.

### Operands:



### Relay Ladder

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



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

### Structured Text

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

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

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

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

## 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** Arithmetic status flags are affected.

**Fault Conditions** none

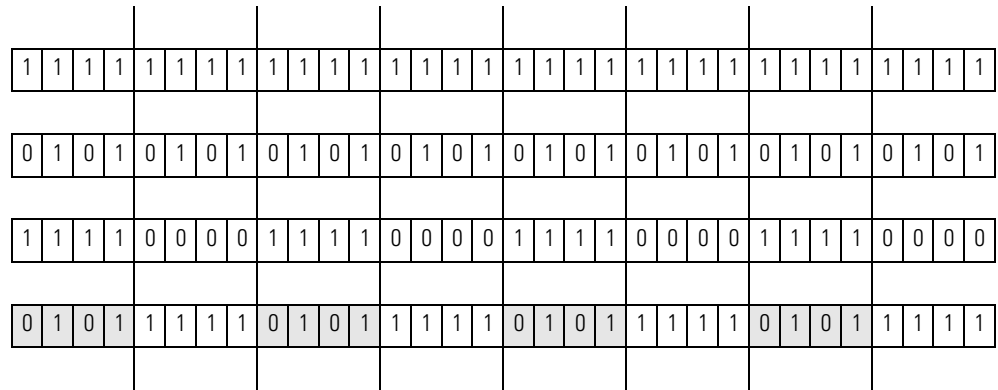
**Execution:** duong than cong. com

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.

cuu duong than cong. com

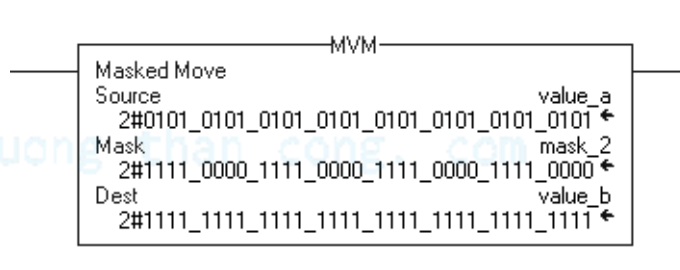


**Example:** Copy data from *value\_a* to *value\_b*, while allowing data to be masked (a 0 masks the data in *value\_a*).



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 13-287.

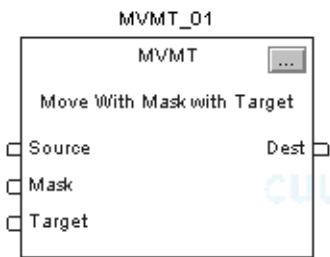
## Operands:



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	<p><b>Function Block</b></p> <p>If cleared, the instruction does not execute and outputs are not updated.</p> <p>If set, the instruction executes.</p> <p>Default is set.</p> <p><b>Structured Text</b></p> <p>No effect. The instruction executes.</p>
Source	DINT	<p>Input value to move to Destination based on value of Mask.</p> <p>Valid = any integer</p>
Mask	DINT	<p>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.</p> <p>Valid = any integer</p>
Target	DINT	<p>Input value to move to Dest prior to moving Source bits through the Mask.</p> <p>Valid = any integer</p>

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 Using an Input Reference

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

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

**Arithmetic Status Flags:** Arithmetic status flags are affected.

**Fault Conditions:** none

### Execution:

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

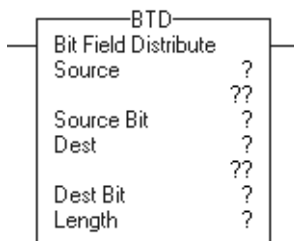


## Bit Field Distribute (BTD)

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 296.

### Operands:



### Relay Ladder

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

**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.

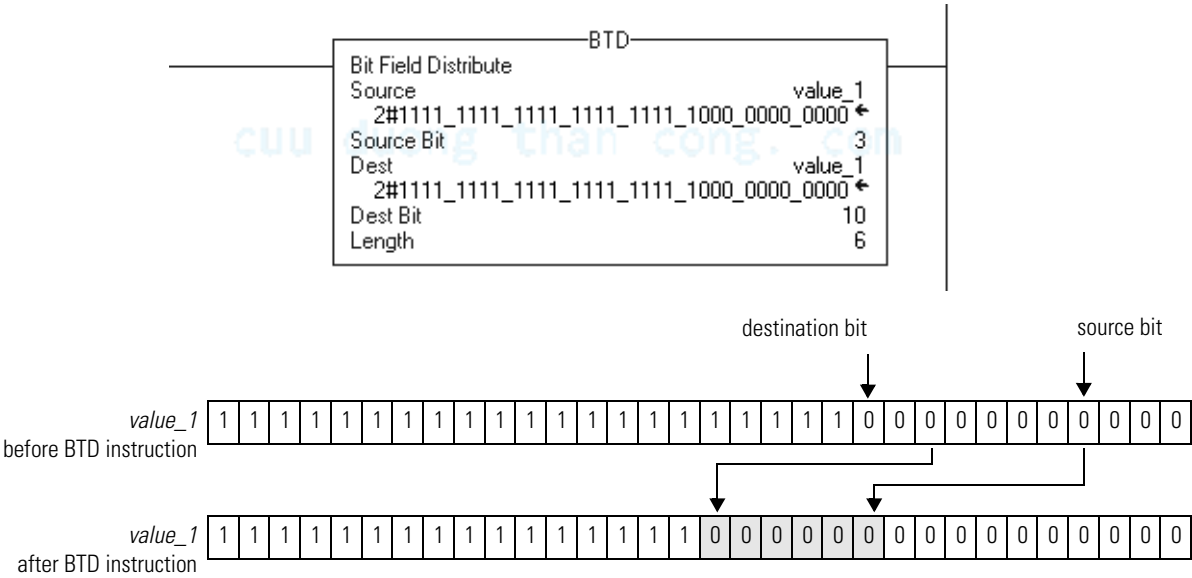
**Arithmetic Status Flags:** not affected

**Fault Conditions:** none

**Execution:**

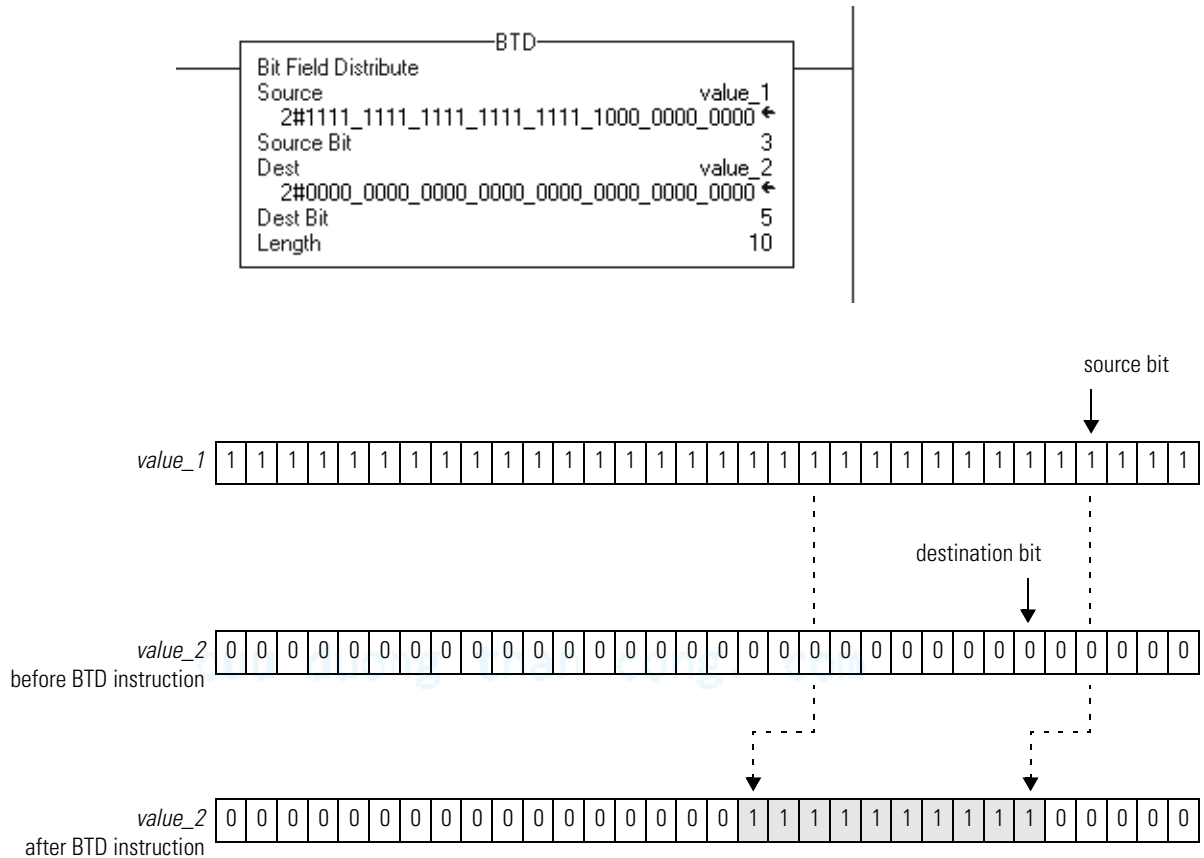
Condition	Relay Ladder Action
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	The instruction copies and shifts the Source bits to the Destination. The rung-condition-out is set to true.
postscan	The rung-condition-out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

cuu duong than cong. com

# Bit Field Distribute with Target (BTDT)

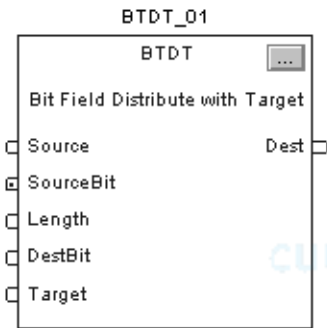
The BTDT instruction first copies the Target to the Destination. Then the instruction copies the specified bits from the Source, shifts the bits to the appropriate position, and writes the bits into the Destination. The Target and Source remain unchanged.

This instruction is available in relay ladder as BTD, see page 13-293.

## 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	<b>Function Block:</b>  If cleared, the instruction does not execute and outputs are not updated.  If set, the instruction executes.  Default is set.
Source	DINT	<b>Structured Text:</b>  No effect. The instruction executes.
SourceBit	DINT	Input value containing the bits to move to Destination.  Valid = any integer
Length	DINT	The bit position in Source (lowest bit number from where to start the move).  Valid = 0-31
		Number of bits to move  Valid = 1-32



Input Parameter	Data Type	Description:
DestBit	DINT	The bit position in Dest (lowest bit number to start copying bits into).  Valid = 0-31
Target	DINT	Input value to move to Dest prior to moving bits from the Source.  Valid = any integer
Output Parameter:	Data Type:	Description:
EnableOut	BOOL	The instruction produced a valid result.
Dest	DINT	Result of the bit move operation. Arithmetic status flags are set for this output.

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

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

**Arithmetic Status Flags:** Arithmetic status flags are affected

**Fault Conditions:** none

### Execution:

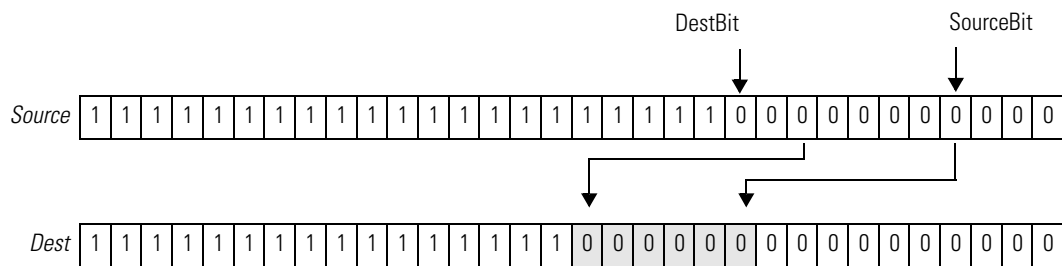
Condition	Function Block Action	Structured Text Action
prescan	No action taken.	No action taken.
instruction first scan	No action taken.	No action taken.
instruction first run	No action taken.	No action taken.
EnableIn is cleared	EnableOut is cleared, the instruction does nothing, and the outputs are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

**Example:** 1. The controller copies Target into Dest.

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

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

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



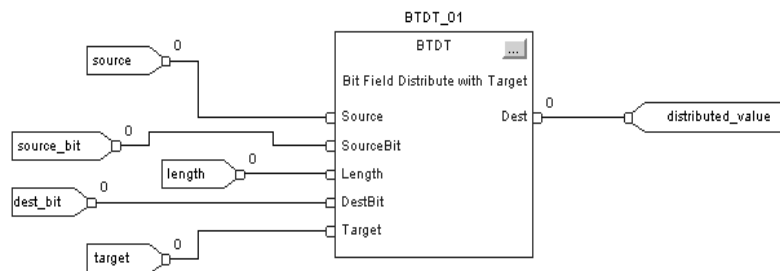
### Structured Text

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

```
BTDT(BTDT_01);
```

```
distributed_value := BTDT_01.Dest;
```

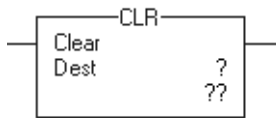
### Function Block



## Clear (CLR)

The CLR instruction clears all the bits of the Destination.

### Operands:



### Relay Ladder

Operand	Type	Format	Description
Destination	SINT	tag	tag to clear
	INT		
	DINT		
	REAL		



```
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.

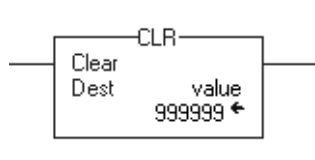
**Fault Conditions:** none

### 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;
```

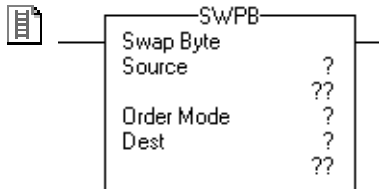
cuu duong than cong. com

cuu duong than cong. com

## Swap Byte (SWPB)

The SWPB instruction rearranges the bytes of a value.

### Operands:



### Relay Ladder

Operand	Type	Format	Enter
Source	INT	tag	tag that contains the bytes that you want to rearrange
	DINT		
	REAL		
Order Mode		<b>If the Source Is an</b>	<b>And You Want To Change the Bytes To This Pattern (Each Letter Represents a Different Byte)</b>
		INT	n/a
		DINT	ABCD $\Rightarrow$ DCBA
		REAL	ABCD $\Rightarrow$ CDAB
			ABCD $\Rightarrow$ BADC
Destination			<b>Then Select</b>
			any of the options
			REVERSE (or enter 0)
			WORD (or enter 1)
			HIGH/LOW (or enter 2)
		<b>If the Source Is an</b>	<b>Then the Destination Must Be an</b>
		INT	INT
			DINT
		DINT	DINT
		REAL	REAL



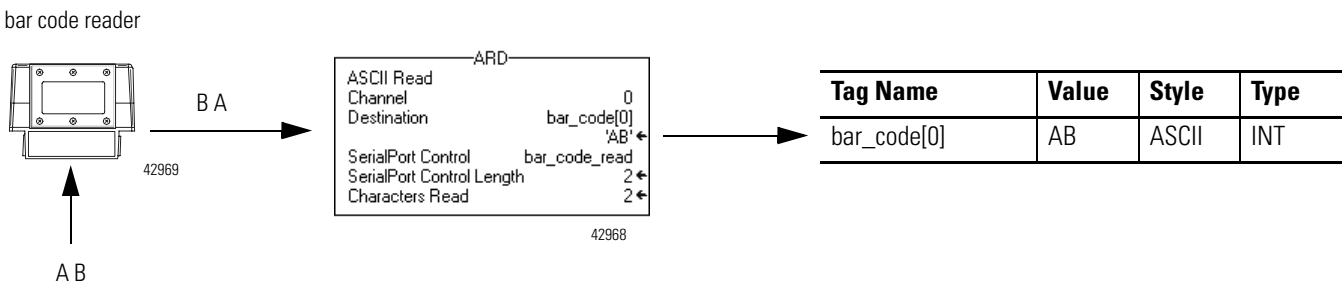
SWPB(Source, OrderMode, Dest);

### Structured Text

The operands are the same as those for the relay ladder SWPB instruction. If you select the HIGH/LOW order mode, enter it as HIGHLOW or HIGH\_LOW (without the slash).

**Description:** The SWPB instruction rearranges the order of the bytes of the Source. It places the result in the Destination.

When you read or write ASCII characters, you typically *do not* need to swap characters. The ASCII read and write instructions (ARD, ARL, AWA, AWT) automatically swap characters, as shown below.



**Arithmetic Status Flags:** not affected

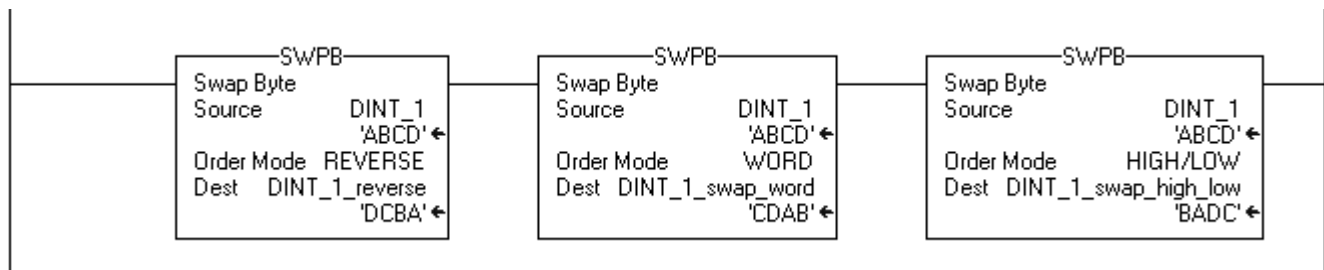
**Fault Conditions:** none

**Execution:**

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

**Example 1:** The three SWPB instructions each reorder the bytes of *DINT\_1* according to a different order mode. The display style is ASCII, and each character represents one byte. Each instruction places the bytes, in the new order, in a different Destination.

### Relay Ladder



### Structured Text

```

SWPB(DINT_1, REVERSE, DINT_1_reverse);
SWPB(DINT_1, WORD, DINT_1_swap_word);
SWPB(DINT_1, HIGHLOW, DINT_1_swap_high_low);

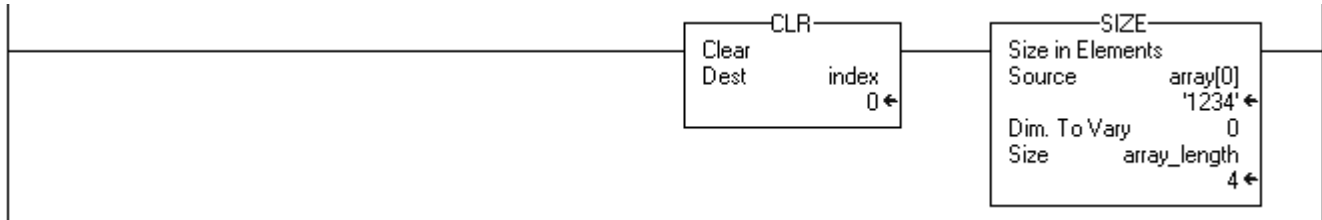
```

**Example 2:** The following example reverses the bytes in each element of an array. For an RSLogix 5000 project that contains this example, open the RSLogix 5000\Projects\Samples folder, Swap\_Bytes\_in\_Array.ACD file.

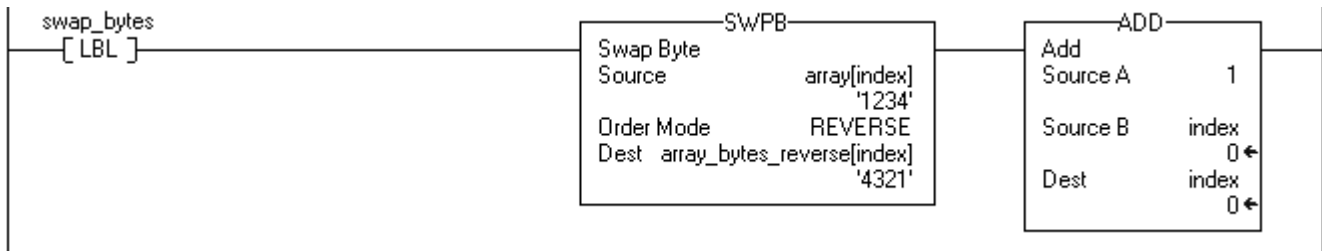
1. Initialize the tags. The SIZE instruction finds the number of elements in *array* and stores that value in *array\_length*. A subsequent instruction uses this value to determine when the routine has acted on all the elements in the array.
2. Reverse the bytes in one element of *array*.
  - The SWPB instruction reverses the bytes of the element number that is indicated by the value of *index*. For example, when *index* equals 0, the SWPB instruction acts on *array[0]*.
  - The ADD instruction increments *index*. The next time the instruction executes, the SWPB instruction acts on the next element in *array*.
3. Determine when the SWPB instruction has acted on all the elements in the array.
  - If *index* is less than the number of elements in the array (*array\_length*), then continue with the next element in the array.
  - If *index* equals *array\_length*, then the SWPB has acted on all the elements in the array.

## Relay Ladder

Initialize the tags.



Reverse the bytes.



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



## Structured Text

```

index := 0;
SIZE (array[0],0,array_length);
REPEAT
    SWPB(array[index],REVERSE,array_bytes_reverse[index]);
    index := index + 1;
UNTIL(index >= array_length)END_REPEAT;
  
```

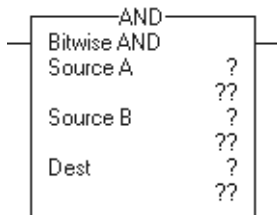


## Bitwise AND (AND)

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

To perform a logical AND, see page 319.

### Operands:



### Relay Ladder

Operand	Type	Format	Description
Source A	SINT	immediate	value to AND with Source B
	INT	tag	
<b>DINT</b>			
A SINT or INT tag converts to a DINT value by zero-fill.			
Source B	SINT	immediate	value to AND with Source A
	INT	tag	
<b>DINT</b>			
A SINT or INT tag converts to a DINT value by zero-fill.			
Destination	SINT	tag	stores the result
	INT		
<b>DINT</b>			

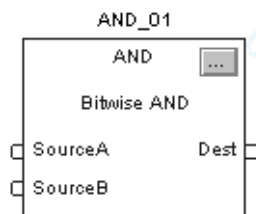


```
dest := sourceA AND sourceB
```

### Structured Text

Use AND or the ampersand sign “&” as an operator within an expression. This expression evaluates *sourceA* AND *sourceB*.

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



### Function Block

Operand	Type	Format	Description
AND tag	FBD_LOGICAL	structure	AND structure

**FBD\_LOGICAL Structure**

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

**Description:** When enabled, the instruction evaluates the AND operation:

If the Bit In Source A Is	And the Bit In Source B Is:	The Bit In the Destination Is
0	0	0
0	1	0
1	0	0
1	1	1

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

**Arithmetic Status Flags:** Arithmetic status flags are affected.

**Fault Conditions:** none

**Execution:**

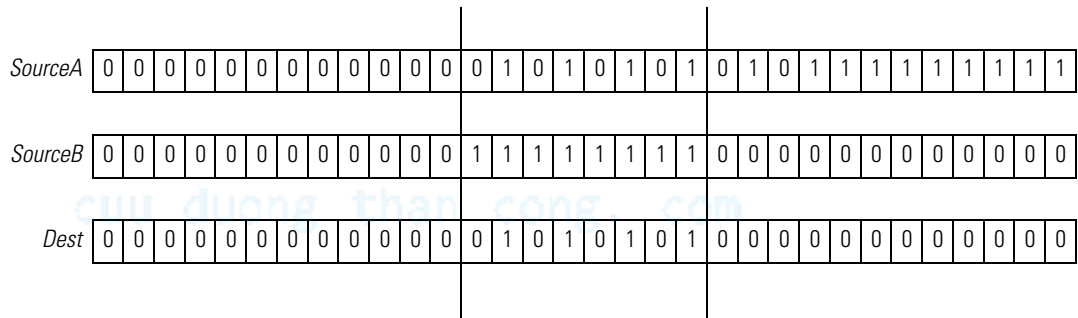
**Relay Ladder**

Condition	Action
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	The instruction performs a bitwise AND operation. The rung-condition-out is set to true.
postscan	The rung-condition-out is set to false.

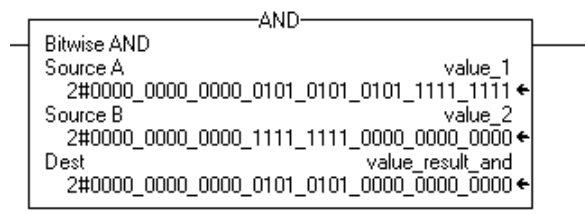
## Function Block

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

**Example:** When enabled, the AND instruction performs a bitwise AND operation on SourceA and SourceB and places the result in the Dest.



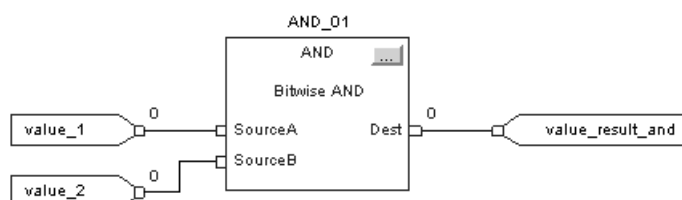
## Relay Ladder



## Structured Text

```
value_result_and := value_1 AND value_2;
```

## Function Block

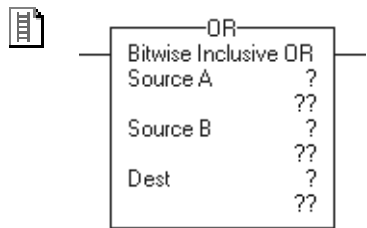


# Bitwise OR (OR)

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

To perform a logical OR, see page 13-322.

## Operands:



## Relay Ladder

Operand	Type	Format	Description
Source A	SINT	immediate	value to OR with Source B
	INT	tag	
<b>DINT</b>			
A SINT or INT tag converts to a DINT value by zero-fill.			
Source B	SINT	immediate	value to OR with Source A
	INT	tag	
<b>DINT</b>			
A SINT or INT tag converts to a DINT value by zero-fill.			
Destination	SINT	tag	stores the result
	INT		
<b>DINT</b>			



dest := sourceA OR sourceB

## Structured Text

Use OR as an operator within an expression. This expression evaluates *sourceA* OR *sourceB*.

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



## Function Block

Operand	Type	Format:	Description
OR tag	FBD_LOGICAL	structure	OR structure

**FBD\_LOGICAL Structure**

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

**Description:** When enabled, the instruction evaluates the OR operation:

If the Bit In Source A Is	And the Bit In Source B Is	The Bit In the Destination Is
0	0	0
0	1	1
1	0	1
1	1	1

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

**Arithmetic Status Flags** Arithmetic status flags are affected.

**Fault Conditions:** none

**Execution:**

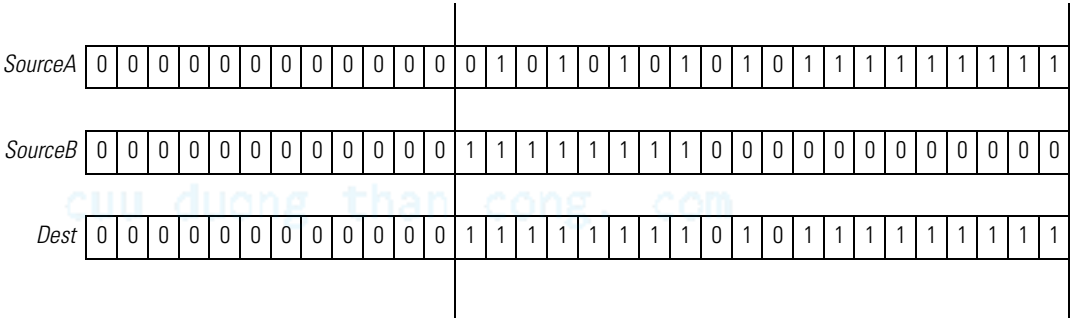
**Relay Ladder**

Condition	Action
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	The instruction performs a bitwise OR operation. The rung-condition-out is set to true.
postscan	The rung-condition-out is set to false.

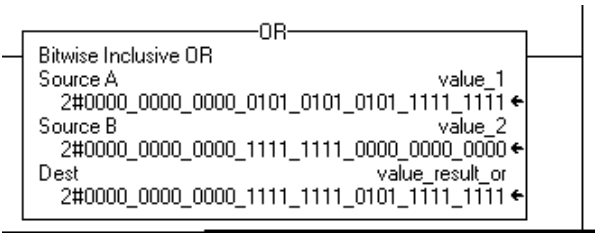
Function Block

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

**Example:** When enabled, the OR instruction performs a bitwise OR operation on SourceA and SourceB and places the result in Dest.



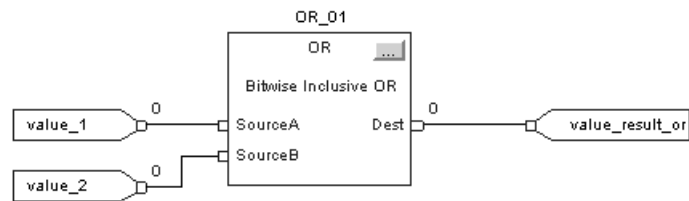
Relay Ladder



Structured Text

value\_result\_or := value\_1 OR value\_2;

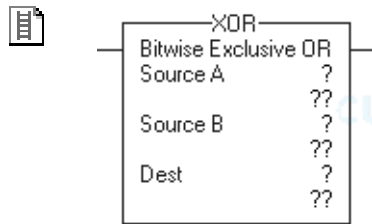
Function Block



**Bitwise Exclusive OR (XOR)** The XOR instruction performs a bitwise XOR operation using the bits in Source A and Source B and places the result in the Destination.

To perform a logical XOR, see page 13-325.

Operands:



Relay Ladder

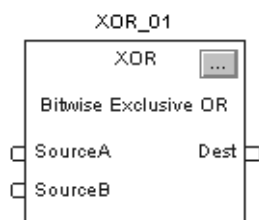
Operand	Type	Format	Description
Source A	SINT	immediate	value to XOR with Source B
	INT	tag	
	DINT		A SINT or INT tag converts to a DINT value by zero-fill.
Source B	SINT	immediate	value to XOR with Source A
	INT	tag	
	DINT		A SINT or INT tag converts to a DINT value by zero-fill.
Destination	SINT	tag	stores the result
	INT		
	DINT		

```
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.

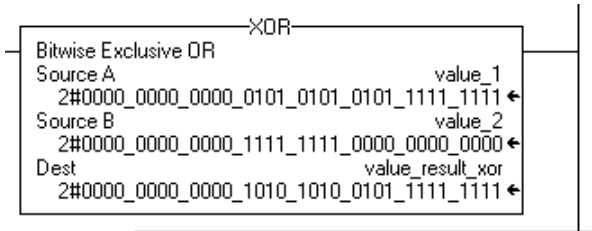
**Function Block**

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.

value_1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1
value_2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
value_result_xor	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	1	0	0	1	0	1	1	1	1	1	1	1	1	1

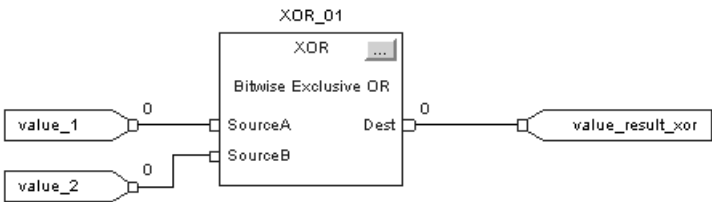
Relay Ladder



Structured Text

value\_result\_xor := value\_1 XOR value\_2;

Function Block

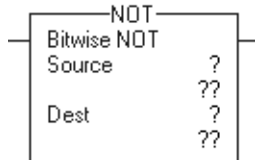


## Bitwise NOT (NOT)

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

To perform a logical NOT, see page 13-328.

### Operands:



### Relay Ladder

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



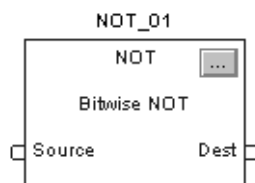
dest := NOT source

### 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:**

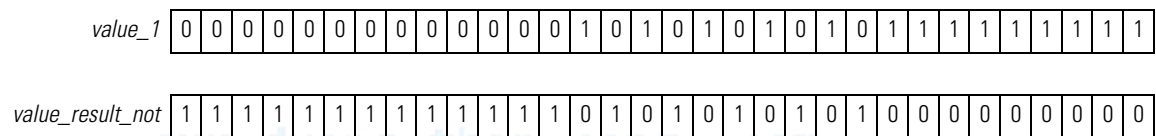
**Relay Ladder**

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.

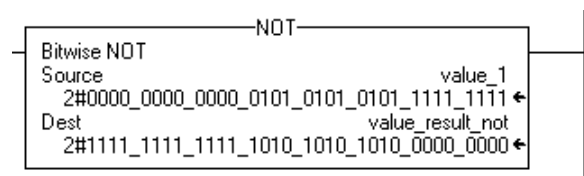
## Function Block

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.



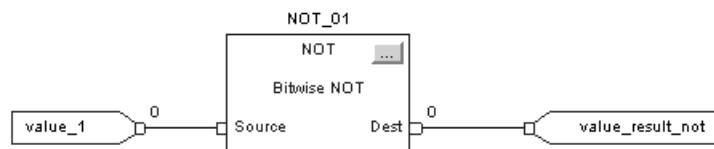
## Relay Ladder



## Structured Text

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

## Function Block



cuu duong than cong. com

cuu duong than cong. com

## Boolean AND (BAND)

The BAND instruction logically ANDs as many as 8 boolean inputs.

To perform a bitwise AND, see page 13-305.

### Operands:

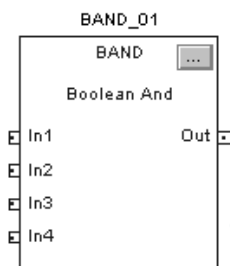


```
IF operandA AND operandB THEN
    <statement>;
END_IF;
```

### Structured Text

Use AND or the ampersand sign “&” as an operator within an expression. The operands must be BOOL values or expressions that evaluate to BOOL values. This expression evaluates whether *operandA* and *operandB* are both set (true).

See 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.

Input Parameter	Data Type	Description
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:** [cuuduongthancong.com](http://cuuduongthancong.com)

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.

[cuuduongthancong.com](http://cuuduongthancong.com)



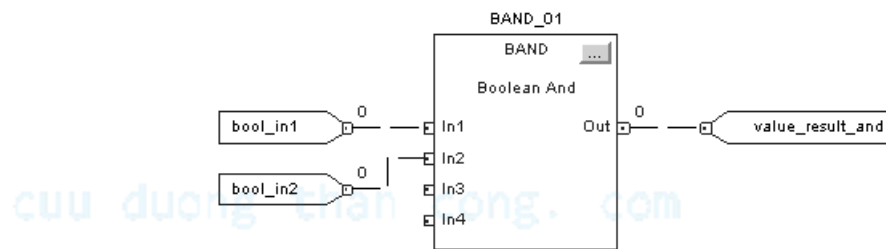
**Example 1:** This example ANDs *bool\_in1* and *bool\_in2* and places the result in *value\_result\_and*.

If BOOL_IN1 Is	If BOOL_IN2 Is	Then VALUE_RESULT_AND Is
0	0	0
0	1	0
1	0	0
1	1	1

### Structured Text

```
value_result_and := bool_in1 AND bool_in2;
```

### Function Block



**Example 2:** If both *bool\_in1* and *bool\_in2* are set (true), *light1* is set (on). Otherwise, *light1* is cleared (off).

### Structured Text

```
IF bool_in1 AND bool_in2 THEN
    light1 := 1;
ELSE
    light1 := 0;
END_IF;
```

# Boolean OR (BOR)

The BOR instruction logically ORs as many as eight boolean inputs.

To perform a bitwise OR, see page 13-308.

## Operands:

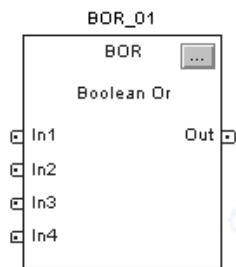


```
IF operandA OR operandB THEN
    <statement>;
END_IF;
```

## Structured Text

Use OR as an operator within an expression. The operands must be BOOL values or expressions that evaluate to BOOL values. This expression evaluates whether *operandA* or *operandB* or both are set (true).

See 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.

Input Parameter	Data Type	Description
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).

$Out = In1 \text{ OR } In2 \text{ OR } In3 \text{ OR } In4 \text{ OR } In5 \text{ OR } In6 \text{ OR } In7 \text{ OR } In8$

**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.

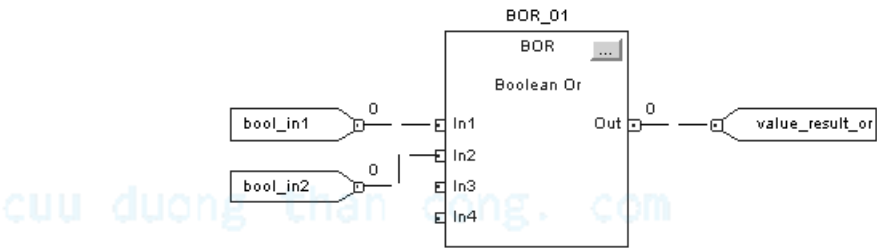
**Example 1:** This example ORs *bool\_in1* and *bool\_in2* and places the result in *value\_result\_or*.

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:

- 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;
```

cuu duong than cong. com

## Boolean Exclusive OR (BXOR)

The BXOR performs an exclusive OR on two boolean inputs.

To perform a bitwise XOR, see page 13-311.

### Operands:

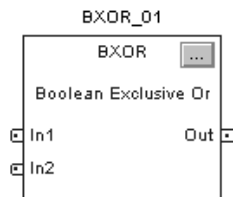


```
IF operandA XOR operandB THEN
    <statement>;
END_IF;
```

### Structured Text

Use XOR as an operator within an expression. The operands must be BOOL values or expressions that evaluate to BOOL values. This expression evaluates whether only *operandA* or only *operandB* is set (true).

See 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.

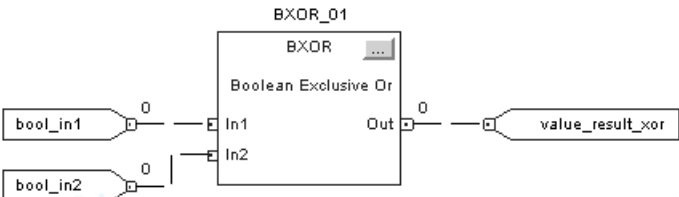
**Example 1:** This example performs an exclusive OR on *bool\_in1* and *bool\_in2* and places the result in *value\_result\_xor*.

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;
```

cuu duong than cong. com

cuu duong than cong. com

## Boolean NOT (BNOT)

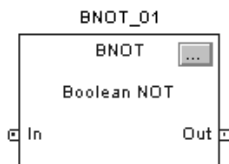
The BNOT instruction complements a boolean input.

To perform a bitwise NOT, see page 13-315.

### Operands:



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



### Structured Text

Use NOT as an operator within an expression. The operand must be a BOOL values or expressions that evaluate to BOOL values. This expression evaluates whether *operand* is cleared (false).

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

### Function Block

Operand	Type	Format	Description
BNOT tag	FBD_BOOLEAN_NOT	structure	BNOT structure

### FBD\_BOOLEAN\_NOT Structure

Input Parameter	Data Type	Description
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set.
In	BOOL	Input to the instruction. Default is set.
Output Parameter	Data Type	Description:
EnableOut	BOOL	Enable output.
Out	BOOL	The output of the instruction.

**Description:** The BNOT instruction complements a boolean input.

Out = NOT In

**Arithmetic Status Flags:** not affected

**Fault Conditions:** none



**Execution:**

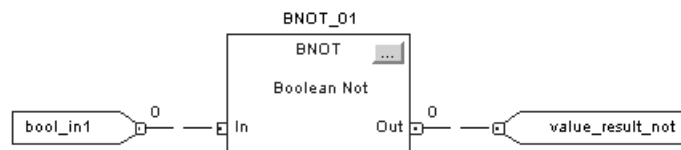
Condition	Function Block Action
prescan	No action taken.
instruction first scan	No action taken.
instruction first run	No action taken.
EnableIn is cleared	EnableOut is cleared.
EnableIn is set	The instruction executes. EnableOut is set.
postscan	No action taken.

**Example 1:** This example complements *bool\_in1* and places the result in *value\_result\_not*.

If 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:

cuu duong than cong. com

cuu duong than cong. com

## Array (File)/Misc. Instructions

### (FAL, FSC, COP, CPS, FLL, AVE, SRT, STD, SIZE)

### Introduction

The file/miscellaneous instructions operate on arrays of data.

If You Want To	Use This Instruction	Available In These Languages	See Page
perform arithmetic, logic, shift, and function operations on values in arrays	FAL	relay ladder structured text <sup>(1)</sup>	337
search for and compare values in arrays	FSC	relay ladder	349
copy the contents of one array into another array	COP	relay ladder structured text	358
copy the contents of one array into another array without interruption	CPS	relay ladder structured text	358
fill an array with specific data	FLL	relay ladder structured text <sup>(1)</sup>	364
calculate the average of an array of values	AVE	relay ladder structured text <sup>(1)</sup>	368
sort one dimension of array data into ascending order	SRT	relay ladder structured text	373
calculate the standard deviation of an array of values	STD	relay ladder structured text <sup>(1)</sup>	378
find the size of a dimension of an array	SIZE	relay ladder structured text	384

<sup>(1)</sup> 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.

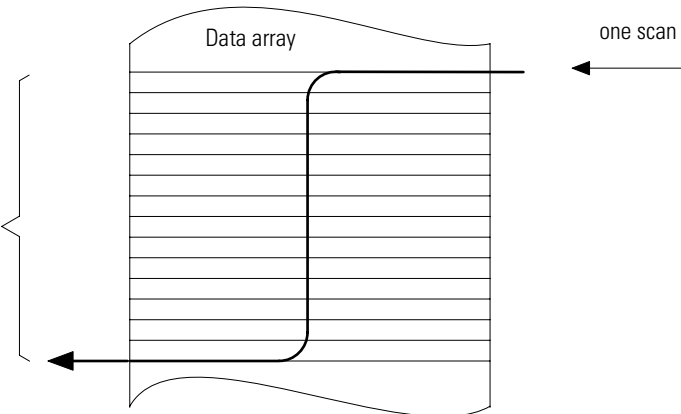
# Selecting Mode of Operation

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	Numerical
enter the number of elements to operate on per scan (1-2147483647)	
manipulate one element of the array each time the rung-condition-in goes from false to true	Incremental

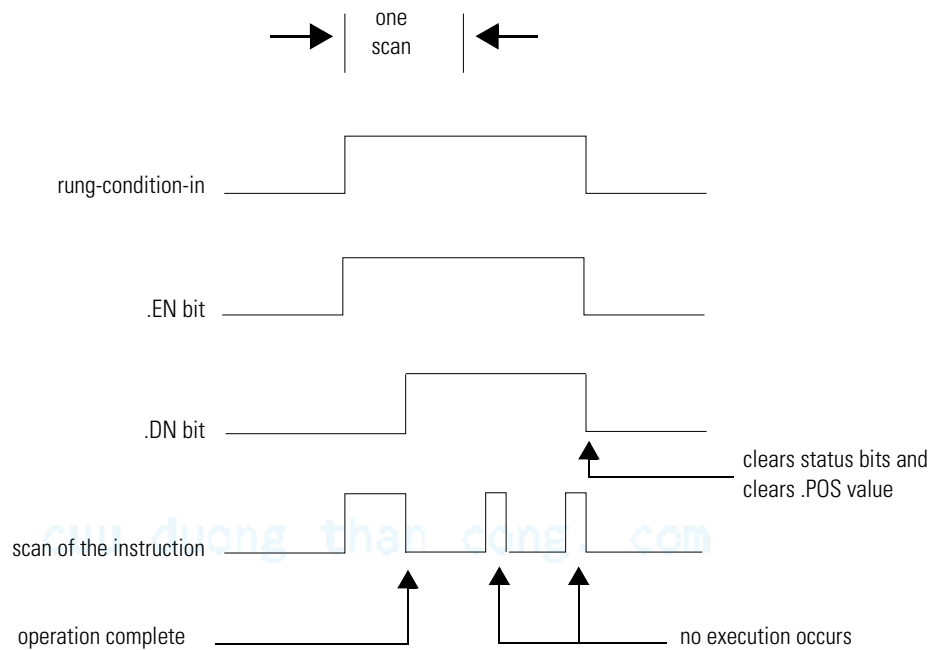
## All mode

In All mode, all the specified elements in the array are operated on before continuing on to the next instruction. The operation begins when the instruction's rung-condition-in goes from false to true. The position (.POS) value in the control structure points to the element in the array that the instruction is currently using. Operation stops when the .POS value equals the .LEN value.



16639

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

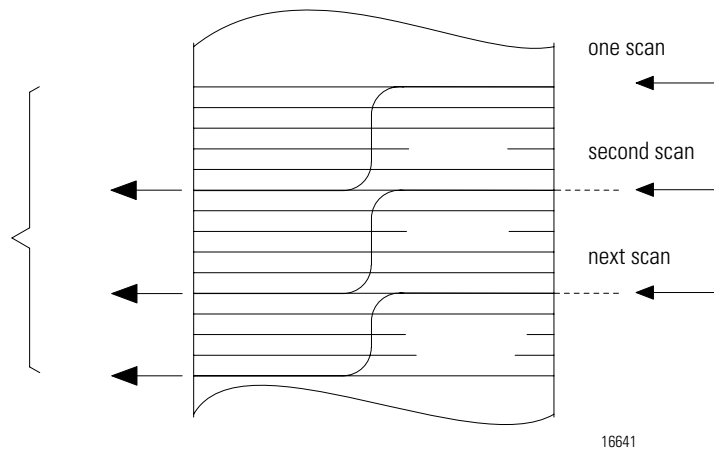


40010

## Numerical mode

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

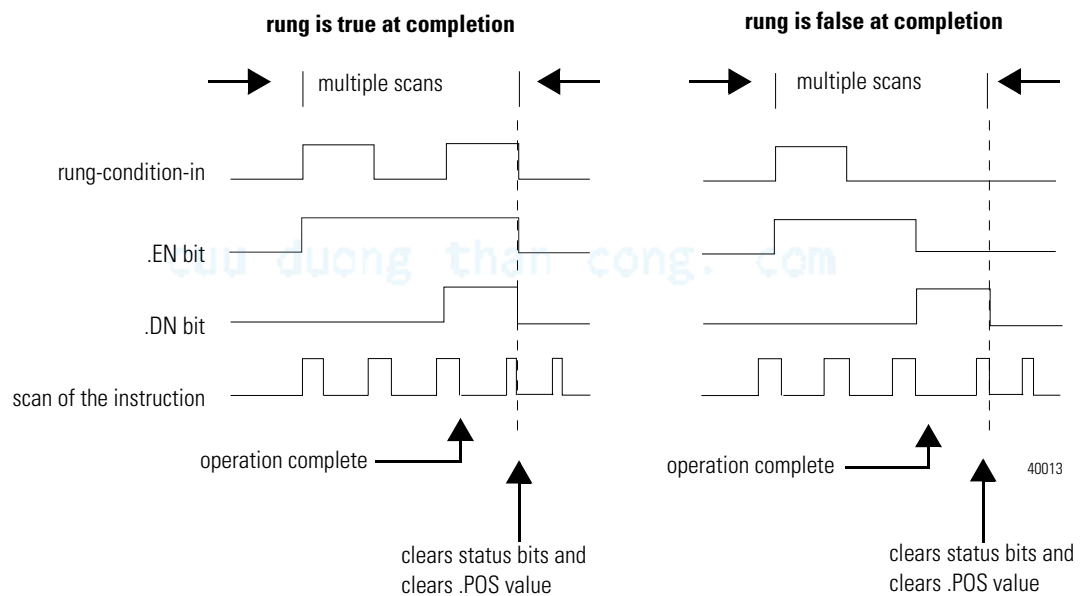
Execution is triggered when the rung-condition-in goes from false to true. Once triggered, the instruction is executed each time it is scanned for the number of scans necessary to complete operating on the entire array. Once triggered, rung-condition-in can change repeatedly without interrupting execution of the instruction.



### IMPORTANT

Avoid using the results of a file instruction operating in numerical mode until the .DN bit is set.

The following timing diagram shows the relationship between status bits and instruction operation. When the instruction execution is complete, the .DN bit is set.

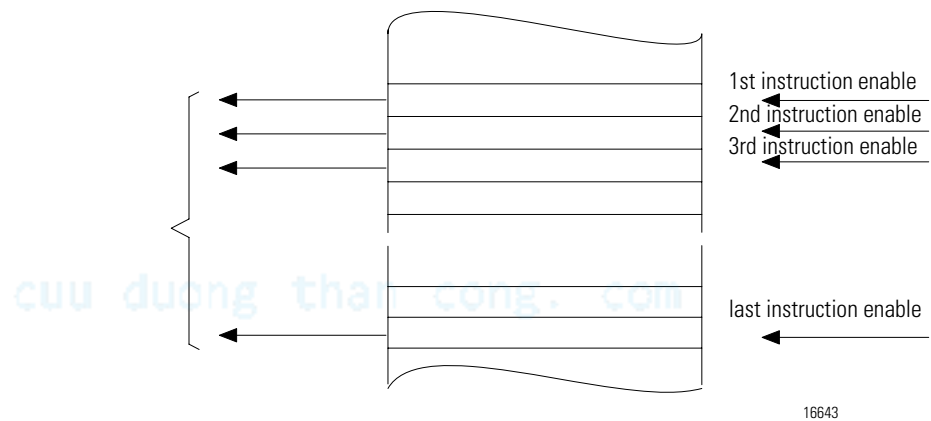


If the rung-condition-in is true at completion, the .EN and .DN bit are set until the rung-condition-in goes false. When the rung-condition-in goes false, these bits are cleared and the .POS value is cleared.

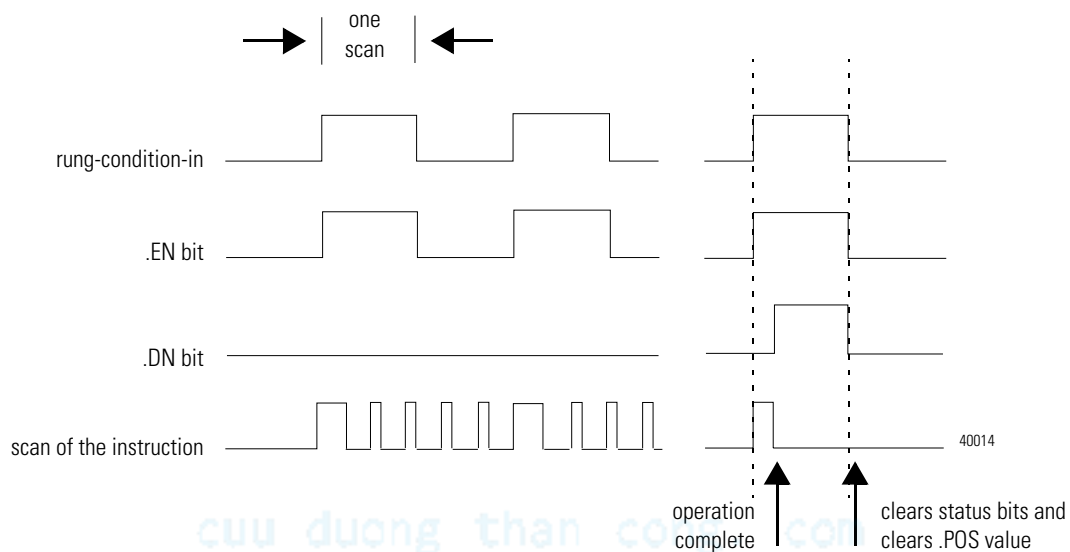
If the rung-condition-in is false at completion, the .EN bit is cleared immediately. One scan after the .EN bit is cleared, the .DN bit and the .POS value are cleared.

## Incremental mode

Incremental mode manipulates one element of the array each time the instruction's rung-condition-in goes from false to true.



The following timing diagram shows the relationship between status bits and instruction operation. Execution occurs only in a scan in which the rung-condition-in goes from false to true. Each time this occurs, only one element of the array is manipulated. If the rung-condition-in remains true for more than one scan, the instruction only executes during the first scan.



The .EN bit is set when rung-condition-in is true. The .DN bit is set when the last element in the array has been manipulated. When the last element has been manipulated and the rung-condition-in goes false, the .EN bit, the .DN bit, and the .POS value are cleared.

The difference between incremental mode and numerical mode at a rate of one element per scan is:

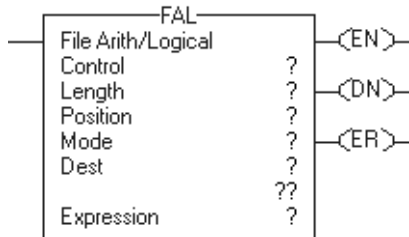
- Numerical mode with any number of elements per scan requires only one false-to-true transition of the rung-condition-in to start execution. The instruction continues to execute the specified number of elements each scan until completion regardless of the state of the rung-condition-in.
- Incremental mode requires the rung-condition-in to change from false to true to manipulate one element in the array.



## File Arithmetic and Logic (FAL)

The FAL instruction performs copy, arithmetic, logic, and function operations on data stored in an array.

### Operands:



### Relay Ladder

Operand	Type	Format	Description
Control	CONTROL	tag	control structure for the operation
Length	DINT	immediate	number of elements in the array to be manipulated
Position	DINT	immediate	current element in array initial value is typically 0
Mode	DINT	immediate	how to distribute the operation select INC, ALL, or enter a number
Destination	SINT	tag	tag to store the result
	INT		
	<b>DINT</b>		
	<b>REAL</b>		
Expression	SINT	immediate	an expression consisting of tags and/or immediate values separated by operators
	INT	tag	
	<b>DINT</b>		
	<b>REAL</b>		
A SINT or INT tag converts to a DINT value by sign-extension.			



### Structured Text

Structured text does not have an FAL instruction, but you can achieve the same results using a SIZE instruction and a FOR...DO or other loop construct.

```
SIZE(destination,0,length-1);
FOR position = 0 TO length DO
    destination[position] := numeric_expression;
END_FOR;
```

See Appendix B 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.

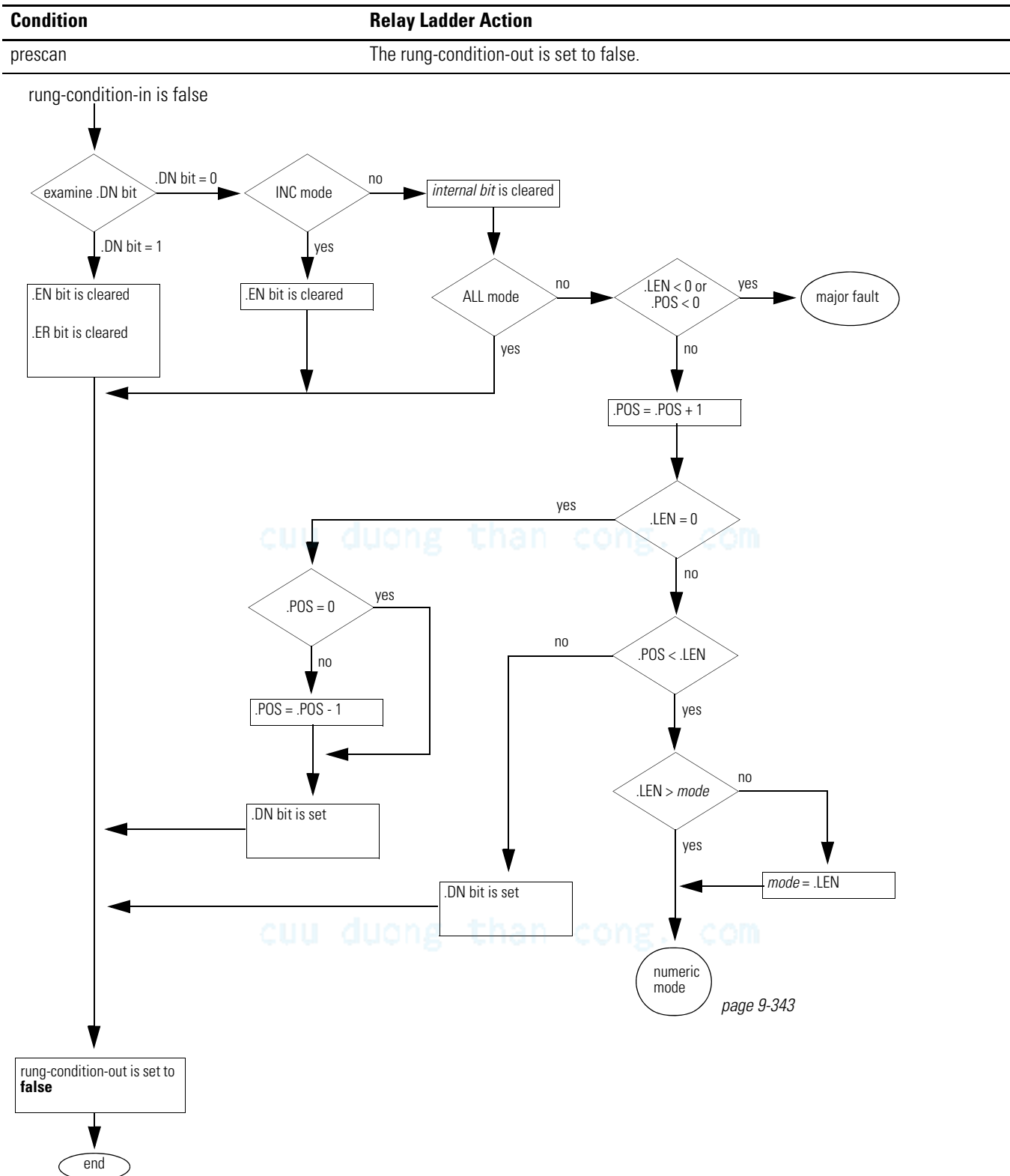
The examples that start on page 9-344 show how to use the .POS value to step through an array. If a subscript in the expression of the Destination is out of range, the FAL instruction generates a major fault (type 4, code 20).

**Arithmetic Status Flags:** Arithmetic status flags are affected.

**Fault Conditions:**

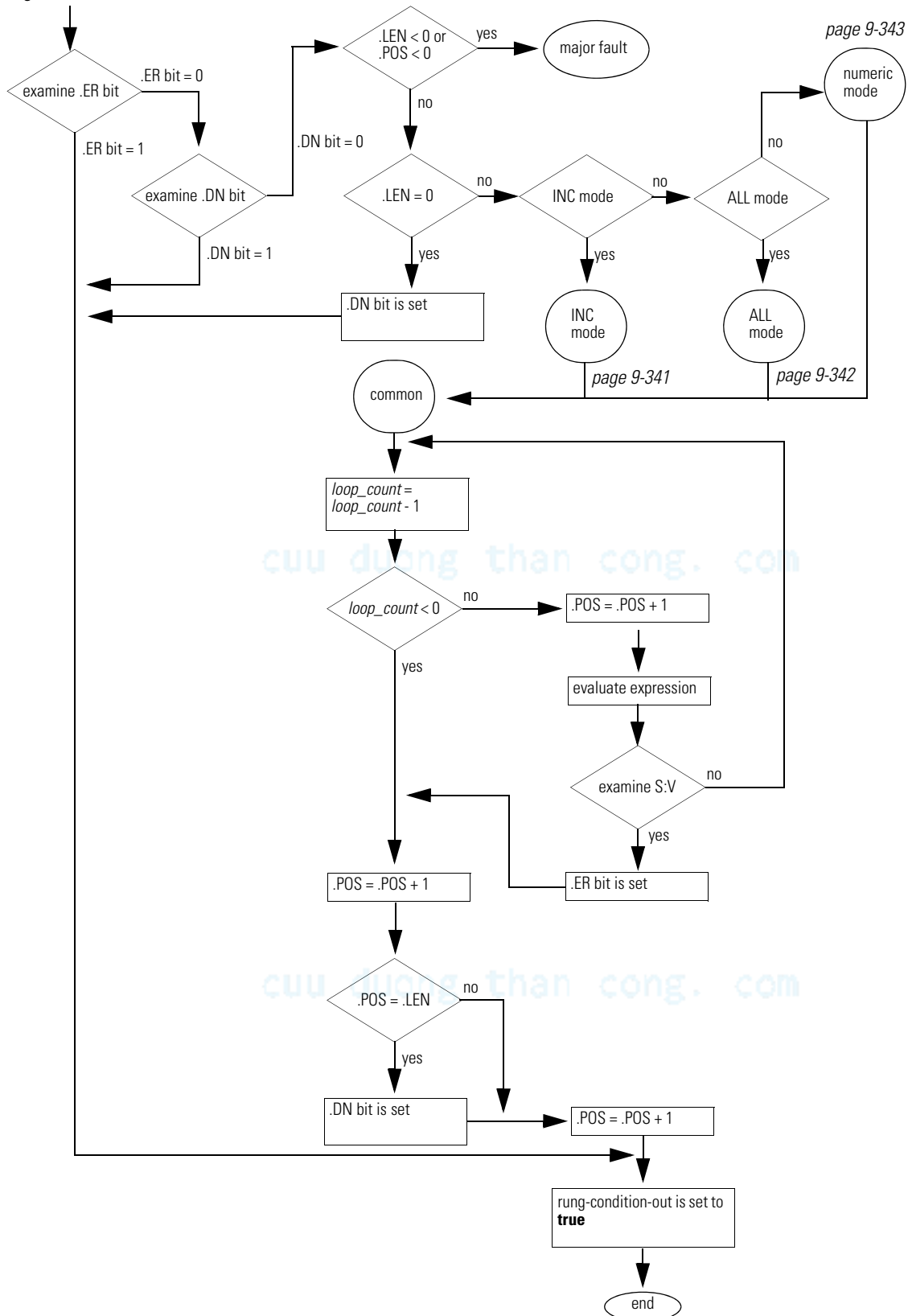
A Major Fault Will Occur If	Fault Type	Fault Code
subscript is out of range	4	20
.POS < 0 or .LEN < 0	4	21

## Execution:



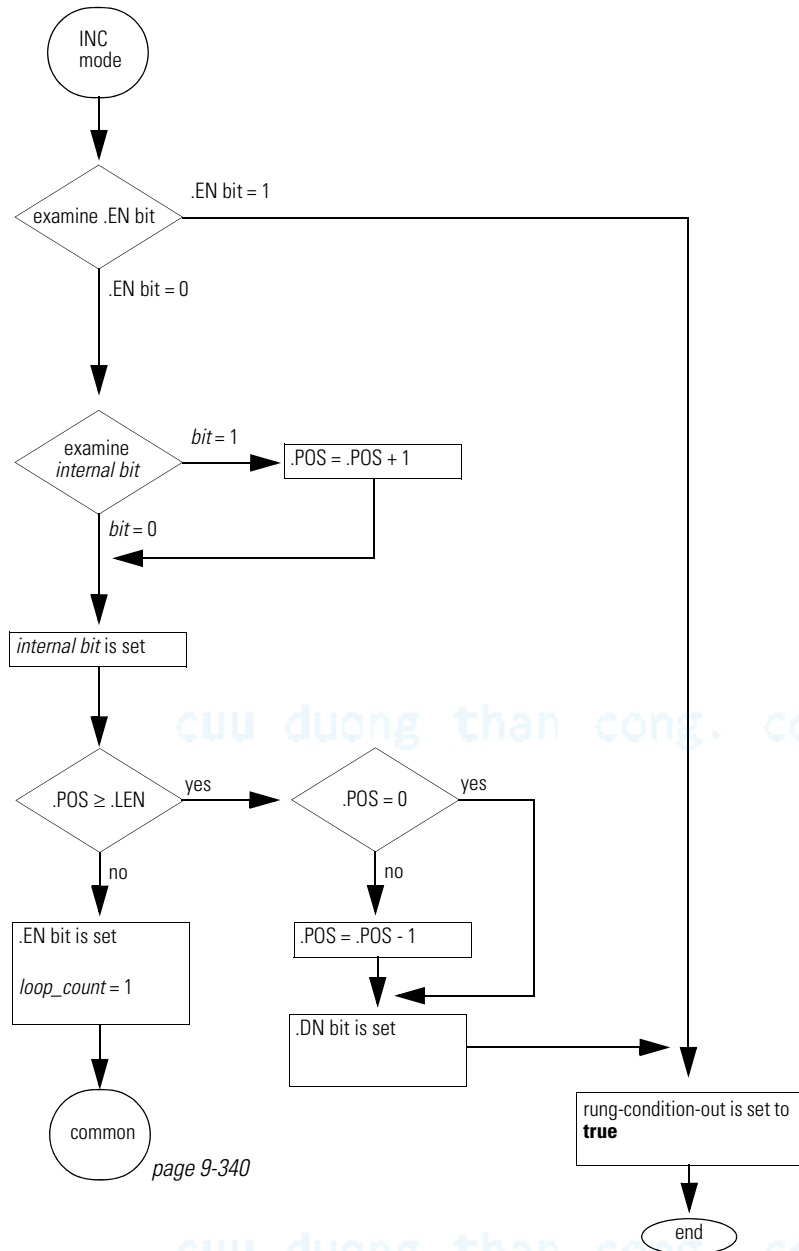
**Condition****Relay Ladder Action**

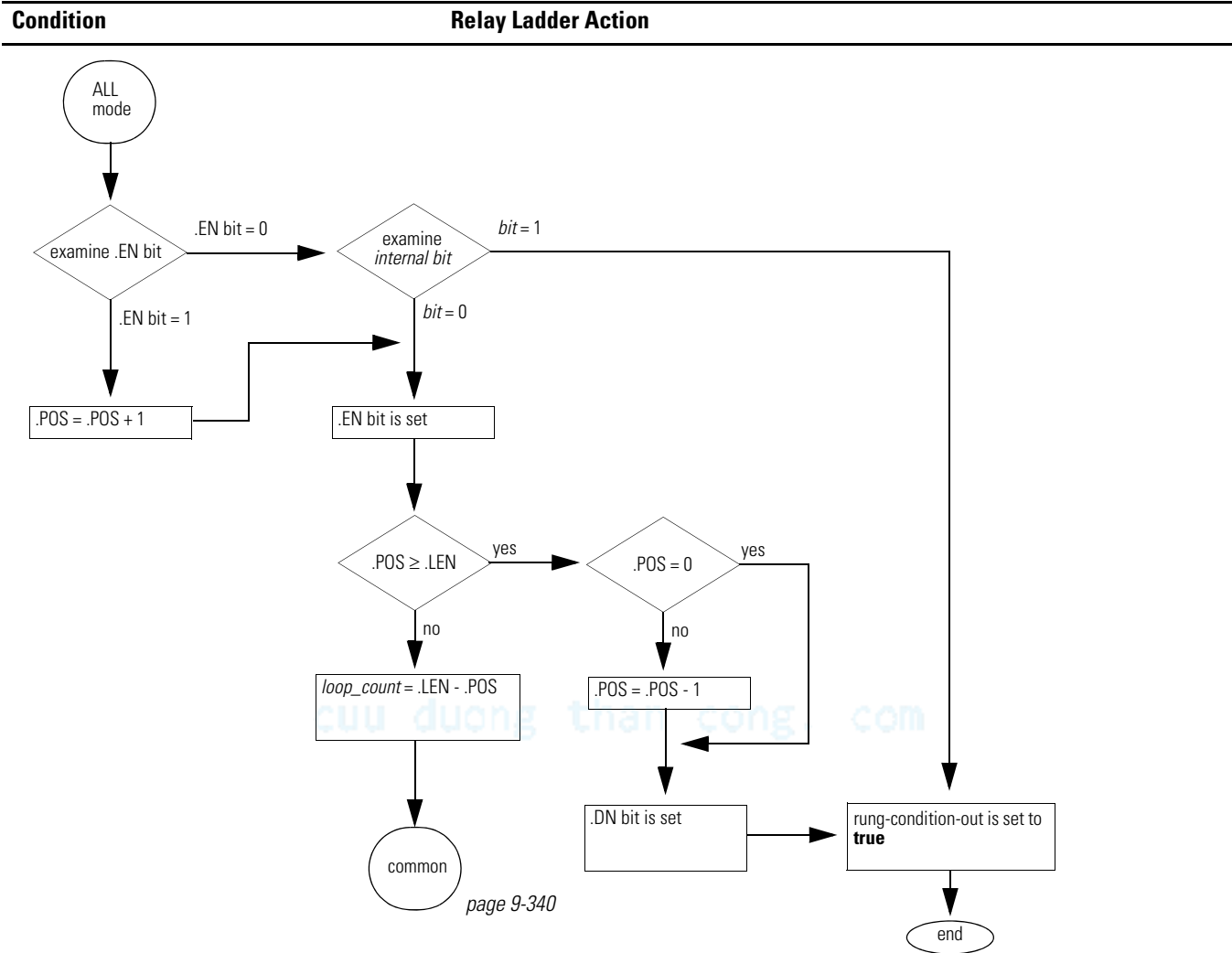
rung-condition-in is true

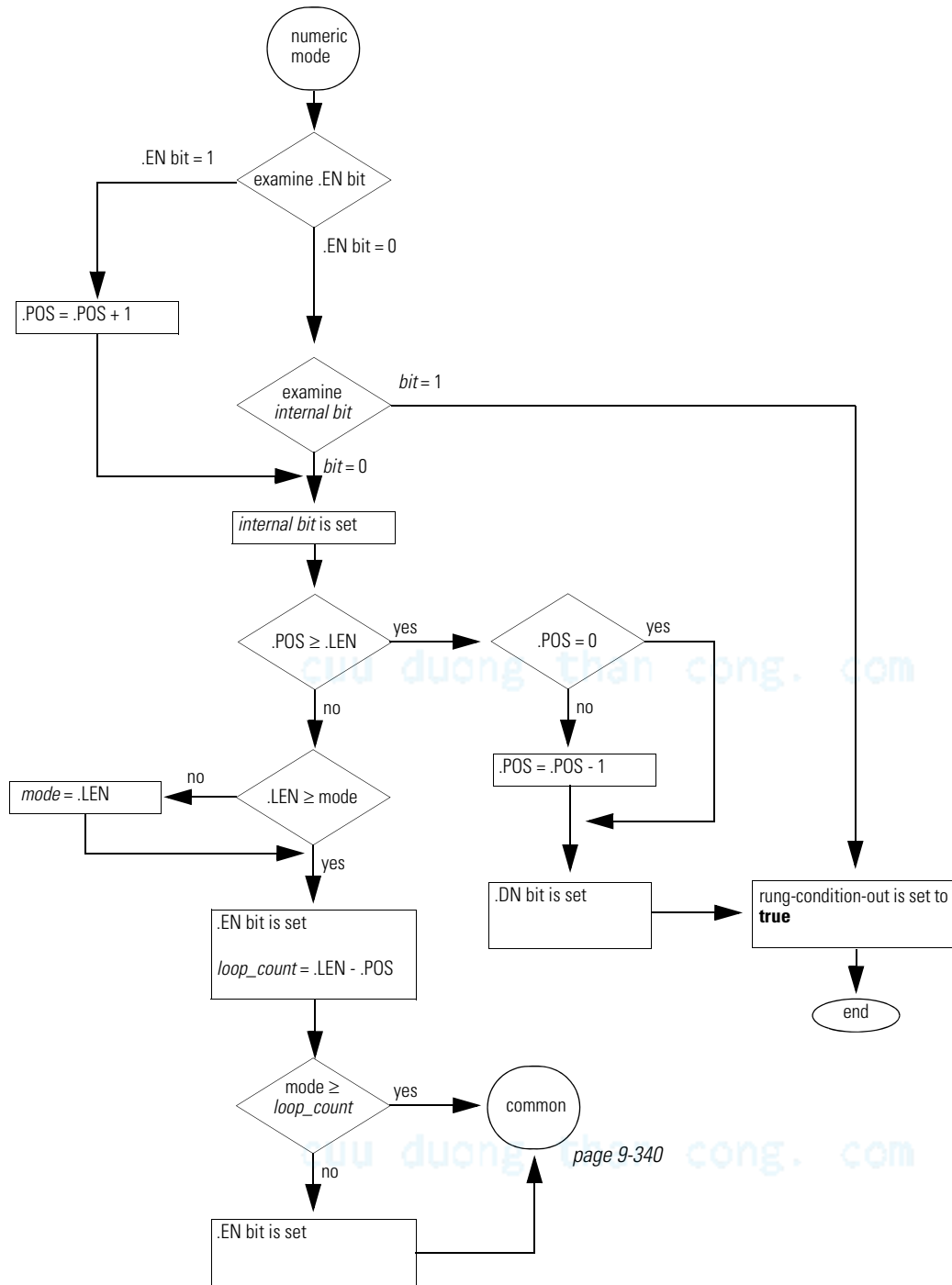


## Condition

## Relay Ladder Action



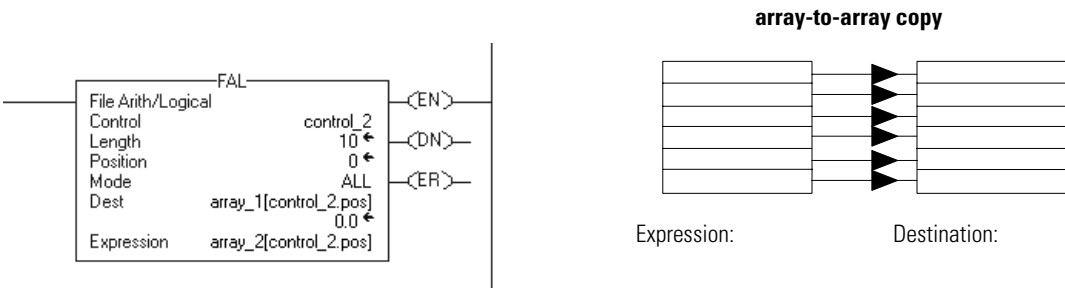


**Condition**
**Relay Ladder Action**


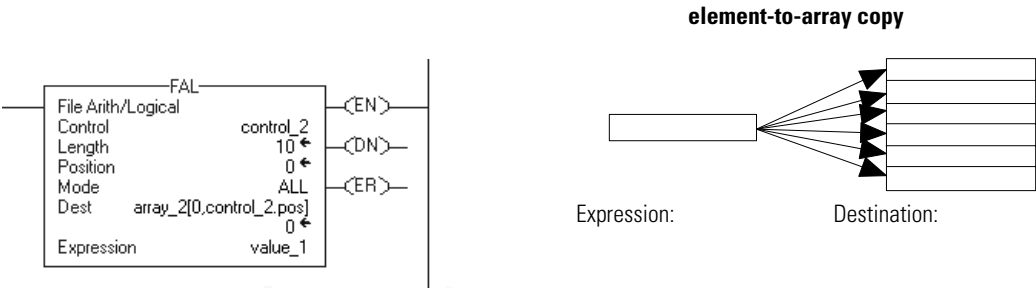
postscan

The rung-condition-out is set to false.

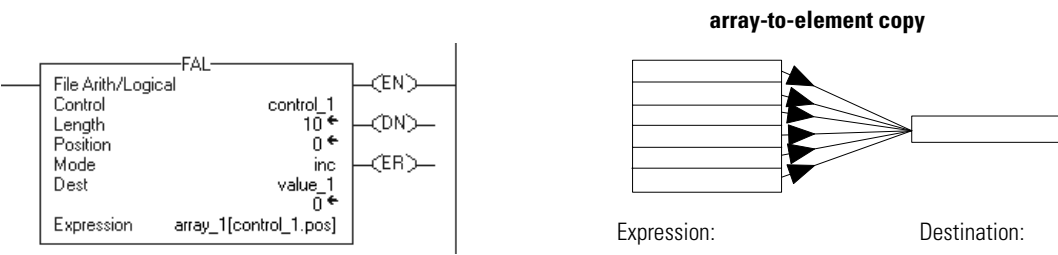
**Example 1:** When enabled, the FAL instruction copies each element of *array\_2* into the same position within *array\_1*.



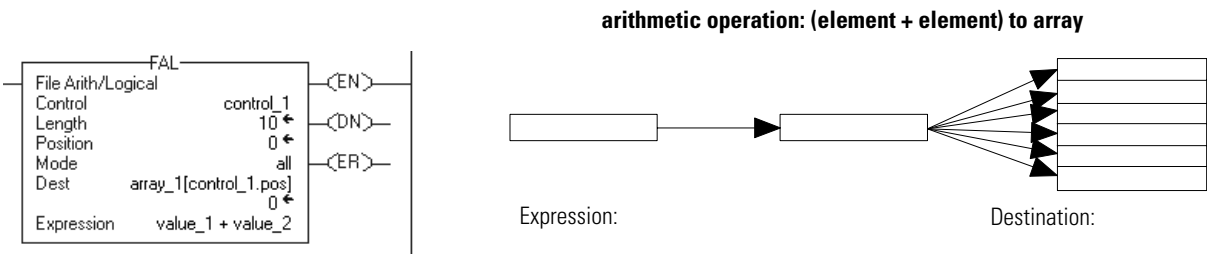
**Example 2:** When enabled, the FAL instruction copies *value\_1* into the first 10 positions of the second dimension of *array\_2*.



**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*.

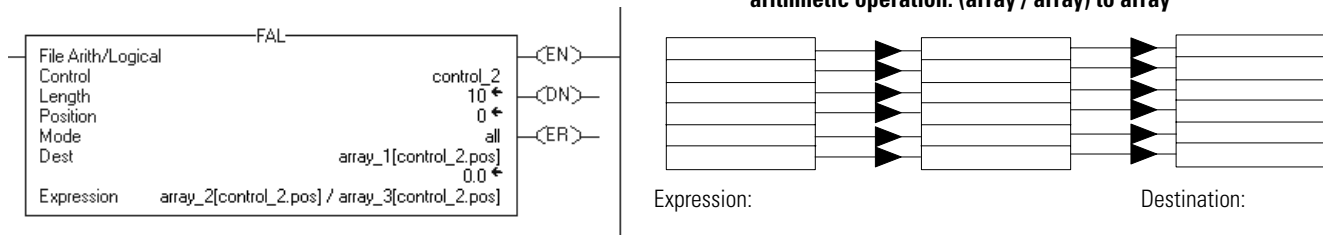


**Example 4:** When enabled, the FAL instruction adds *value\_1* and *value\_2* and stores the result in the current position of *array\_1*.

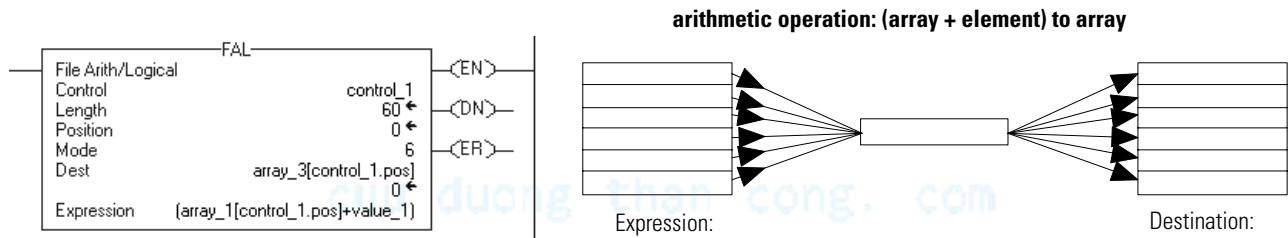




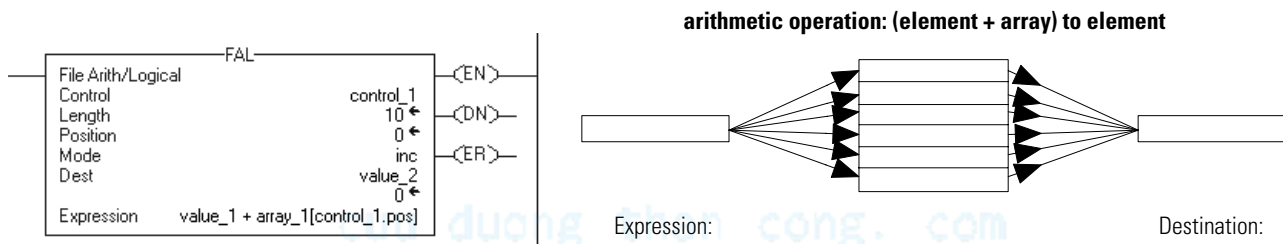
**Example 5:** When enabled, the FAL instruction divides the value in the current position of *array\_2* with the value in the current position of *array\_3* and stores the result in the current position of *array\_1*.



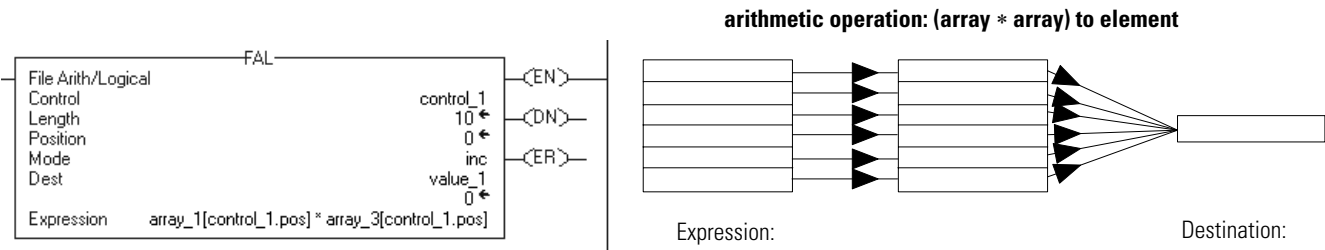
**Example 6:** When enabled, the FAL instruction adds the value at the current position in *array\_1* to *value\_1* and stores the result in the current position in *array\_3*. The instruction must execute 10 times for the entire *array\_1* and *array\_3* to be manipulated.



**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

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

## 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( <i>tag_a</i> )
two operands	operand_a operator operand_b	<ul style="list-style-type: none"> <li><i>tag_b</i> + 5</li> <li><i>tag_c</i> AND <i>tag_d</i></li> <li>(<i>tag_e</i> ** 2) MOD (<i>tag_f</i> / <i>tag_g</i>)</li> </ul>

## 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.

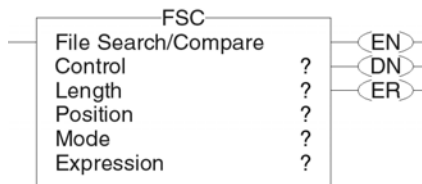
Operations of equal order are performed from left to right.

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.

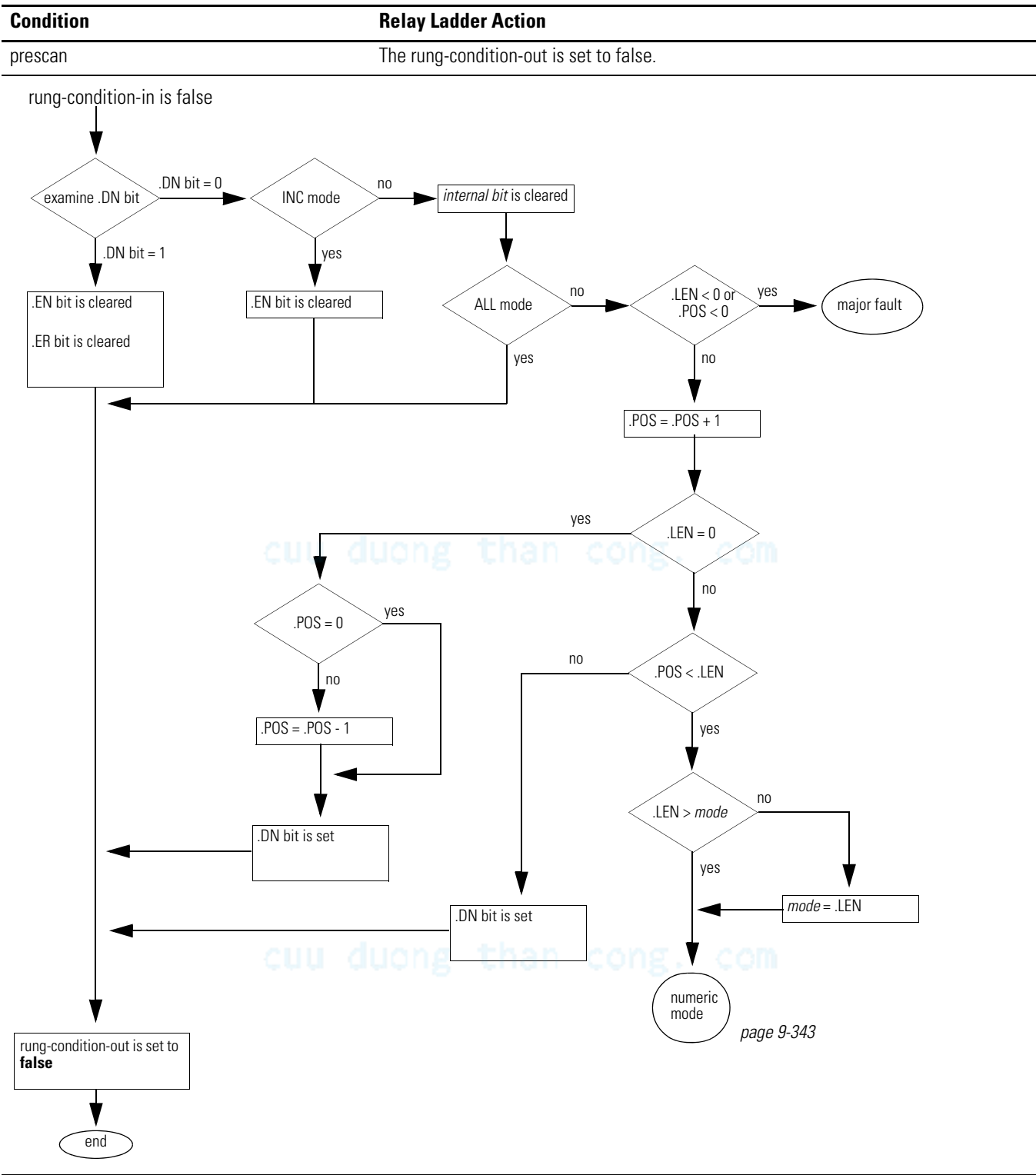
**Description:** When the FSC instruction is enabled and the comparison is true, the instruction sets the .FD bit and the .POS bit reflects the array position where the instruction found the true comparison. The instruction sets the .IN bit to prevent further searching.

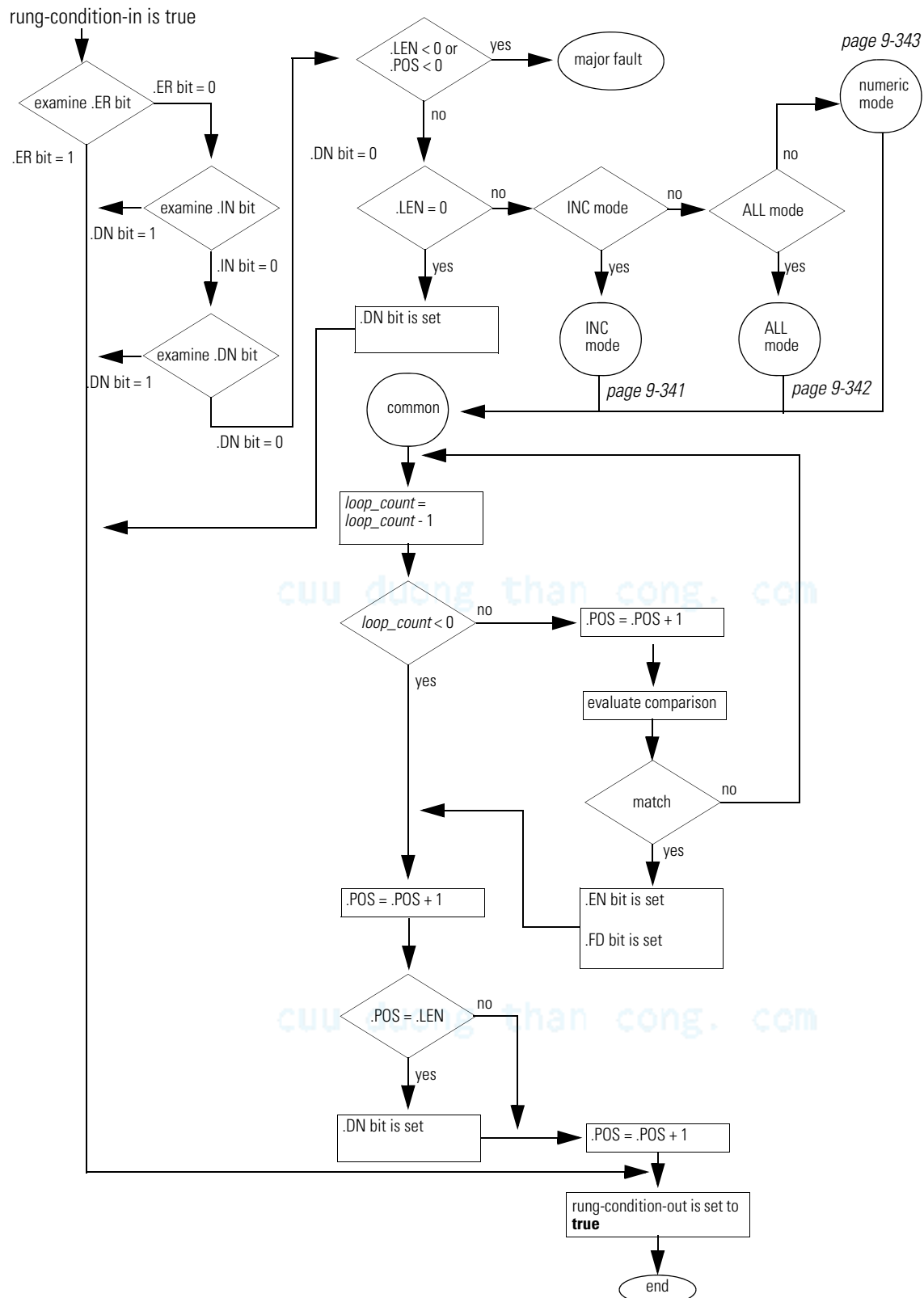
**Arithmetic Status Flags:** Arithmetic status flags are affected.

### Fault Conditions:

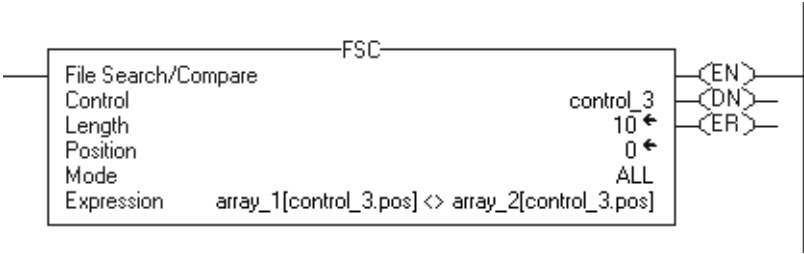
A Major Fault Will Occur If	Fault Type	Fault Code
.POS < 0 or .LEN < 0	4	21

**Execution:**



**Condition****Relay Ladder Action**

**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*.

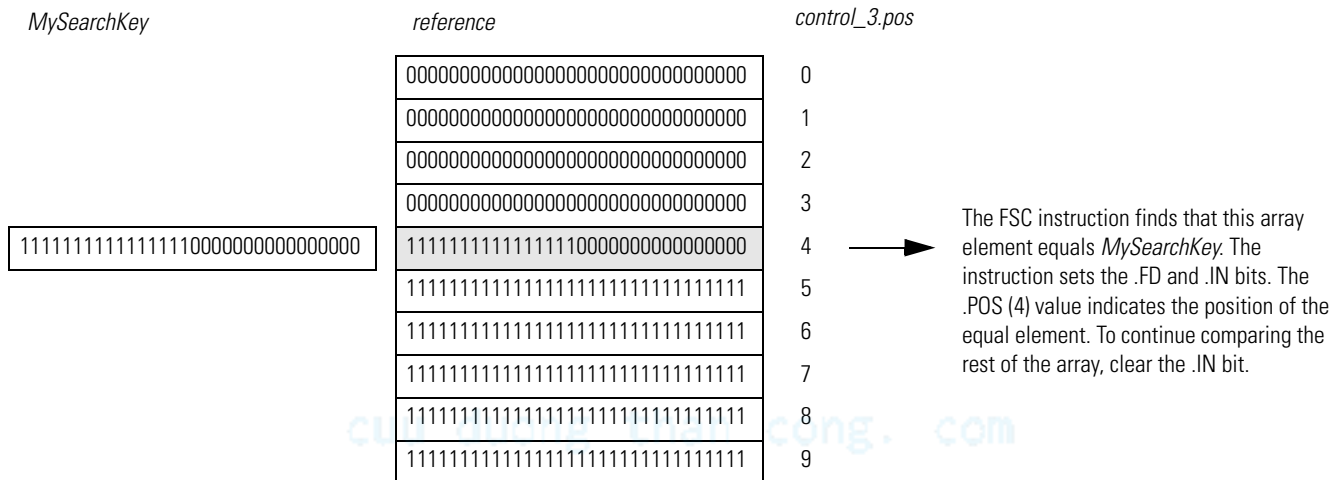
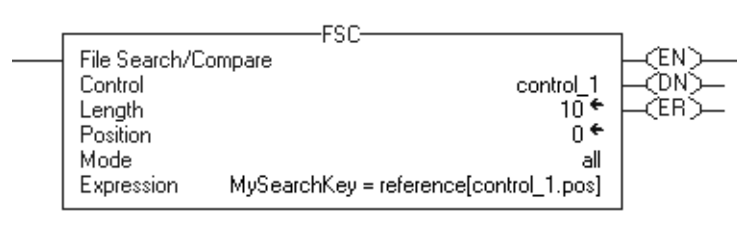


array_1	array_2	control_3.pos
00000000000000000000000000000000	00000000000000000000000000000000	0
00000000000000000000000000000000	00000000000000000000000000000000	1
00000000000000000000000000000000	00000000000000000000000000000000	2
00000000000000000000000000000000	00000000000000000000000000000000	3
00000000000000000111111111111111	11111111111111110000000000000000	4
11111111111111111111111111111111	11111111111111111111111111111111	5
11111111111111111111111111111111	11111111111111111111111111111111	6
11111111111111111111111111111111	11111111111111111111111111111111	7
11111111111111111111111111111111	11111111111111111111111111111111	8
11111111111111111111111111111111	11111111111111111111111111111111	9

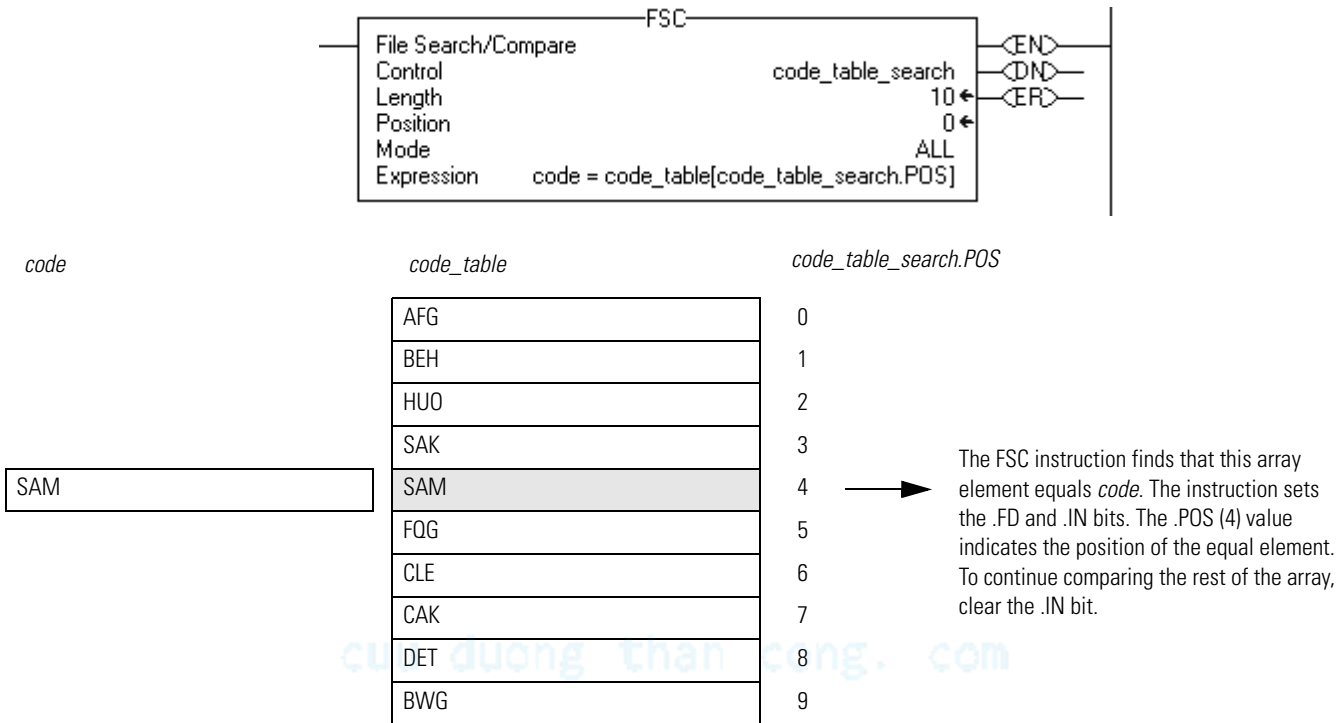
The FSC instruction finds that these elements are not equal. The instruction sets the .FD and .IN bits. The .POS value (4) indicates the position of the elements that are not equal. To continue comparing the rest of the array, clear the .IN bit.



**Example 2:** Search for a match in an array. When enabled, the FSC instruction compares the *MySearchKey* to 10 elements in *array\_1*.



**Example 3:** Search for a string in an array of strings. When enabled, the FSC instruction compares the characters in *code* to 10 elements in *code\_table*.



### FSC expressions

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

## Valid Operators

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

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

## 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	<ul style="list-style-type: none"> <li><math>tag\_b + 5</math></li> <li><math>tag\_c \text{ AND } tag\_d</math></li> <li><math>(tag\_e ** 2) \text{ MOD } (tag\_f / tag\_g)</math></li> </ul>

## 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

l  
e  
s  
s  
e  
r

↑

g  
r  
e  
a  
t  
e  
r

↓

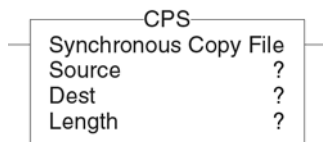
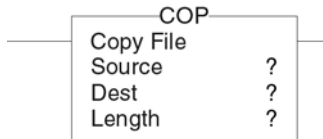
— AB < B

— a > B

## Copy File (COP) Synchronous Copy File (CPS)

The COP and CPS instructions copy the value(s) in the Source to the Destination. The Source remains unchanged.

### Operands:



### Relay Ladder

Operand	Type	Format	Description
Source	<b>SINT</b>	tag	initial element to copy
	<b>INT</b>		
	<b>DINT</b>		
	<b>REAL</b>		
	<b>string</b>		
	structure		
Destination	<b>SINT</b>	tag	initial element to be overwritten by the Source
	<b>INT</b>		
	<b>DINT</b>		
	<b>REAL</b>		
	<b>string</b>		
	structure		
Length	DINT	immediate	number of Destination elements to copy
		tag	



`COP (Source, Dest, Length) ;`

`CPS (Source, Dest, Length) ;`

### Structured Text

The operands are the same as those for the relay ladder COP and CPS instructions.

**Description:** During execution of the COP and CPS instructions, other controller actions may try to interrupt the copy operation and change the source or destination data:

If the Source Or Destination Is	And You Want To	Then Select	Notes
<ul style="list-style-type: none"> <li>produced tag</li> <li>consumed tag</li> <li>I/O data</li> <li>data that another task can overwrite</li> </ul>	prevent the data from changing during the copy operation	CPS	<ul style="list-style-type: none"> <li>Tasks that attempt to interrupt a CPS instruction are delayed until the instruction is done.</li> <li>To estimate the execution time of the CPS instruction, see <i>ControlLogix System User Manual</i>, publication 1756-UM001.</li> </ul>
	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.

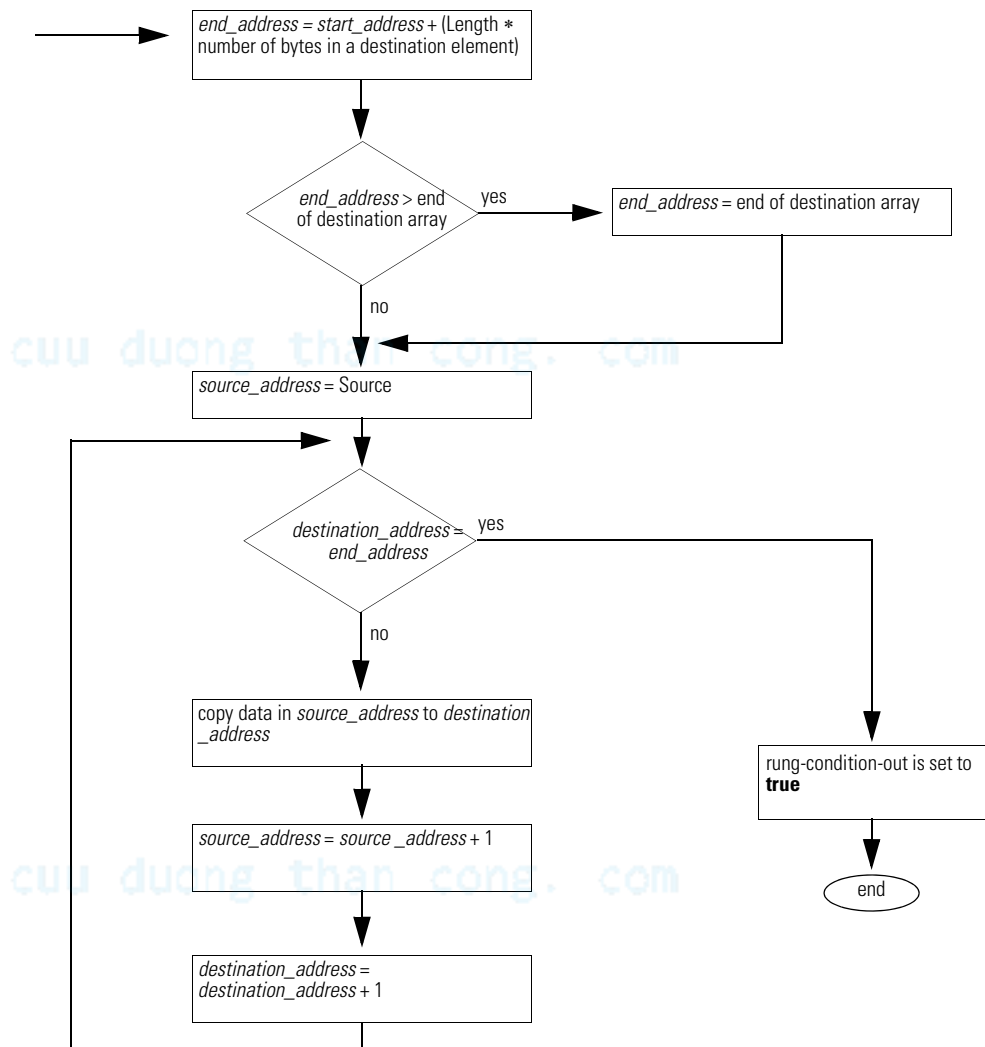
**Arithmetic Status Flags:** not affected

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

instruction execution

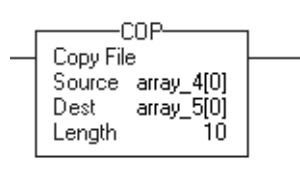


postscan	The rung-condition-out is set to false.	No action taken.
----------	---	------------------



**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

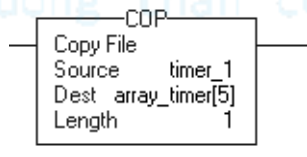


### Structured Text

```
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



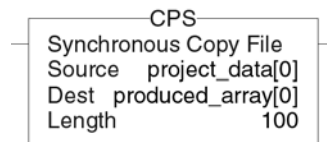
### Structured Text

```
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



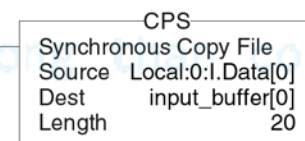
### Structured Text

```
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*.

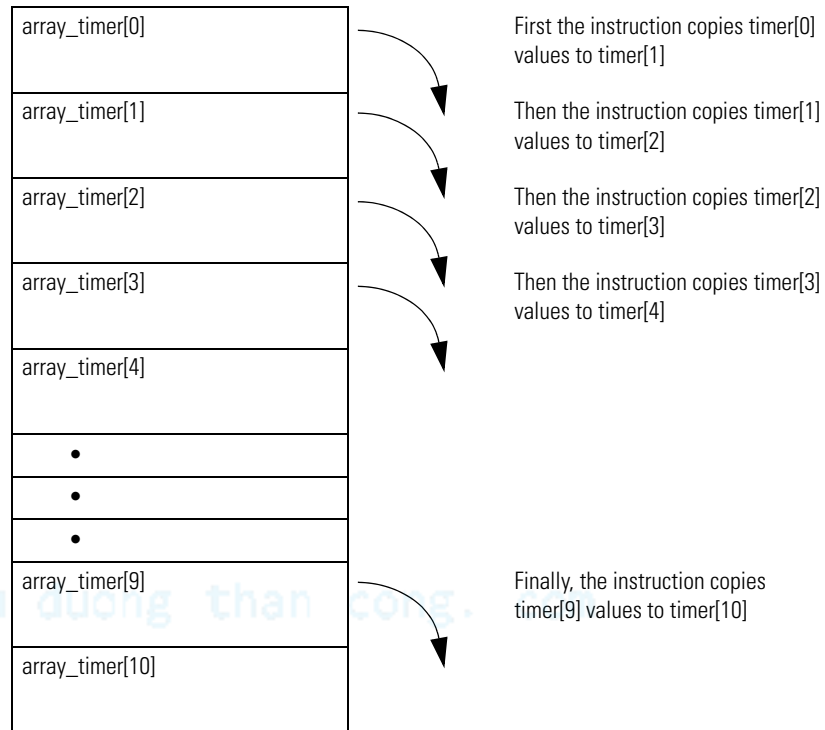
### Relay Ladder



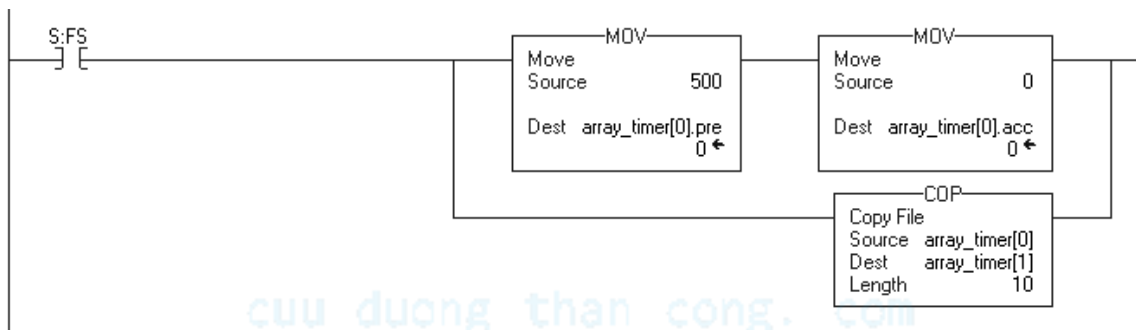
### Structured Text

```
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;

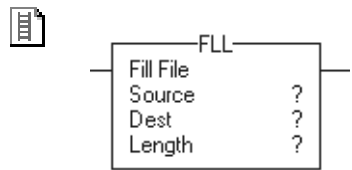
    COP(array_timer[0],array_timer[1],10);

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	<b>SINT</b>	immediate	element to copy
	<b>INT</b>	tag	<b>Important:</b> the Source and Destination operands should be the same data type, or unexpected results may occur
	<b>DINT</b>		
	<b>REAL</b>		
Destination	<b>SINT</b>	tag	initial element to be overwritten by the Source
	<b>INT</b>		<b>Important:</b> the Source and Destination operands should be the same data type, or unexpected results may occur
	<b>DINT</b>		
	<b>REAL</b>		
	structure		The preferred way to initialize a structure is to use the COP instruction.
Length	DINT	immediate	number of elements to fill



## Structured Text

Structured text does not have an FLL instruction, but you can achieve the same results using a SIZE instruction and a FOR...DO or other loop construct.

```

SIZE(destination,0,length);
FOR position = 0 TO length-1 DO
    destination[position] := source;
END_FOR;

```

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

**Description:** The number of bytes filled is:

$$\text{Byte count} = \text{Length} * (\text{number of bytes in the Destination data type})$$

### 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.

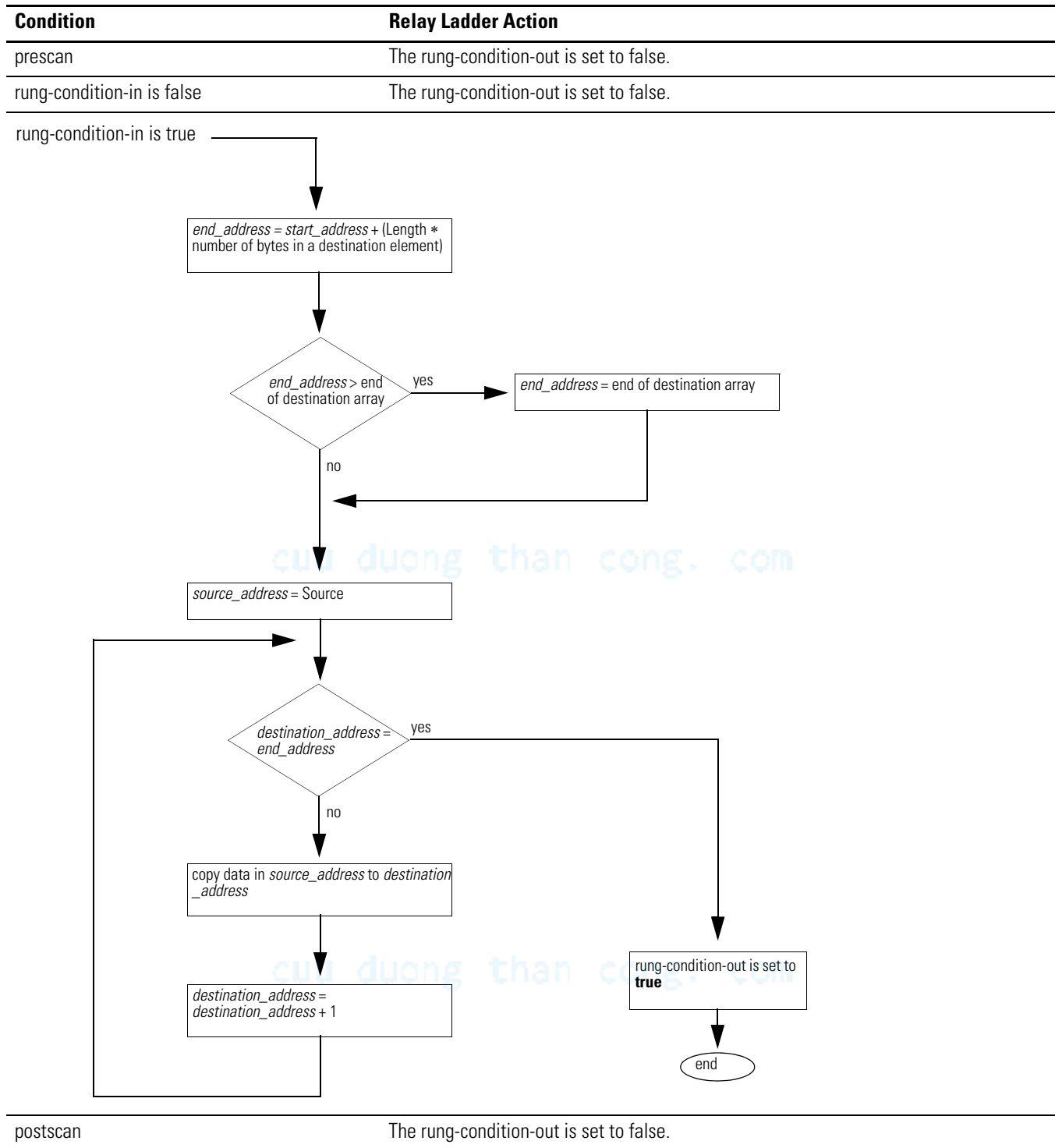
The Length is too big if it is more than the total number of elements in the Destination array.

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 9-362). 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)

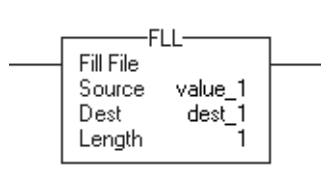
**Arithmetic Status Flags:** not affected

**Fault Conditions:** none

**Execution:**

**Example:** The FLL instruction copies the value in *value\_1* into *dest\_1*

### Relay Ladder



Source ( <i>value_1</i> ) Data Type	Source ( <i>value_1</i> ) Value	Destination ( <i>dest_1</i> ) Data Type	Destination ( <i>dest_1</i> ) Value After FLL
SINT	16#80 (-128)	DINT	16#FFFF FF80 (-128)
DINT	16#1234 5678	SINT	16#78
SINT	16#01	REAL	1.0
REAL	2.0	INT	16#0002
SINT	16#01	TIMER	16#0101 0101
			16#0101 0101
			16#0101 0101
INT	16#0001	TIMER	16#0001 0001
			16#0001 0001
			16#0001 0001
DINT	16#0000 0001	TIMER	16#0000 0001
			16#0000 0001
			16#0000 0001

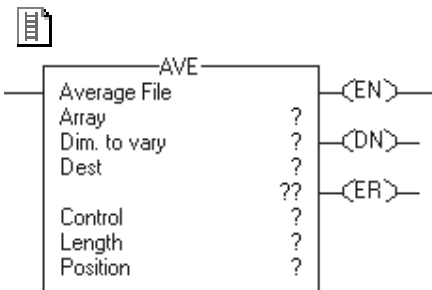
### Structured Text

```
dest_1 := value_1;
```

# File Average (AVE)

The AVE instruction calculates the average of a set of values.

## Operands:



## Relay Ladder

Operand	Type	Format	Description
Array	SINT	array tag	find the average of the values in this array
	INT		specify the first element of the group of elements to average
	DINT		<b>do not</b> use CONTROL.POS in the subscript
	<b>REAL</b>		
Dimension to vary	DINT	immediate	which dimension to use
		(0, 1, 2)	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	tag	result of the operation
	INT		
	DINT		
	<b>REAL</b>		
Control	CONTROL	tag	control structure for the operation
Length	DINT	immediate	number of elements of the array to average
Position	DINT	immediate	current element in the array
			initial value is typically 0



## Structured Text

Structured text does not have an AVE instruction, but you can achieve the same results using a SIZE instruction and a FOR...DO or other loop construct.

```

SIZE(array,0,length);
sum := 0;
FOR position = 0 TO length-1 DO
    sum := sum + array[position];
END_FOR;
```



```
destination := sum / length;
```

See Appendix B 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.

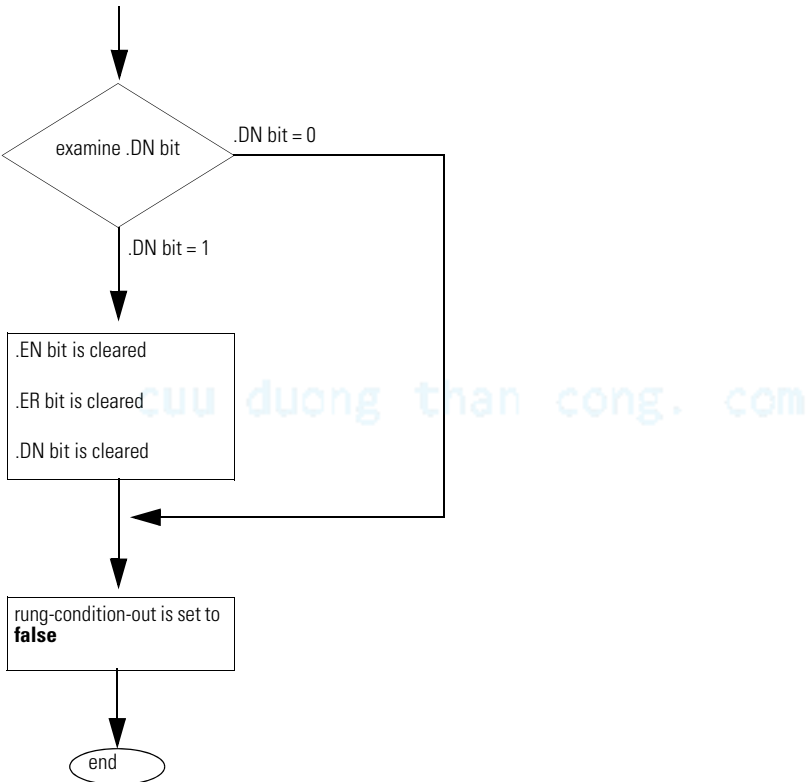
### Fault Conditions:

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 9-339.

postscan

The rung-condition-out is set to false.

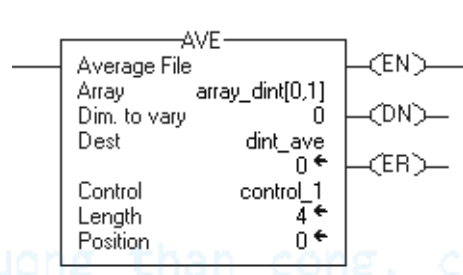
**Example 1:** Average *array\_dint*, which is DINT[4,5].

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

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

$$dint\_ave = 12$$

### Relay Ladder



### Structured Text

```

SIZE(array_dint,0,length);
sum := 0;
FOR position = 0 TO (length-1) DO
    sum := sum + array_dint[position];
END_FOR;
dint_ave := sum / length;

```

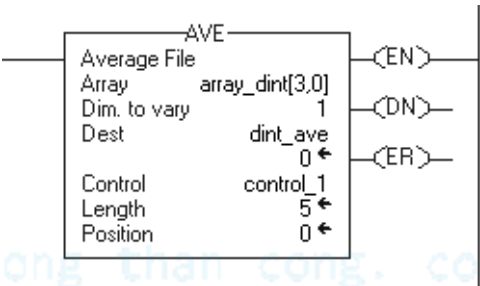
**Example 2:** Average *array\_dint*, which is DINT[4,5].

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

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

$$dint\_ave = 3$$

**Relay Ladder**



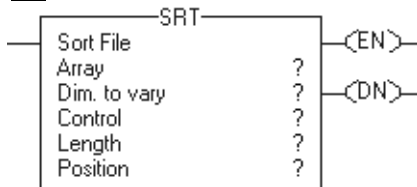
**Structured Text**

```
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	<b>SINT</b>	array tag	array to sort
	<b>INT</b>		specify the first element of the group of elements to sort
	<b>DINT</b>		<b>do not</b> use CONTROL.POS in the subscript
	<b>REAL</b>		
Dimension to vary	DINT	immediate	which dimension to use
		(0, 1, 2)	depending on the number of dimensions, the order is
			array[dim_0,dim_1,dim_2]
			array[dim_0,dim_1]
Control	CONTROL	tag	control structure for the operation
Length	DINT	immediate	number of elements of the array to sort
Position	DINT	immediate	current element in the array
			initial value is typically 0



```
SRT(Array,Dimtovary,
      Control);
```

### Structured Text

The operands are the same as those for the relay ladder SRT instruction. However, you specify the Length and Position values by accessing the .LEN and .POS members of the CONTROL structure, rather than by including values in the operand list.

### CONTROL Structure

Mnemonic	Data Type	Description
.EN	BOOL	The enable bit indicates that the SRT instruction is enabled.
.DN	BOOL	The done bit is set when the specified elements have been sorted.
.ER	BOOL	The error bit is set when either .LEN < 0 or .POS < 0. Either of these conditions also generates a major fault.
.LEN	DINT	The length specifies the number of elements in the array on which the instruction
.POS	DINT	The position contains the position of the current element that the instruction is accessing.

**Description:** The SRT instruction sorts a set of values in one dimension (Dim to vary) of the Array into ascending order.

---

**IMPORTANT**

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 Appendix B.

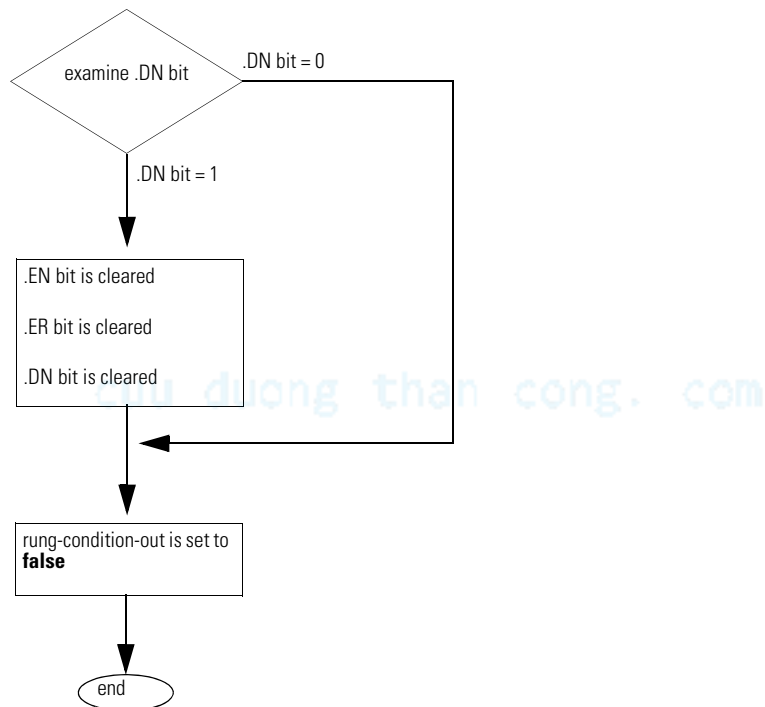
**Arithmetic Status Flags:** Arithmetic status flags are affected.

**Fault Conditions:**

A Major Fault Will Occur If	Fault Type	Fault Code
.POS < 0 or .LEN < 0	4	21
Dimension to vary does not exist for the specified array	4	20
Instruction tries to access data outside of the array boundaries	4	20

### Execution:

Condition	Relay Ladder Action	Structured Text Action
prescan	The .EN bit is cleared. The .DN bit is cleared. The .ER bit is cleared. The rung-condition-out is set to false.	The .EN bit is cleared. The .DN bit is cleared. The .ER bit is cleared.
rung-condition-in is false		na



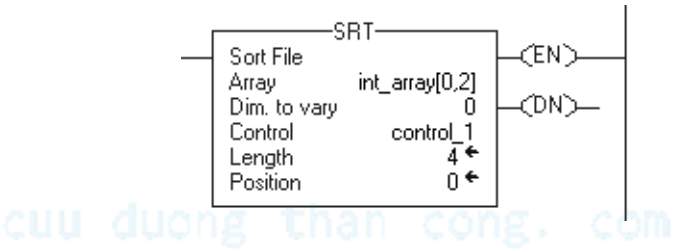
rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	na
EnableIn is set	na	EnableIn is always set.
instruction execution	The instruction sorts the specified elements of the array into ascending order.	The instruction executes. The instruction sorts the specified 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].

		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

		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);
```

cuu duong than cong. com



**Example 2:** Sort *int \_array*, which is DINT[4,5].

Before

subscripts

dimension 1

01234

02019181716

11514131211

2109876

354321

dimension 0

After

subscripts

dimension 1

01234

02019181716

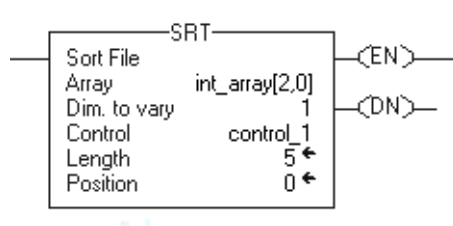
11514131211

2678910

354321

dimension 0

### Relay Ladder



### Structured Text

```
control_1.LEN := 5;

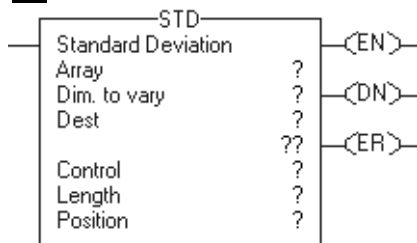
control_1.POS := 0;

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	array tag	find the standard deviation of the values in this array
	INT		specify the first element of the group of elements to use in calculating the standard deviation
	<b>DINT</b>		
	<b>REAL</b>		
			<b>do not</b> use CONTROL.POS in the subscript
A SINT or INT tag converts to a DINT value by sign-extension.			
Dimension to vary	DINT	immediate	which dimension to use
		(0, 1, 2)	depending on the number of dimensions, the order is
			array[dim_0,dim_1,dim_2]
			array[dim_0,dim_1]
			array[dim_0]
Destination	REAL	tag	result of the operation
Control	CONTROL	tag	control structure for the operation
Length	DINT	immediate	number of elements of the array to use in calculating the standard deviation
Position	DINT	immediate	current element in the array
			initial value is typically 0

### CONTROL Structure

Mnemonic	Data Type	Description
.EN	BOOL	The enable bit indicates that the STD instruction is enabled.
.DN	BOOL	The done bit is set when the calculation is complete.
.ER	BOOL	The error bit is set when the instruction generates an overflow. The instruction stops executing until the program clears the .ER bit. The position of the element that caused the overflow is stored in the .POS value.
.LEN	DINT	The length specifies the number of elements in the array on which the instruction operates.
.POS	DINT	The position contains the position of the current element that the instruction is accessing.



## Structured Text

Structured text does not have an STD instruction, but you can achieve the same results using a SIZE instruction and a FOR...DO or other loop construct.

```
SIZE(array,0,length);
sum := 0;
FOR position = 0 TO length-1 DO
    sum := sum + array[position];
END_FOR;
average := sum / length;
sum := 0;
FOR position = 0 TO length-1 DO
    sum := sum + ((array[position] - average)**2);
END_FOR;
destination := SQRT(sum / (length-1));
```

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

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

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

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 =

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

### IMPORTANT

Make sure the Length does not cause the instruction to exceed the specified Dimension to vary. If this happens, the Destination will be incorrect.

**Arithmetic Status Flags:** Arithmetic status flags are affected.

**Fault Conditions:**

A Major Fault Will Occur If	Fault Type	Fault Code
.POS < 0 or .LEN < 0	4	21
Dimension to vary does not exist for the specified array	4	20

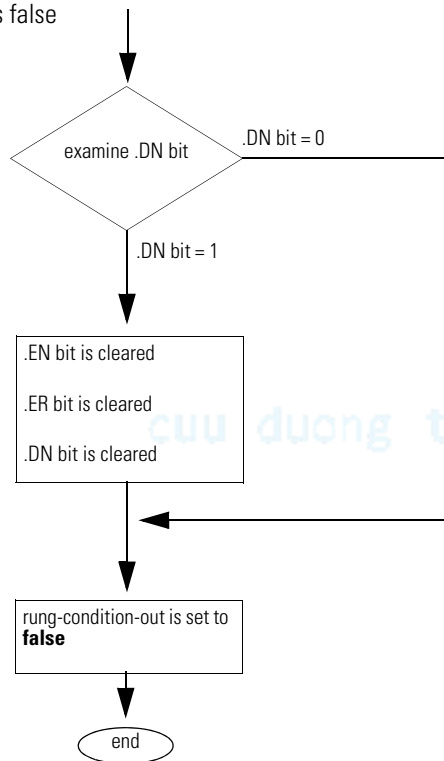
cuu duong than cong. com

cuu duong than cong. com

### 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	<p>The STD instruction calculates the standard deviation of the specified elements.</p> <p>Internally, the instruction uses a FAL instruction to calculate the average:</p> <p>Expression = standard deviation calculation</p> <p>Mode = ALL</p> <p>For details on how the FAL instruction executes, see page 9-339.</p>
postscan	The rung-condition-out is set to false.

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

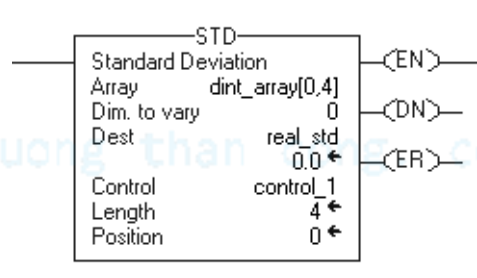
$$AVE = \frac{16 + 11 + 6 + 1}{4} = \frac{34}{4} = 8.5$$

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

$$real\_std = 6.454972$$

subscripts	dimension 1				
	0	1	2	3	4
dimension 0	0	20	19	18	17
	1	15	14	13	12
	2	10	9	8	7
	3	5	4	3	2

### Relay Ladder



### Structured Text

```

SIZE(dint_array,0,length);
sum := 0;
FOR position = 0 TO (length-1) DO
    sum := sum + dint_array[position];
END_FOR;
average := sum / length;
sum := 0;
FOR position = 0 TO (length-1) DO
    sum := sum + ((dint_array[position] - average)**2);
END_FOR;
real_std := SQRT(sum / (length-1));

```

**Example 2:** Calculate the standard deviation of `dint_array`, which is `DINT[4,5]`.

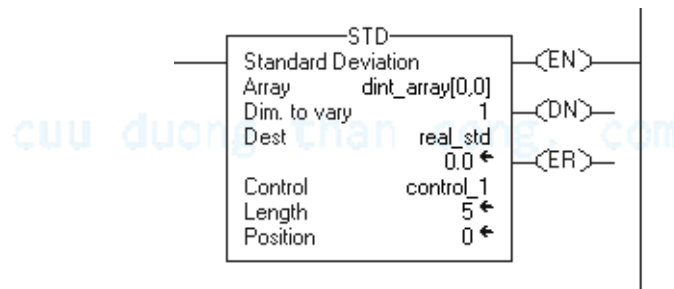
$$AVE = \frac{20 + 19 + 18 + 17 + 16}{5} = \frac{90}{5} = 18$$

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

$$real\_std = 1.581139$$

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

### Relay Ladder



### Structured Text

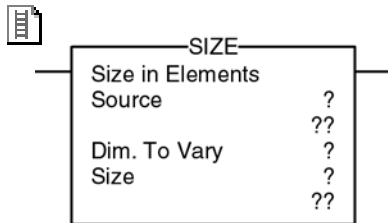
```

SIZE(dint_array,1,length);
sum := 0;
FOR position = 0 TO (length-1) DO
    sum := sum + dint_array[position];
END_FOR;
average := sum / length;
sum := 0;
FOR position = 0 TO (length-1) DO
    sum := sum + ((dint_array[position] - average)**2);
END_FOR;
real_std := SQRT(sum / (length-1));
    
```

## Size In Elements (SIZE)

The SIZE instruction finds the size of a dimension of an array.

### Operands:



### Relay Ladder

Operand	Type	Format	Description
Source	SINT	array tag	array on which the instruction is to operate
	INT		
	DINT		
	REAL		
	structure		
	string		
Dimension to Vary	DINT	immediate (0, 1, 2)	dimension to use:
		<b>For The Size Of</b>	<b>Enter</b>
		first dimension	0
		second dimension	1
		third dimension	2
Size	SINT	tag	tag to store the number of elements in the specified dimension of the array
	INT		
	DINT		
	REAL		



```
SIZE(Source,Dimtovary,Size);
```

### Structured Text

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

**Description:** The SIZE instruction finds the number of elements (size) in the designated dimension of the Source array and places the result in the Size operand.

- The instruction finds the size of one dimension of an array.
- The instruction operates on an:
  - array
  - array in a structure
  - array that is part of a larger array

**Arithmetic Status Flags:** not affected



### Execution:

**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.

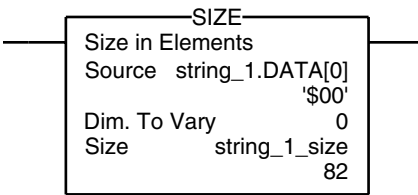
## Relay Ladder

```
SIZE(array_a, 0, array_a_size);
```

```
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**

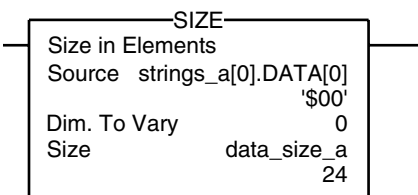


**Structured Text**

```
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);
```

## Array (File)/Shift Instructions

### (BSL, BSR, FFL, FFU, LFL, LFU)

### Introduction

Use the array (file)/shift instructions to modify the location of data within arrays.

If You Want To	Use This Instruction	Available In These Languages	See Page
Load bits into, shift bits through, and unload bits from a bit array one bit at a time.	BSL	relay ladder	388
	BSR	relay ladder	392
Load and unload values in the same order.	FFL	relay ladder	396
	FFU	relay ladder	402
Load and unload values in reverse order.	LFL	relay ladder	408
	LFU	relay ladder	414

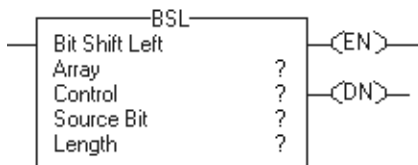
You can mix data types, but loss of accuracy and rounding errors might occur.

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 first element of the group of elements
			<b>do not</b> use CONTROL.POS in the subscript
Control	CONTROL	tag	control structure for the operation
Source bit	BOOL	tag	bit to shift
Length	DINT	immediate	number of bits in the array to shift

### CONTROL Structure

Mnemonic	Data Type	Description
.EN	BOOL	The enable bit indicates that the BSL instruction is enabled.
.DN	BOOL	The done bit is set to indicate that bits shifted one position to the left.
.UL	BOOL	The unload bit is the instruction's output. The .UL bit stores the status of the bit that was shifted out of the range of bits.
.ER	BOOL	The error bit is set when .LEN < 0.
.LEN	DINT	The length specifies the number of array bits to shift.

**Description:** When enabled, the instruction unloads the uppermost bit of the specified bits to the .UL bit, shifts the remaining bits one position left, and loads Source bit into bit 0 of Array.

### 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. In some cases, the instruction shifts bits 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

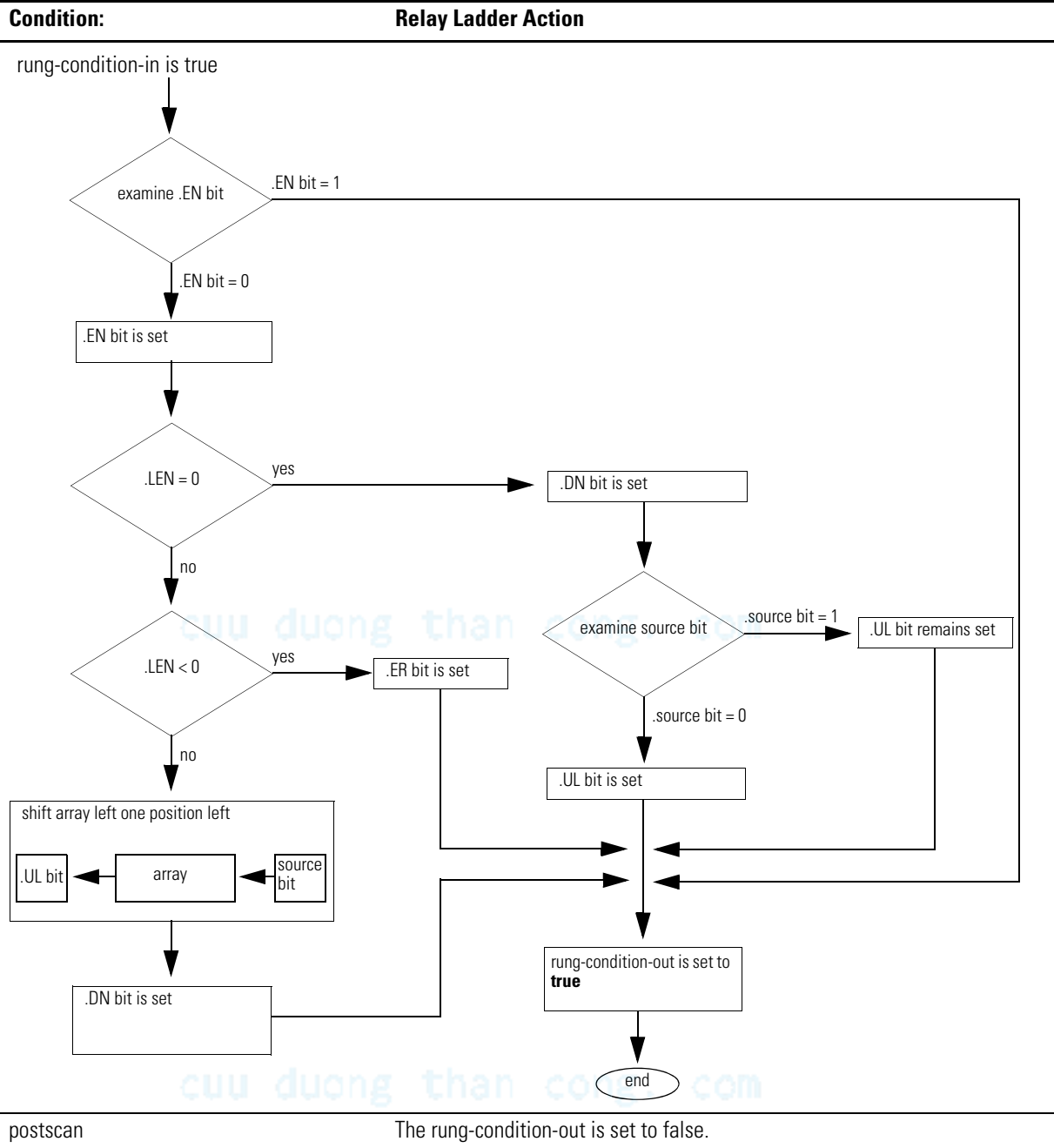
**Fault Conditions:** none

**Execution:**

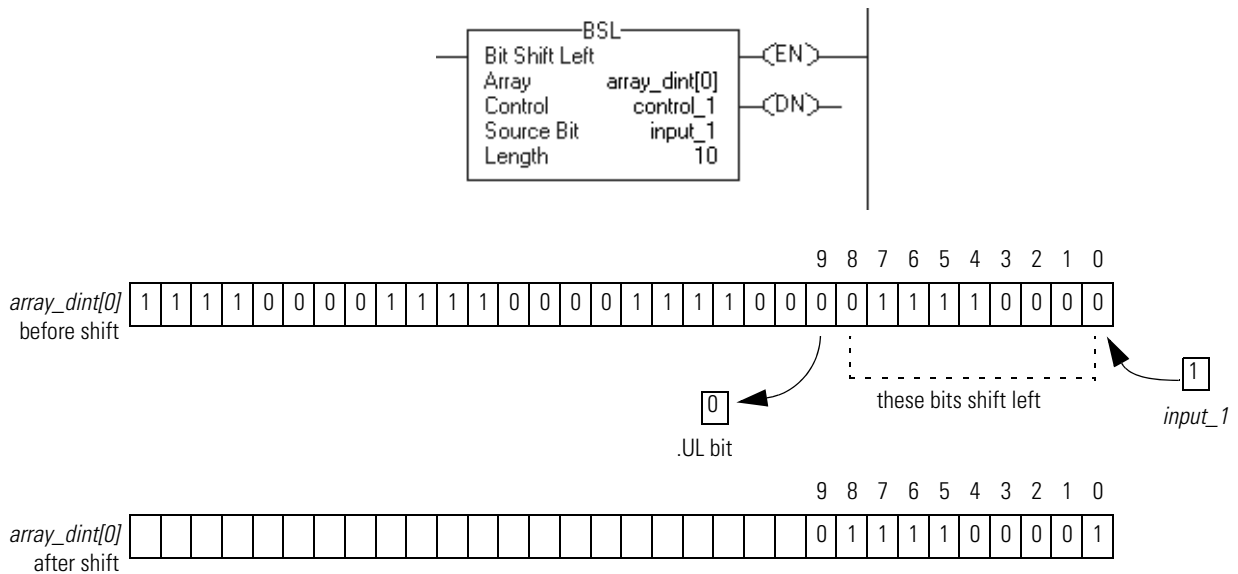
<b>Condition:</b>	<b>Relay Ladder Action</b>
prescan	The .EN bit is cleared.
	The .DN bit is cleared.
	The .ER bit is cleared.
	The .POS value is cleared.
	The rung-condition-out is set to false.
rung-condition-in is false	The .EN bit is cleared.
	The .DN bit is cleared.
	The .ER bit is cleared.
	The .POS value is cleared.
	The rung-condition-out is set to false.

cuu duong than cong. com

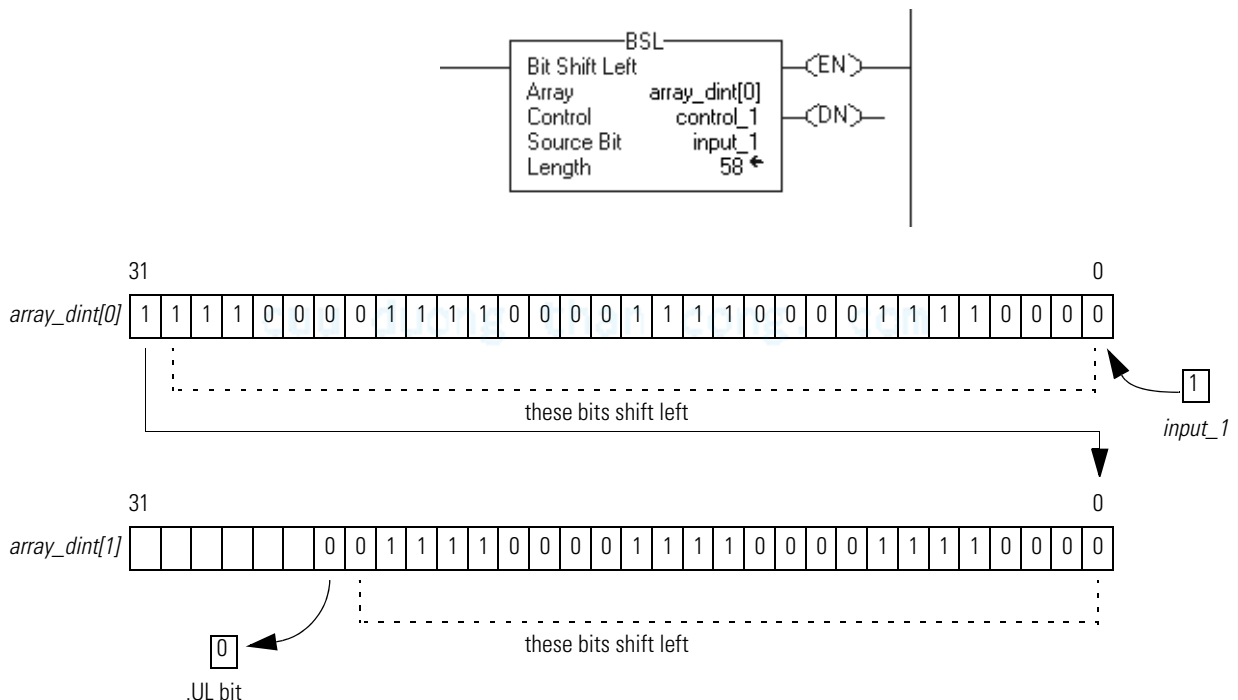
cuu duong than cong. com



**Example 1:** When enabled, the BSL instruction starts at bit 0 in *array\_dint[0]*. The instruction unloads *array\_dint[0].9* into the .UL bit, shifts the remaining bits, and loads *input\_1* into *array\_dint[0].0*. The values in the remaining bits (10-31) are invalid.



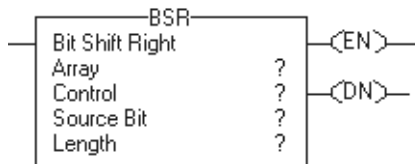
**Example 2:** When enabled, the BSL instruction starts at bit 0 in *array\_dint[0]*. The instruction unloads *array\_dint[1].25* into the .UL bit, shifts the remaining bits, and loads *input\_1* into *array\_dint[0].0*. The values in the remaining bits (31-26 in *array\_dint[1]*) are invalid. Note how *array\_dint[0].31* shifts across words to *array\_dint[1].0*.



## Bit Shift Right (BSR)

The BSR instruction shifts the specified bits within the Array one position right.

### Operands:



### Relay Ladder

Operand	Type	Format	Description
Array	DINT	array tag	array to modify
Control	CONTROL	tag	specify the element where to begin the shift <b>do not</b> use CONTROL.POS in the subscript
Source bit	BOOL	tag	control structure for the operation
Length	DINT	immediate	bit to shift
			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. In some cases, the instruction changes bits in 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:** none

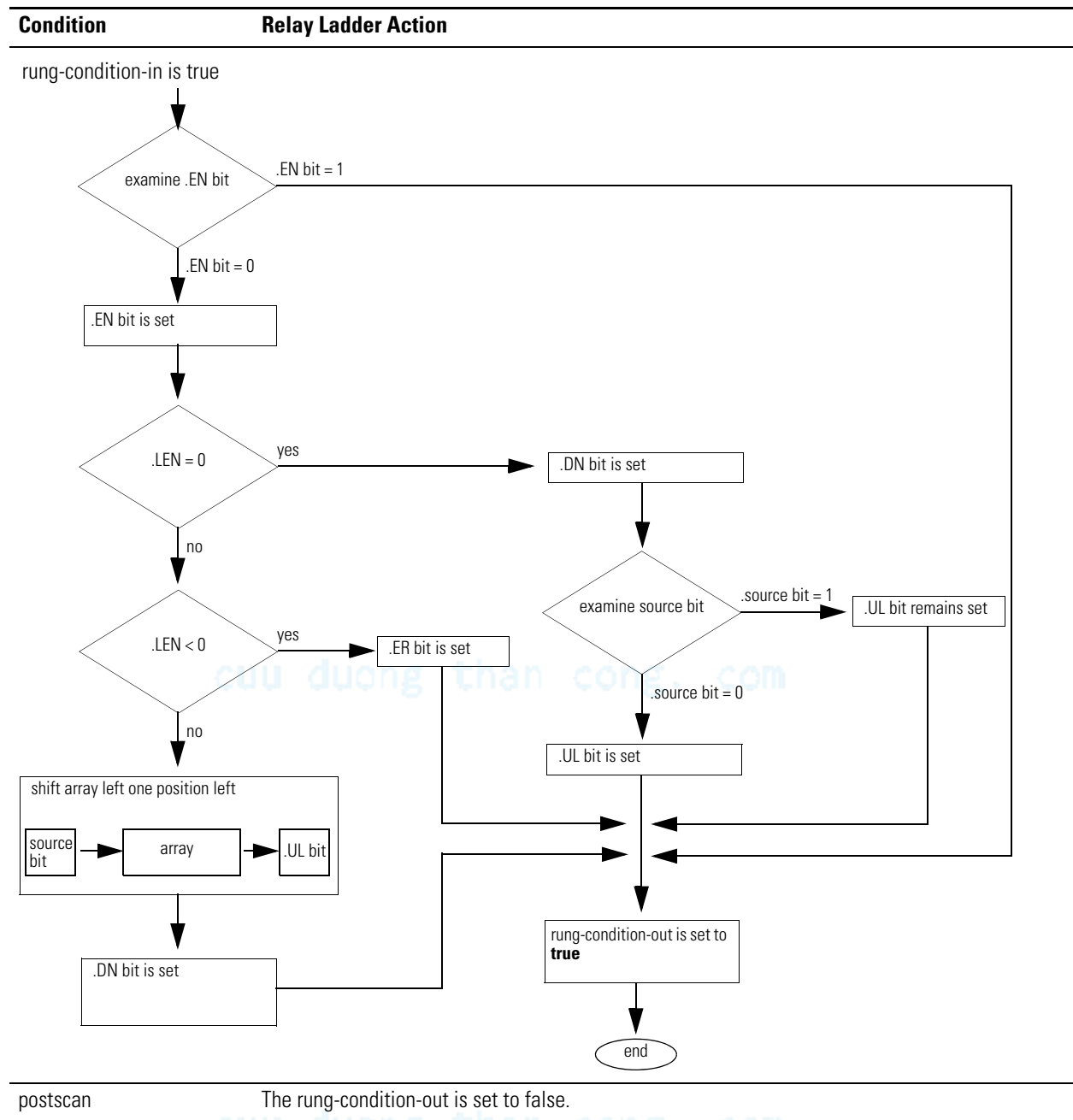


**Execution:**

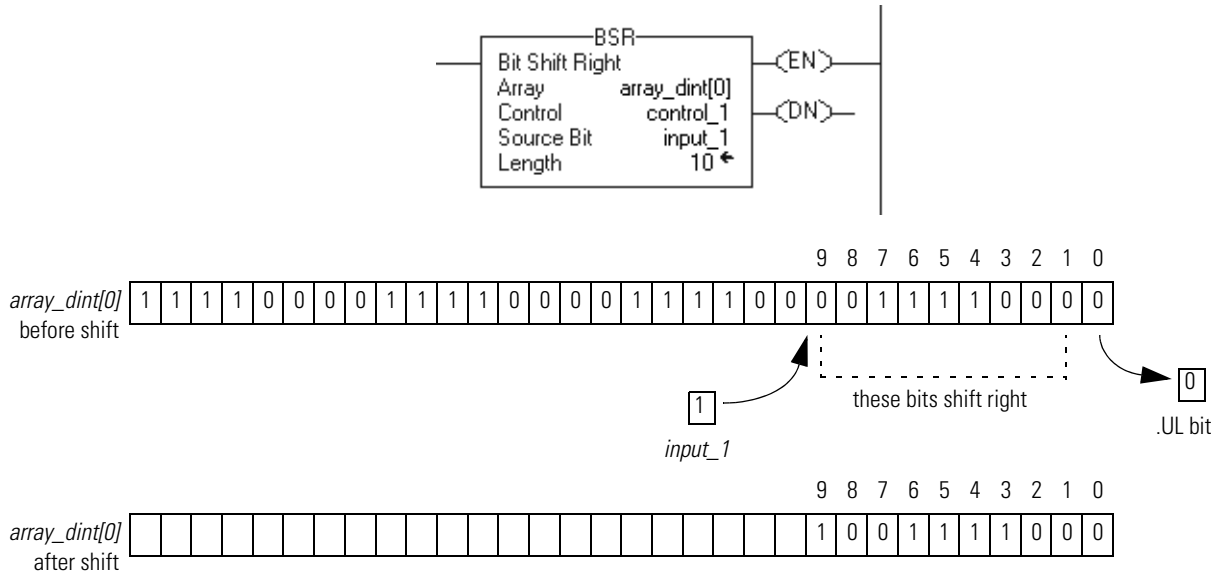
Condition	Relay Ladder Action
prescan	The .EN bit is cleared.
	The .DN bit is cleared.
	The .ER bit is cleared.
	The .POS value is cleared.
	The rung-condition-out is set to false.
rung-condition-in is false	The .EN bit is cleared.
	The .DN bit is cleared.
	The .ER bit is cleared.
	The .POS value is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

cuu duong than cong. com

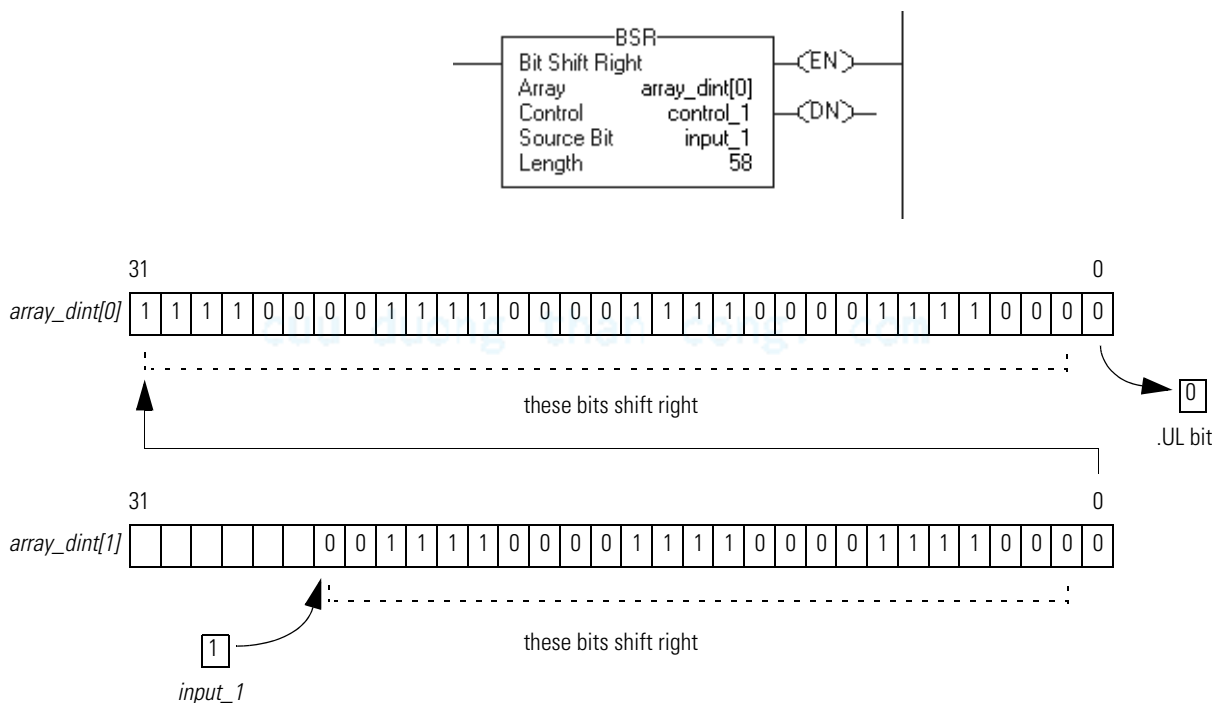
cuu duong than cong. com



**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.



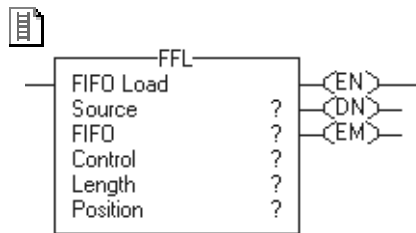
**Example 2:** When enabled, the BSR instruction starts at bit 25 in *array\_dint[1]*. The instruction unloads *array\_dint[0].0* into the .UL bit, shifts the remaining bits right, and loads *input\_1* into *array\_dint[1].25*. The values in the remaining bits (31-26 in *dint\_array[1]*) are invalid. 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	immediate	data to be stored in the FIFO
	INT	tag	
	DINT		
	REAL		
	string		
	structure		The Source converts to the data type of the array tag. A smaller integer converts to a larger integer by sign-extension.
FIFO	SINT	array tag	FIFO to modify
	INT		specify the first element of the FIFO
	DINT		<b>do not</b> use CONTROL.POS in the subscript
	REAL		
	string		
	structure		
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 $.LEN \leq 0$ 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.

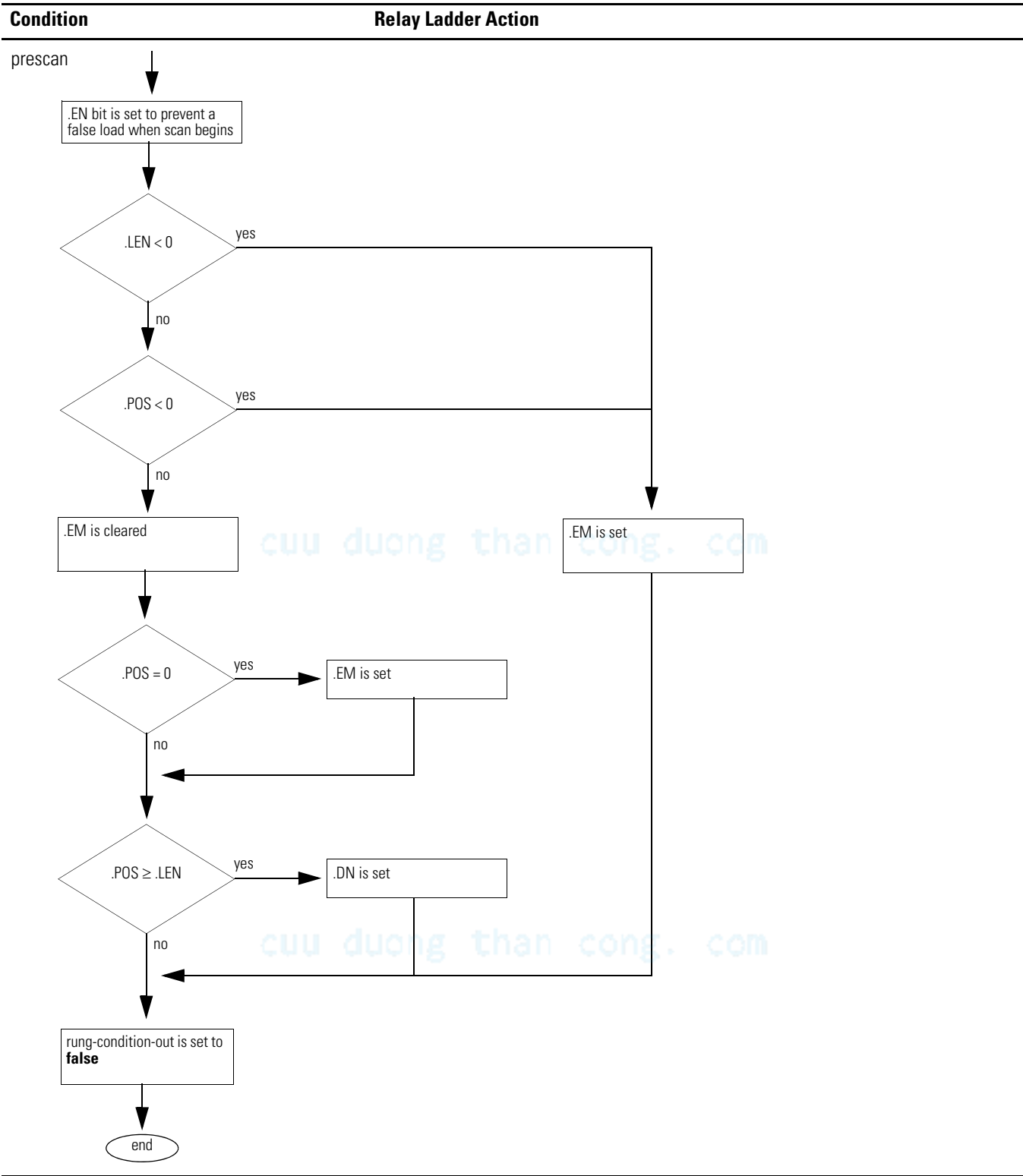
The FFL instruction operates on contiguous memory. In some cases, the instruction loads data past the array into other members of the tag. This happens if the length is too big and the tag is a user-defined data type.

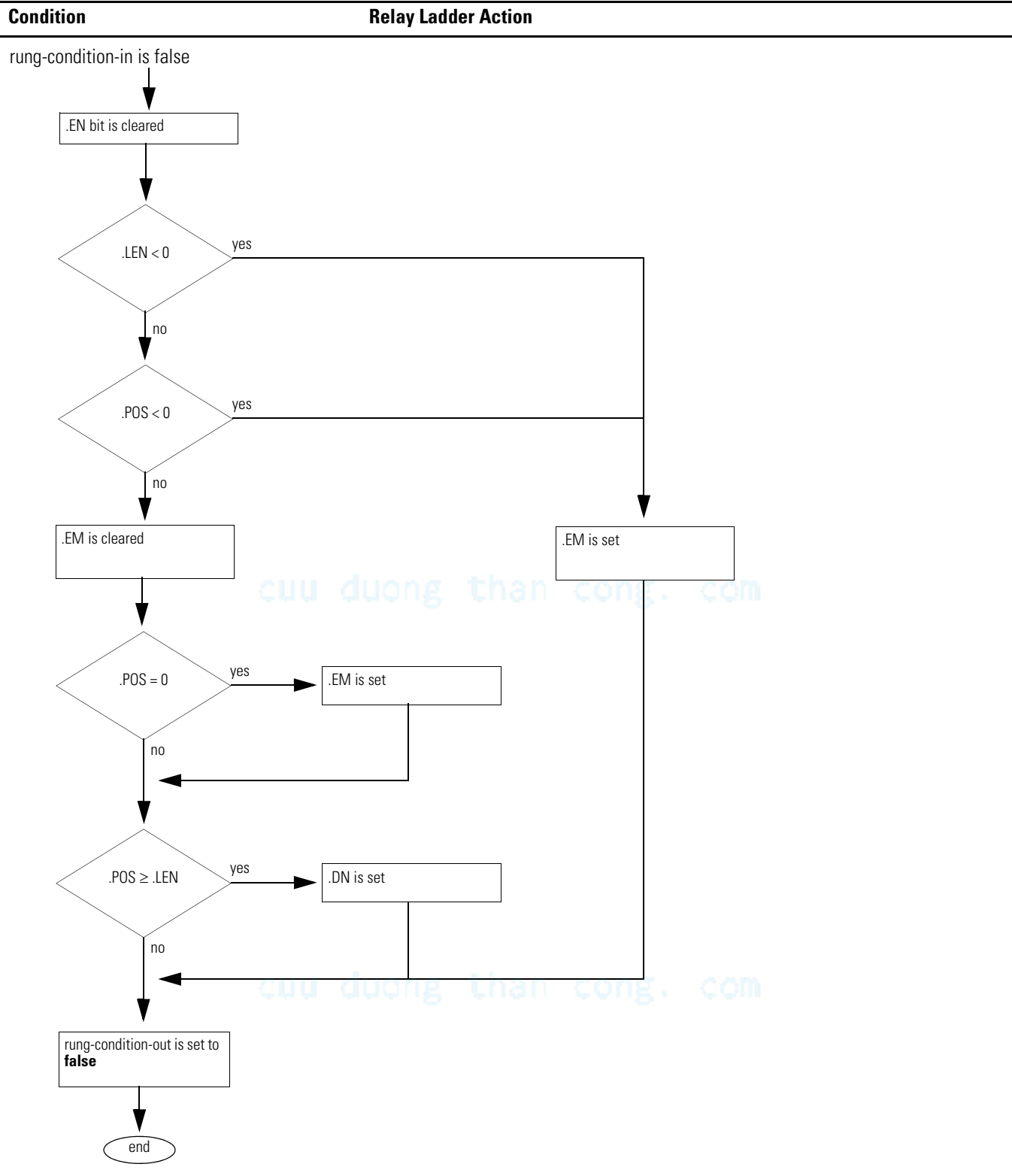
**Arithmetic Status Flags:** not affected

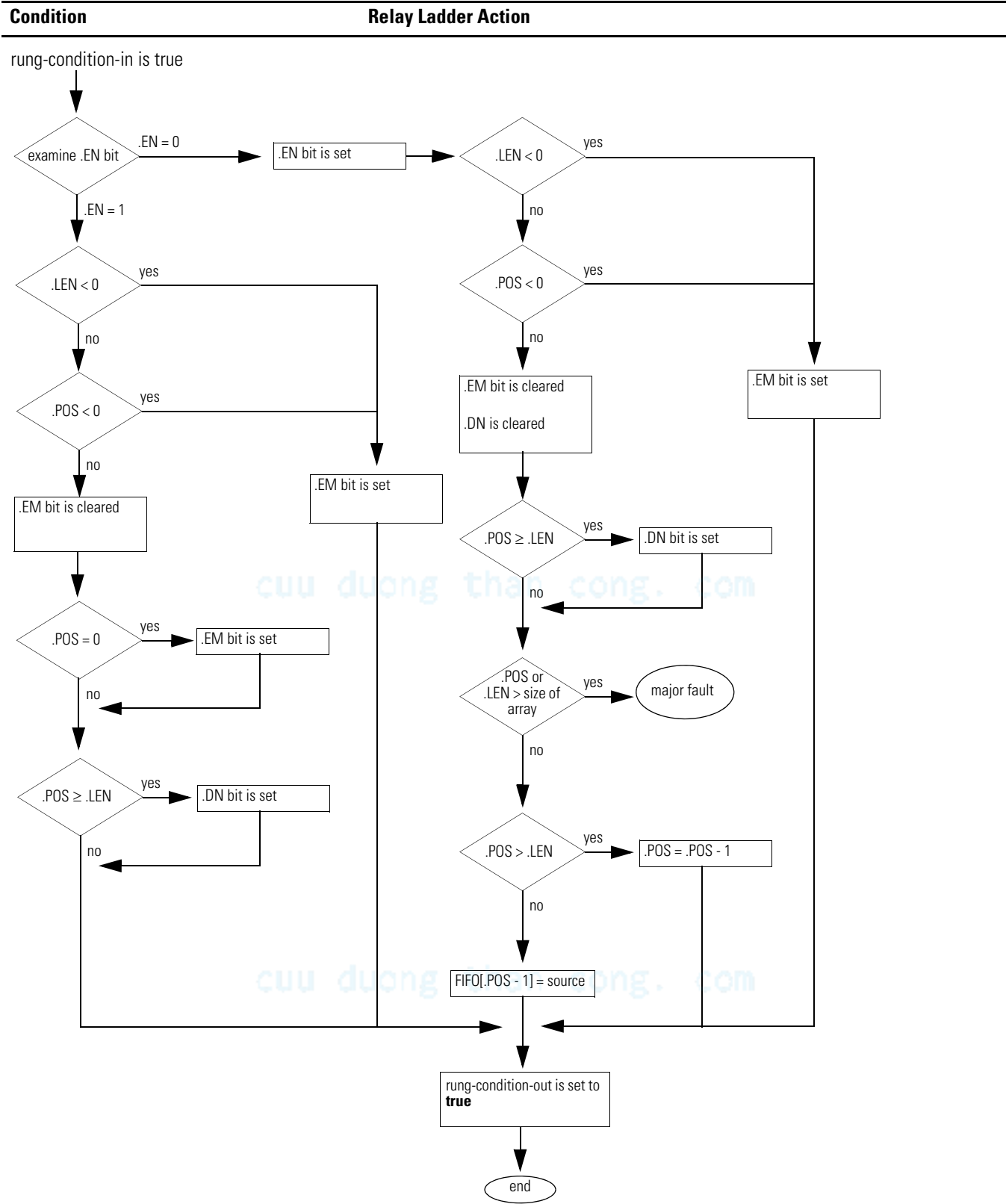
### Fault Conditions:

A Major Fault Will Occur If	Fault Type	Fault Code
(starting element + .POS) > FIFO array size	4	20

**Execution:**





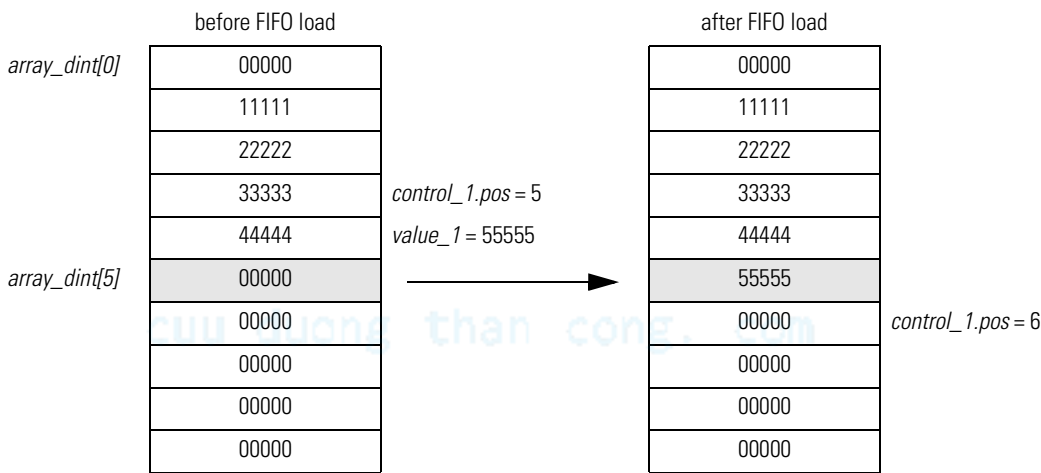
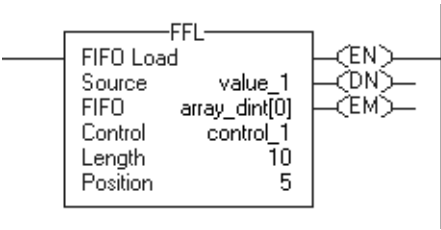


postscan

The rung-condition-out is set to false.



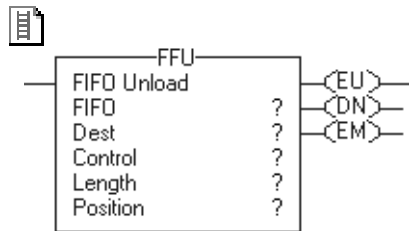
**Example:** When enabled, the FFL instruction loads *value\_1* into the next position in the FIFO, which is *array\_dint[5]* in this example.



# 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	array tag	FIFO to modify
	INT		specify the first element of the FIFO
	DINT		<b>do not</b> use CONTROL.POS in the subscript
	REAL		
	string		
	structure		
Destination	SINT	tag	value that exits the FIFO
	INT		
	DINT		
	REAL		
	string		
	structure		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.

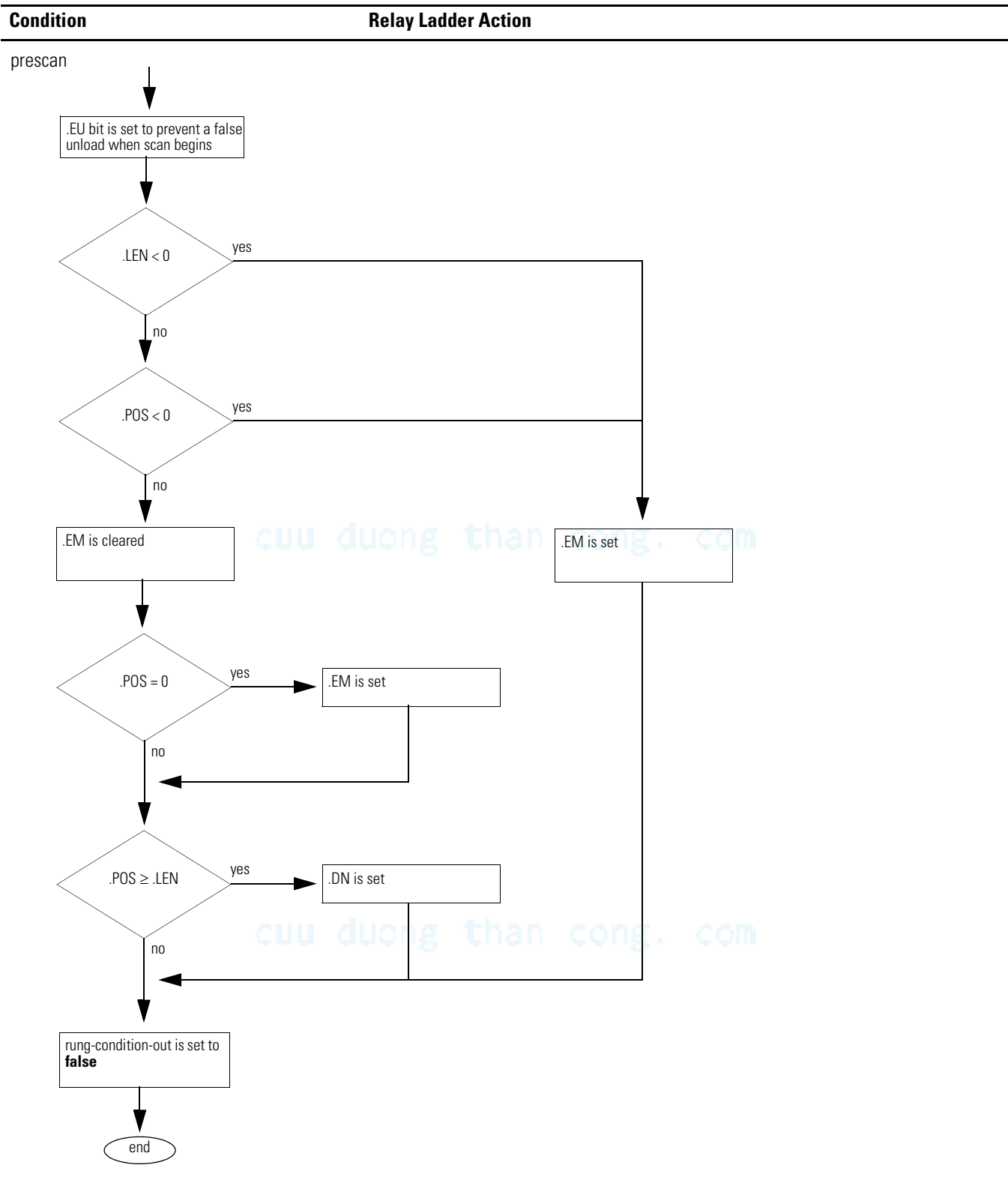
The FFU instruction operates on contiguous memory. In some cases, the instruction unloads data from other members of the tag. This happens if the length is too big and the tag is a user-defined data type.

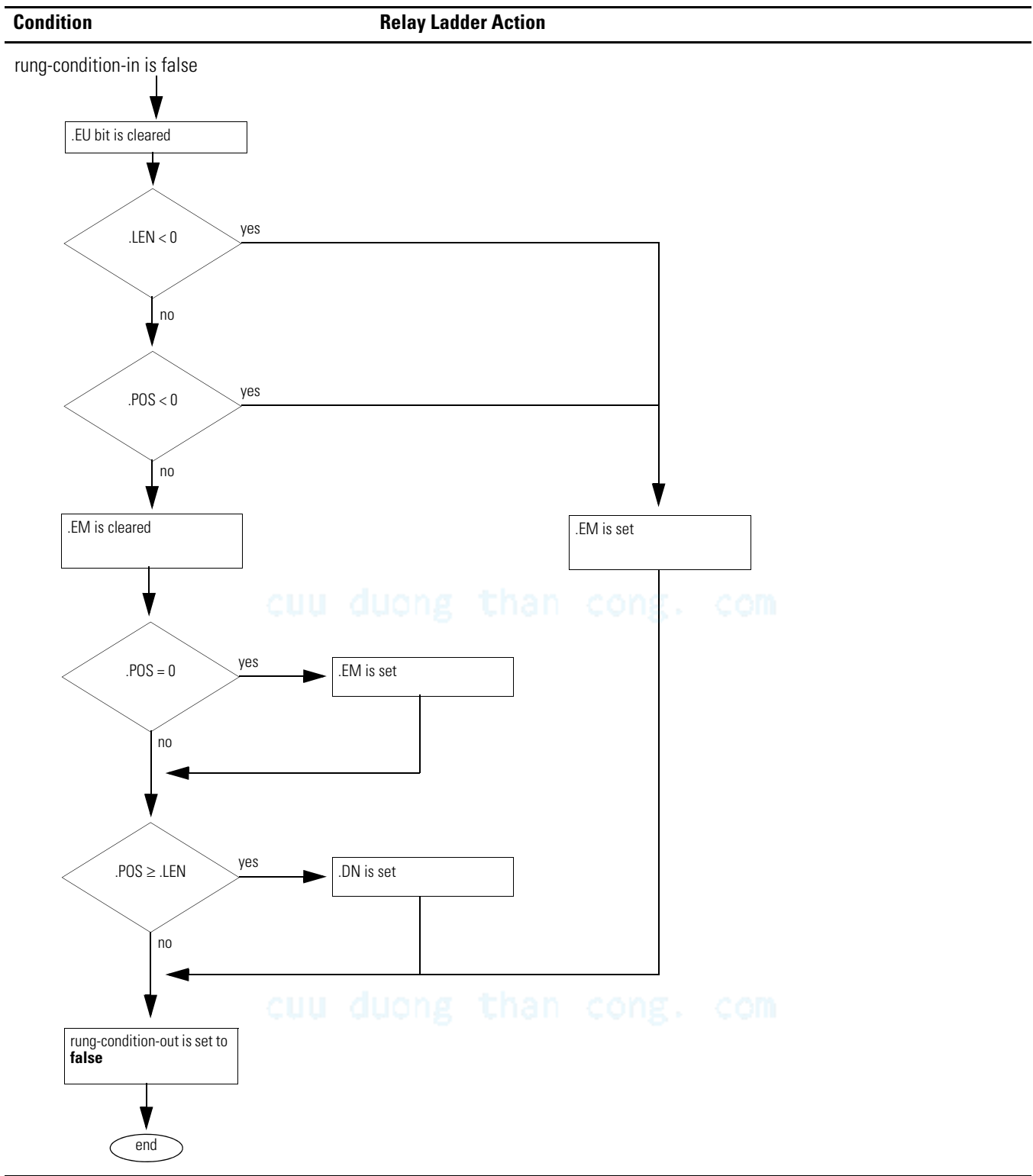
**Arithmetic Status Flags:** not affected

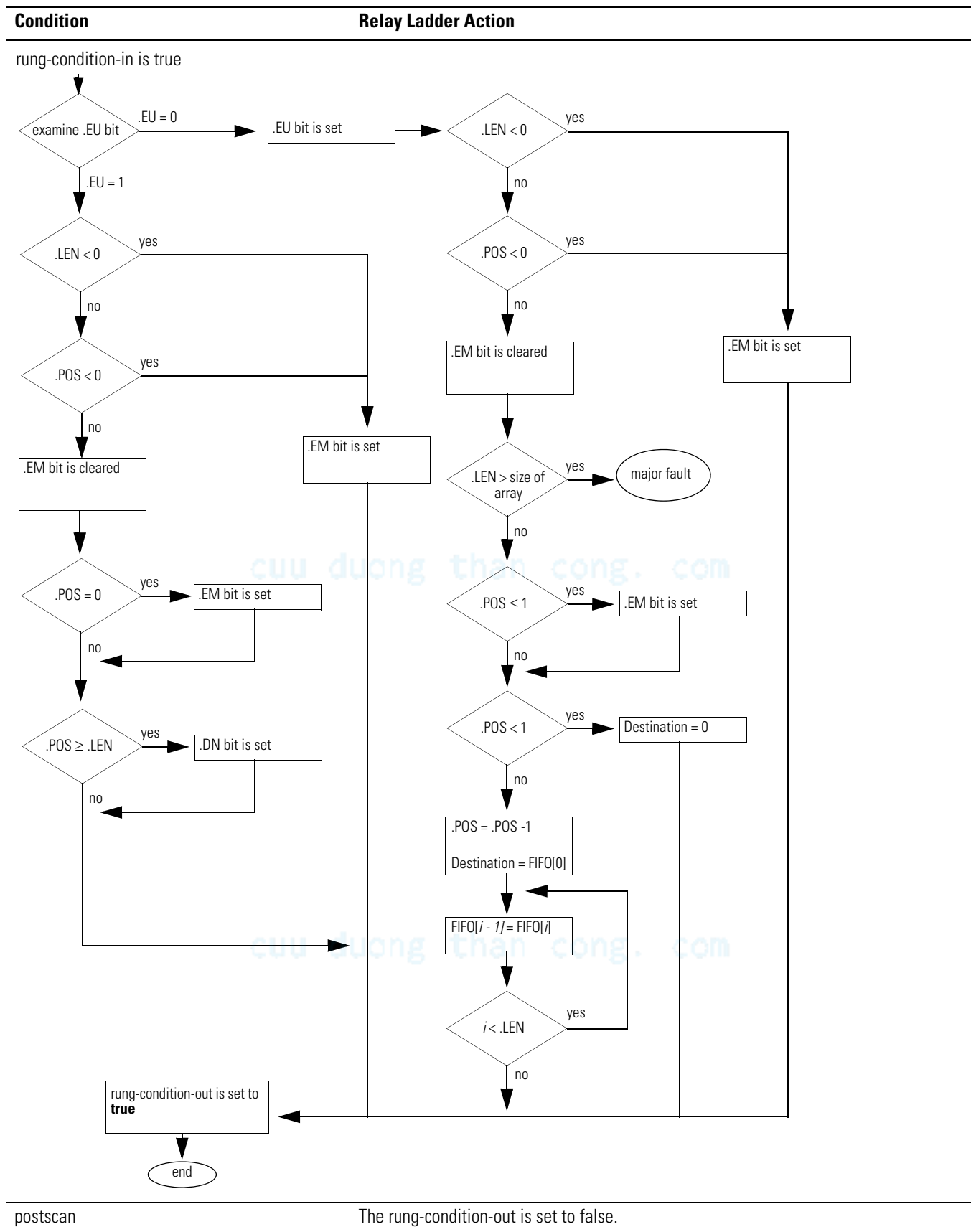
### Fault Conditions:

A Major Fault Will Occur If	Fault Type	Fault Code
Length > FIFO array size	4	20

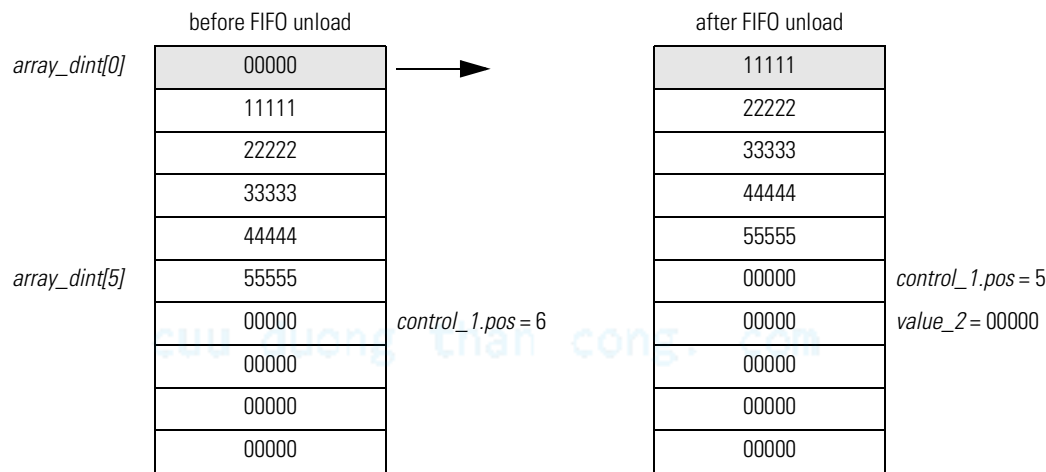
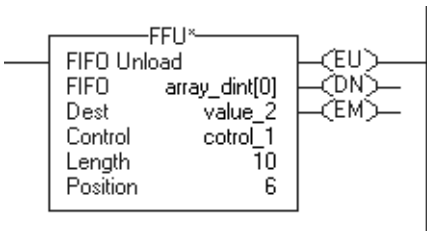
**Execution:**







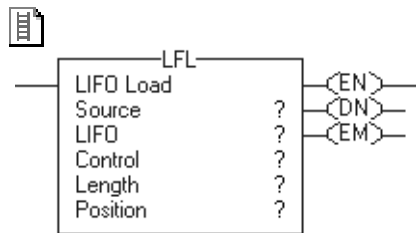
**Example:** When enabled, the FFU instruction unloads *array\_dint[0]* into *value\_2* and shifts the remaining elements in *array\_dint*.



# LIFO Load (LFL)

The LFL instruction copies the Source value to the LIFO.

## Operands:



## Relay Ladder

Operand	Type	Format	Description
Source	<b>SINT</b>	immediate	data to be stored in the LIFO
	<b>INT</b>	tag	
	<b>DINT</b>		
	<b>REAL</b>		
	<b>string</b>		
LIFO	<b>structure</b>		The Source converts to the data type of the array tag. A smaller integer converts to a larger integer by sign-extension.
	<b>SINT</b>	array tag	LIFO to modify
	<b>INT</b>		specify the first element of the LIFO
	<b>DINT</b>		<b>do not</b> use CONTROL.POS in the subscript
	<b>REAL</b>		
Control	<b>CONTROL</b>	tag	control structure for the operation
			typically use the same CONTROL as the associated LFU
Length	<b>DINT</b>	immediate	maximum number of elements the LIFO can hold at one time
Position	<b>DINT</b>	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 $.LEN \leq 0$ 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.

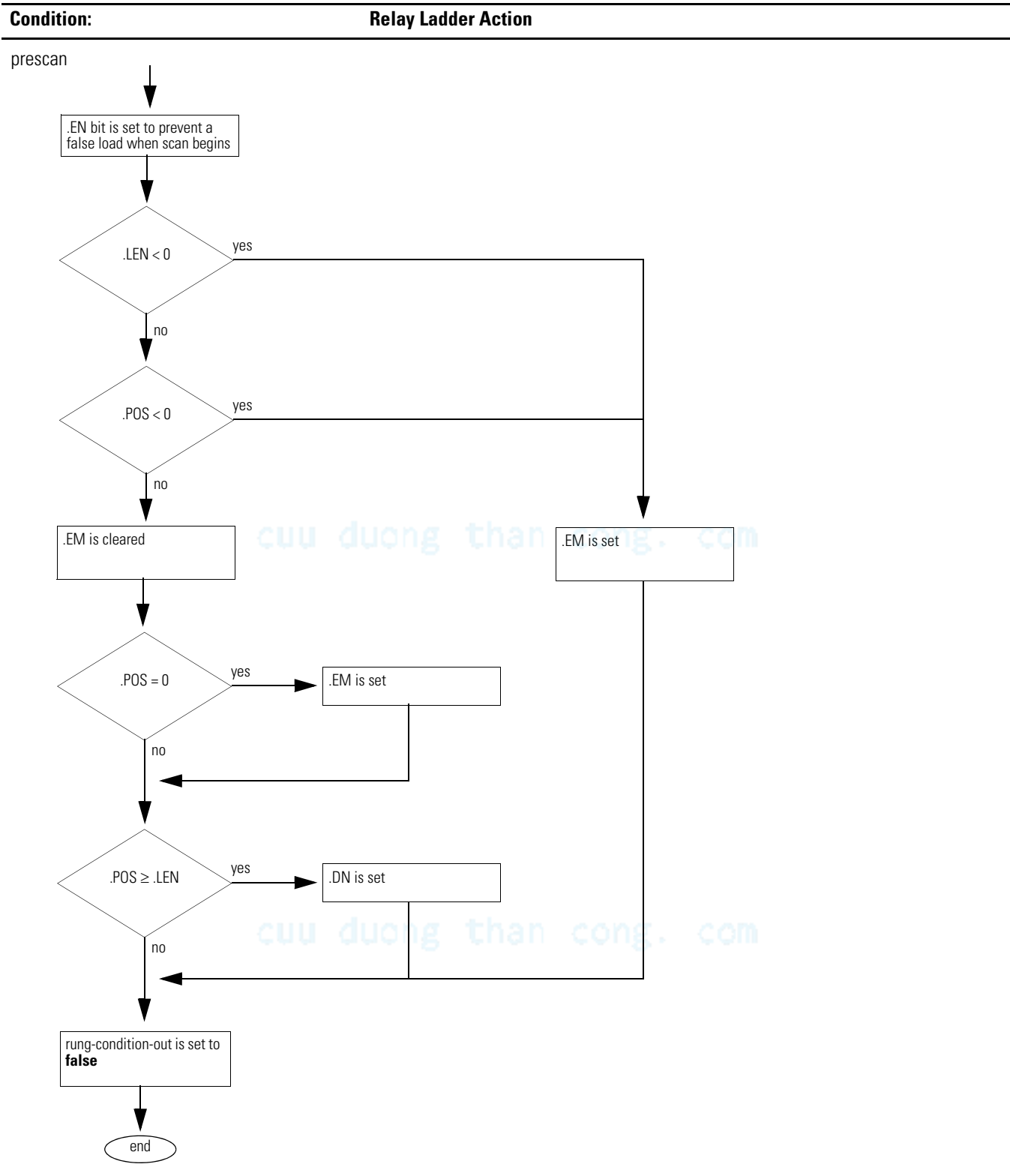
The LFL instruction operates on contiguous memory. In some cases, the instruction loads data past the array into other members of the tag. This happens if the length is too big and the tag is a user-defined data type.

**Arithmetic Status Flags:** not affected

### Fault Conditions:

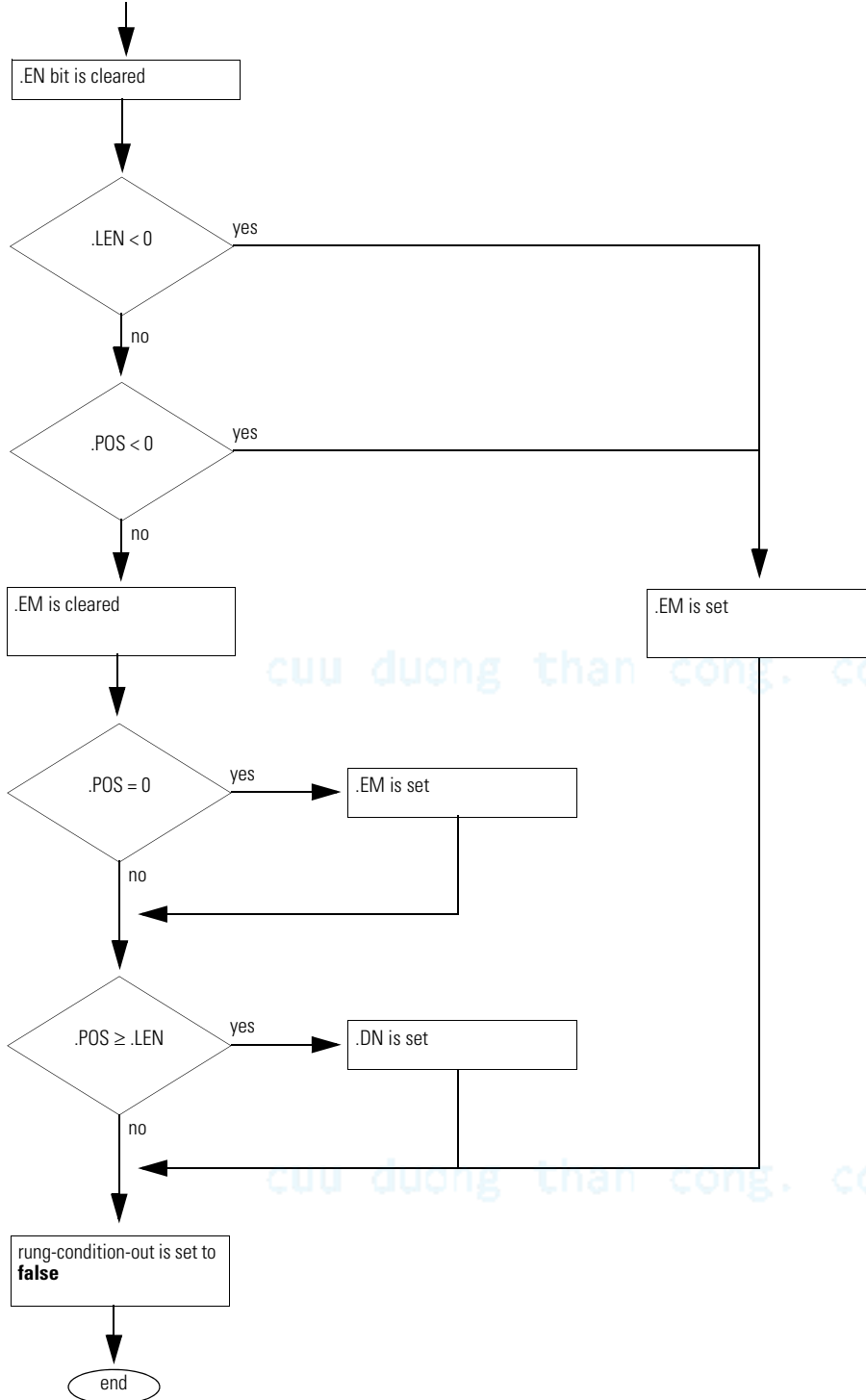
A Major Fault Will Occur If	Fault Type	Fault Code
(starting element + .POS) > LIFO array size	4	20

**Execution:**



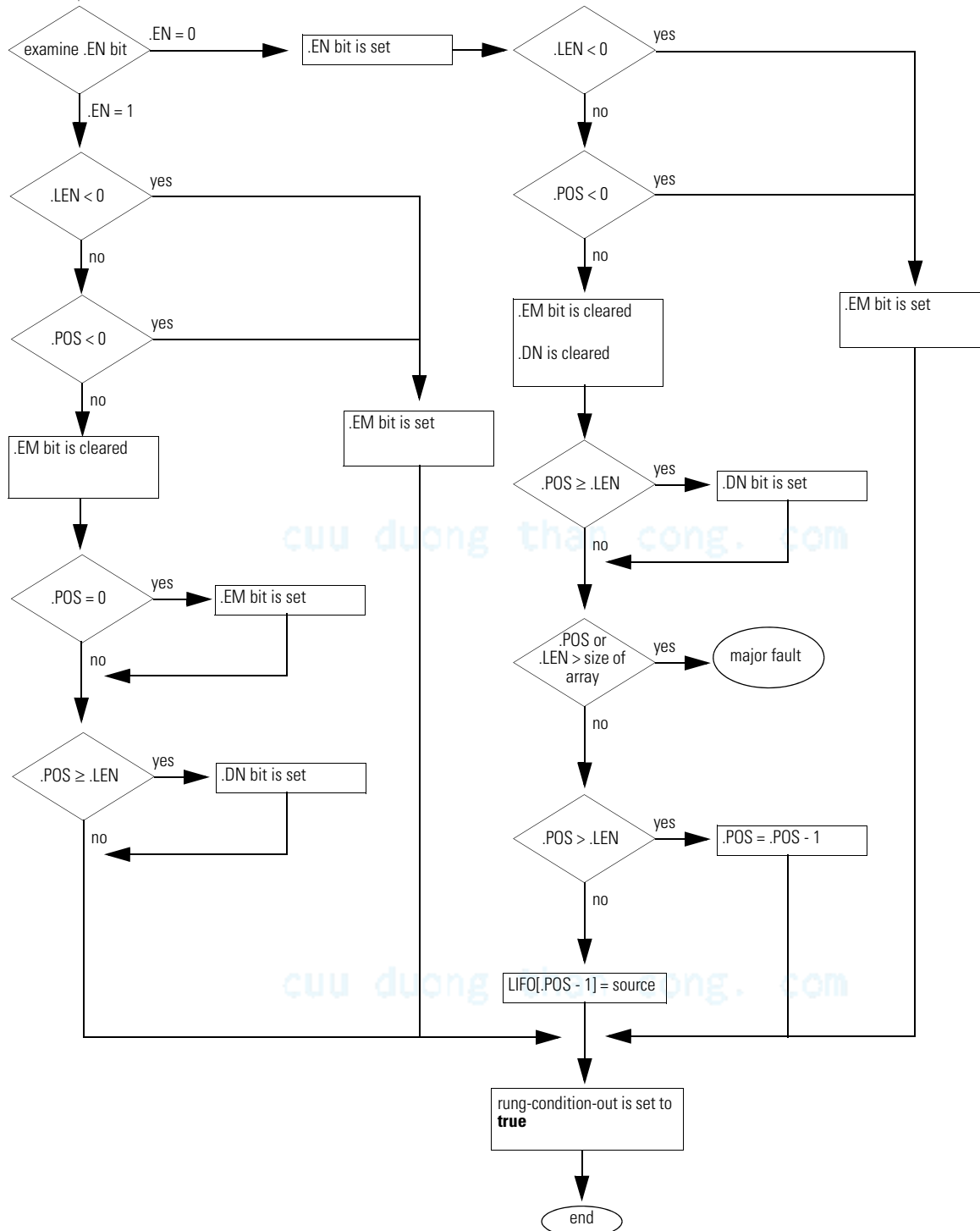
**Condition:****Relay Ladder Action**

rung-condition-in is false



**Condition:****Relay Ladder Action**

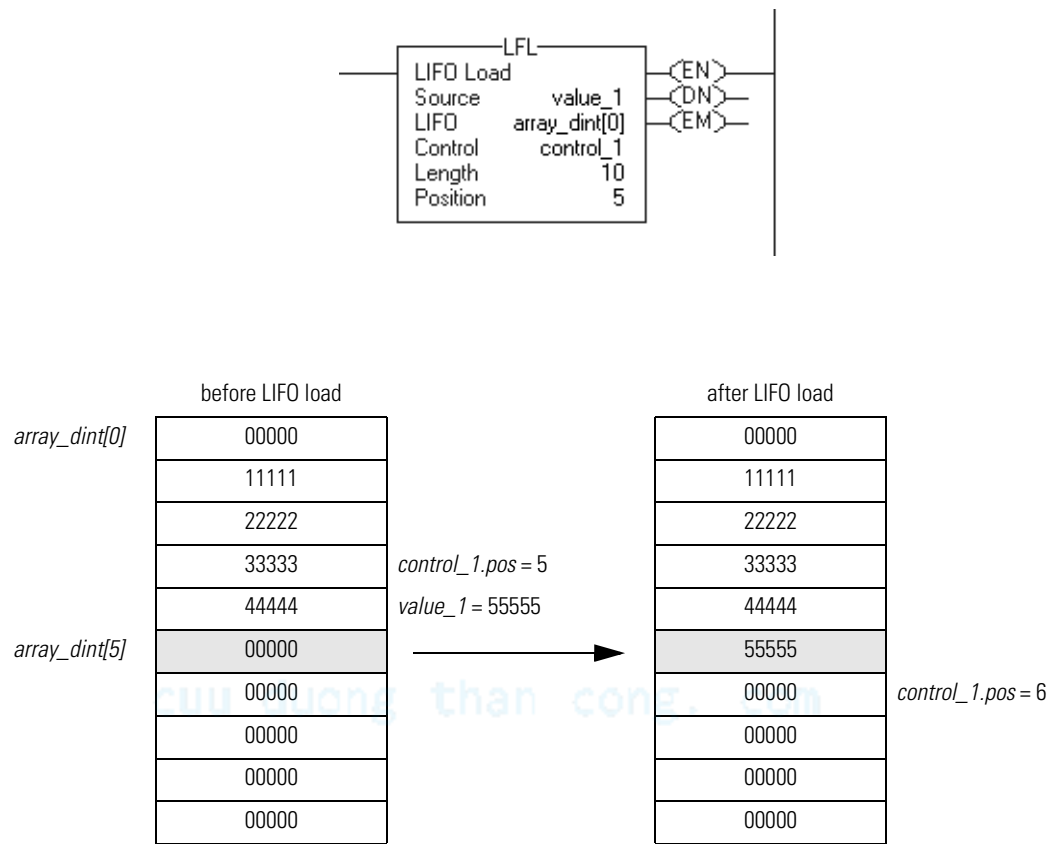
rung-condition-in is true



postscan

The rung-condition-out is set to false.

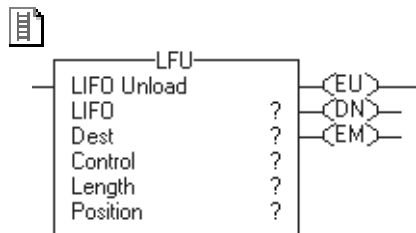
**Example:** When enabled, the LFL instruction loads *value\_1* into the next position in the LIFO, which is *array\_dint[5]* in this example.



# LIFO Unload (LFU)

The LFU instruction unloads the value at .POS of the LIFO and stores 0 in that location.

## Operands:



## Relay Ladder

Operand	Type	Format	Description
LIFO	SINT	array tag	LIFO to modify
	INT		specify the first element of the LIFO
	DINT		<b>do not</b> use CONTROL.POS in the subscript
	REAL		
	string		
	structure		
Destination	SINT	tag	value that exits the LIFO
	INT		
	DINT		
	REAL		
	string		
	structure		
	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.

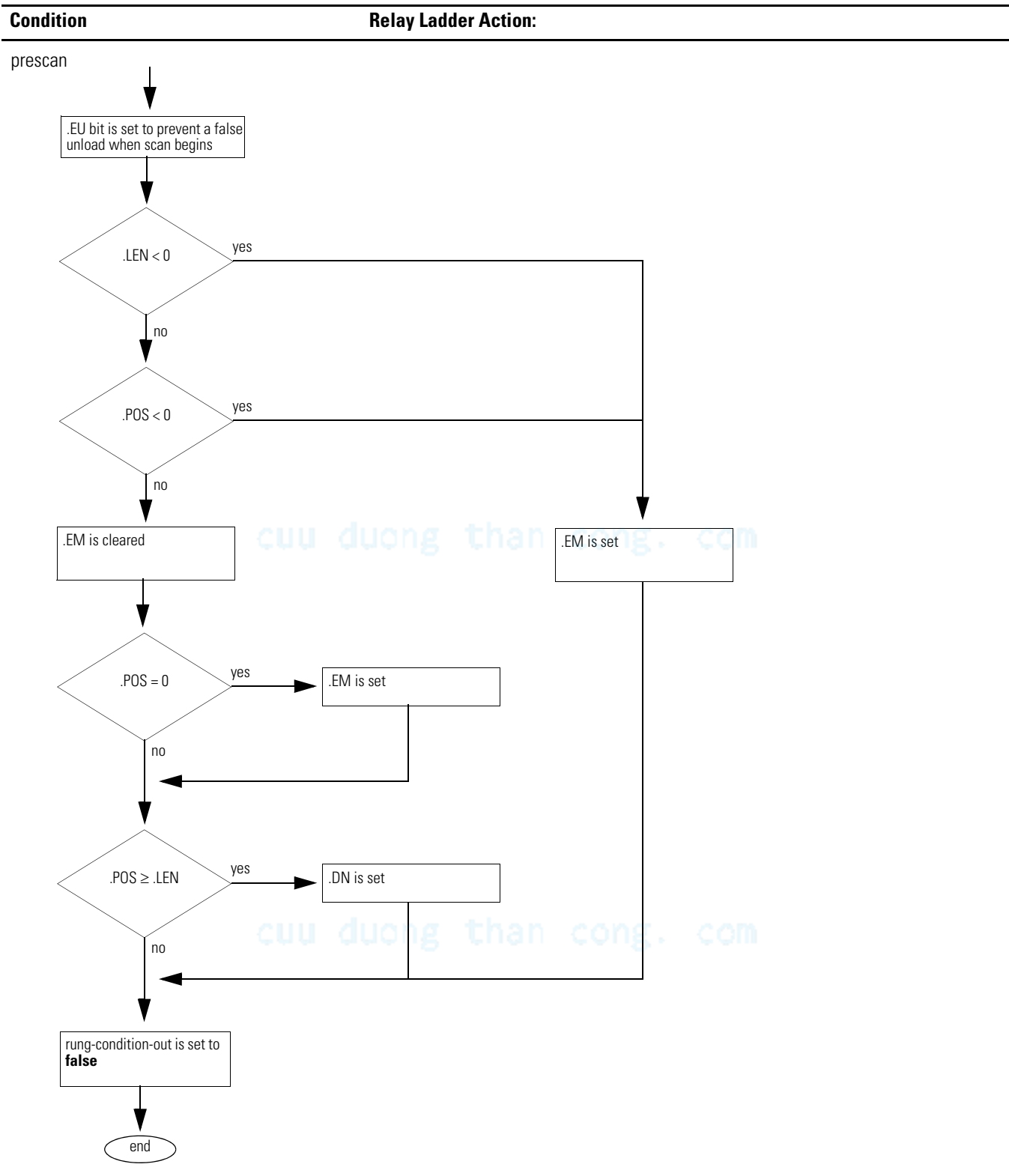
The LFU instruction operates on contiguous memory. In some cases, the instruction unloads data from other members of the tag. This happens if the length is too big and the tag is a user-defined data type.

**Arithmetic Status Flags:** not affected

### Fault Conditions:

A Major Fault Will Occur If	Fault Type	Fault Code
Length > LIFO array size	4	20

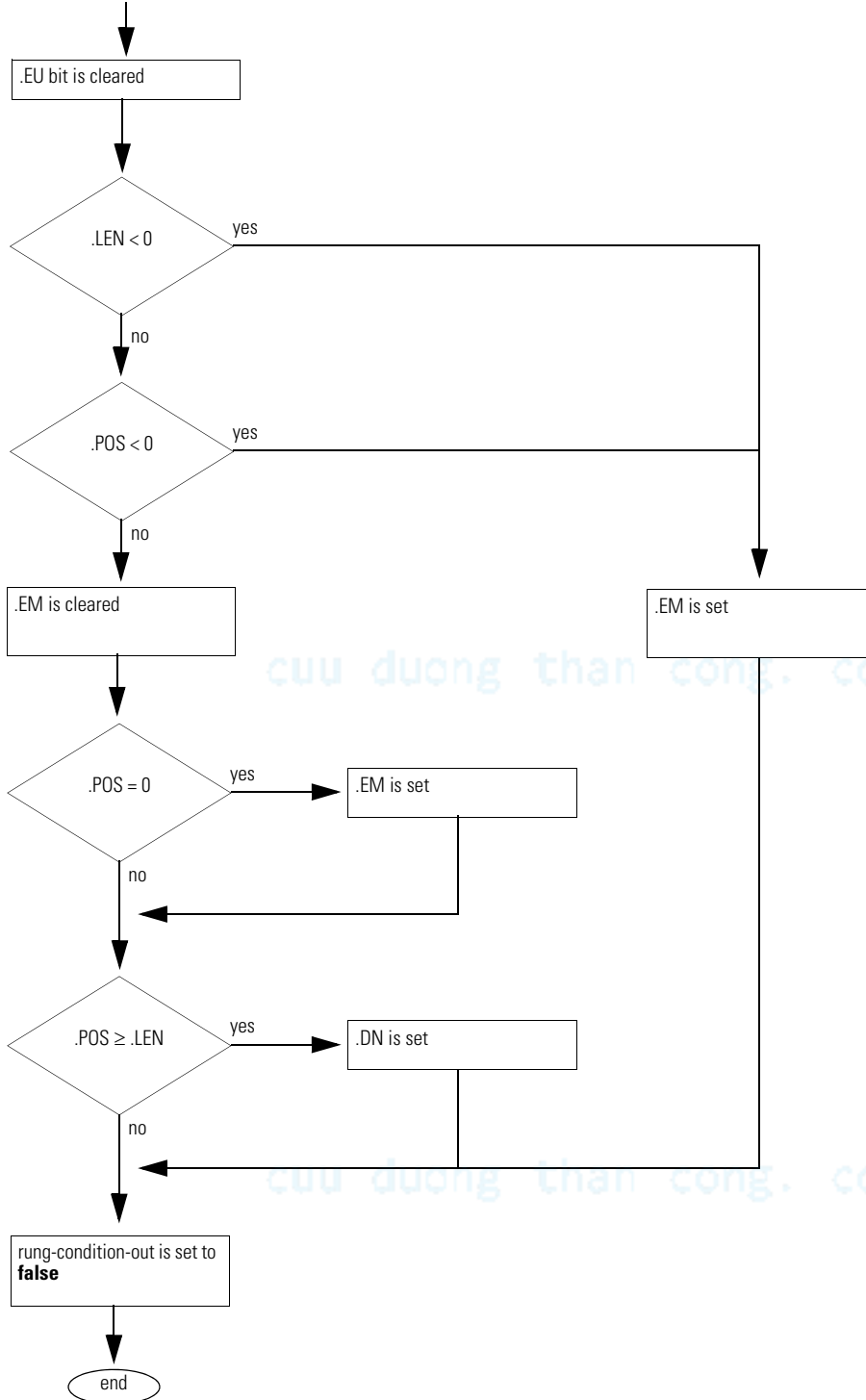
**Execution:**





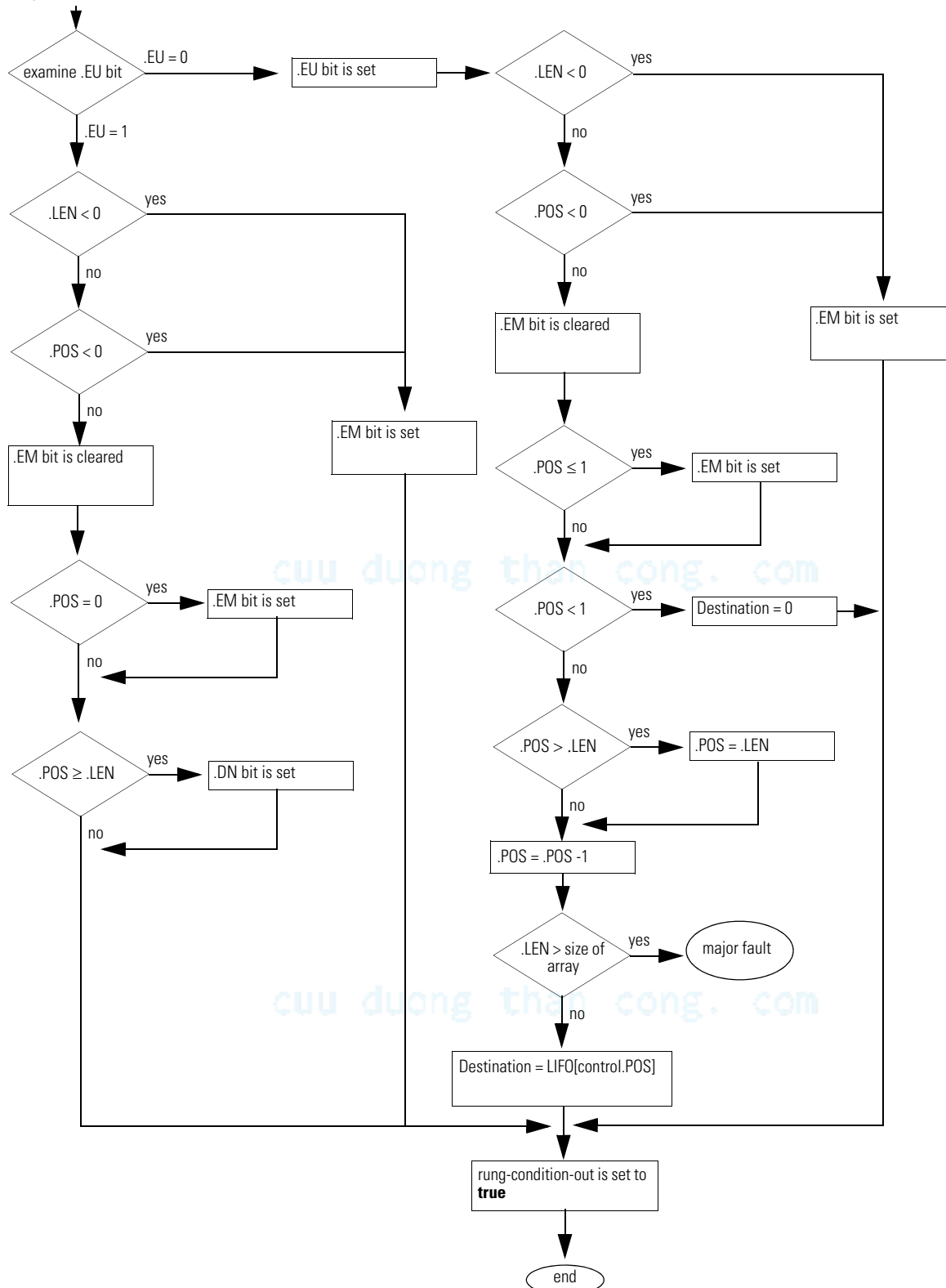
**Condition****Relay Ladder Action:**

rung-condition-in is false



**Condition****Relay Ladder Action:**

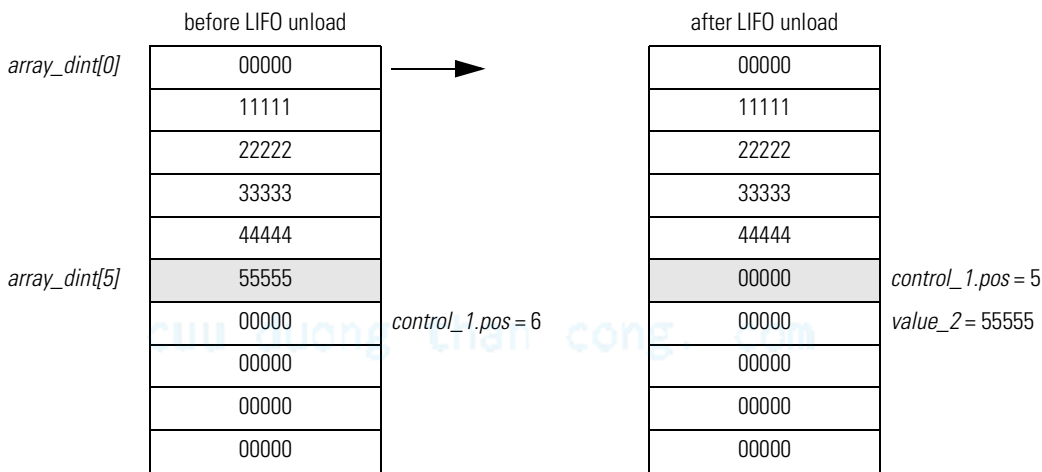
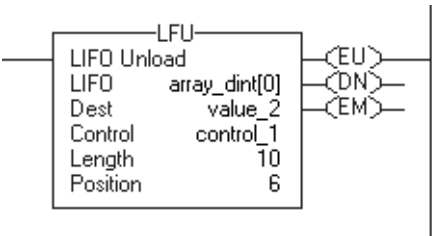
rung-condition-in is true



postscan

The rung-condition-out is set to false.

**Example:** When enabled, the LFU instruction unloads *array\_dint[5]* into *value\_2*.



## Notes:

cuu duong than cong. com

cuu duong than cong. com

## Sequencer Instructions

(SQI, SQO, SQL)

### Introduction

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

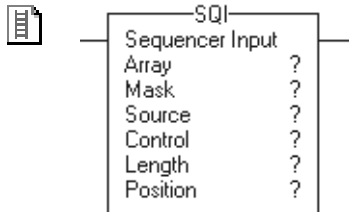
If You Want To	Use This Instruction	Available In These Languages	See Page
Detect when a step is complete.	SQI	relay ladder	422
Set output conditions for the next step.	SQO	relay ladder	426
Load reference conditions into sequencer arrays	SQL	relay ladder	430

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  <b>do not</b> use CONTROL.POS in the subscript
Mask	SINT	tag	which bits to block or pass
	INT	immediate	
<b>DINT</b> A SINT or INT tag converts to a DINT value by sign-extension.			
Source	SINT	tag	input data for the sequencer array
	INT		
	DINT		A SINT or INT tag converts to a DINT value by sign-extension.
Control	CONTROL	tag	control structure for the operation  typically use the same CONTROL as the SQO and SQL instructions
Length	DINT	immediate	number of elements in the Array (sequencer table) to compare
Position	DINT	immediate	current position in the array  initial value is typically 0

### CONTROL Structure

Mnemonic	Data Type	Description
.ER	BOOL	The error bit is set when .LEN $\leq$ 0, .POS < 0, or .POS > .LEN.
.LEN	DINT	The length specifies the number of steps in the sequencer array.
.POS	DINT	The position identifies the element that the instruction is currently comparing.

**Description:** When enabled, the SQI instruction compares a Source element through a Mask to an Array element for equality.

Typically use the same CONTROL structure as the SQO and SQL instructions.

The SQI instruction operates on contiguous memory.

### 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

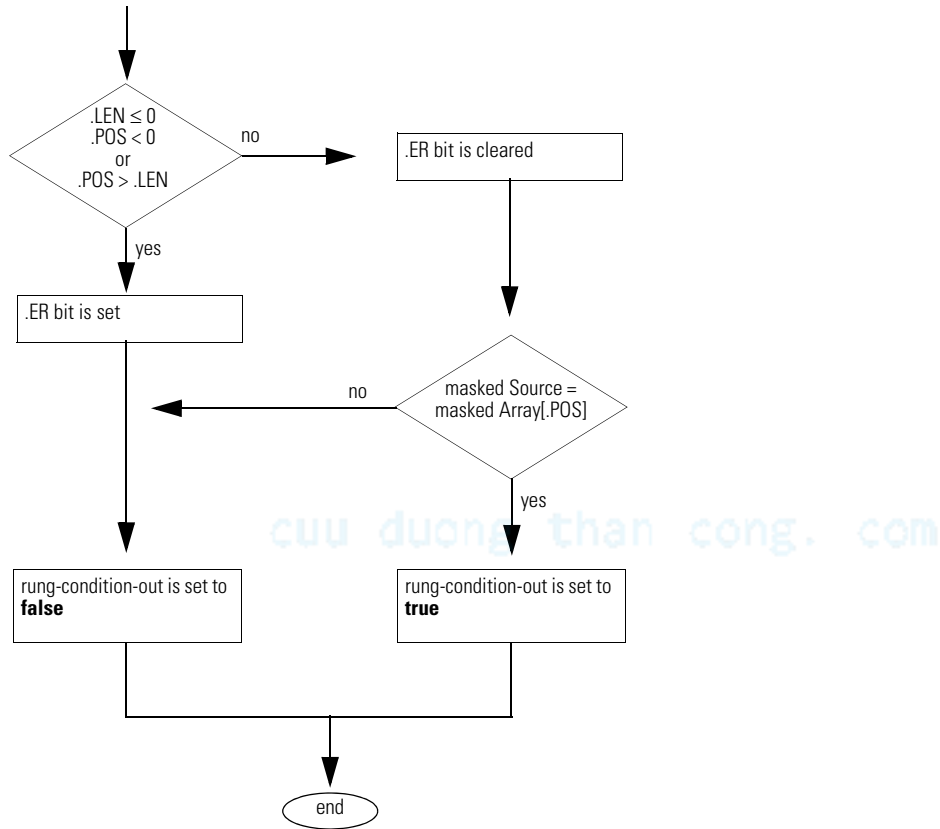
**Arithmetic Status Flags:** not affected

**Fault Conditions:** none

**Execution:**

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

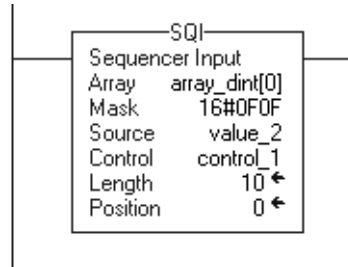
rung-condition-in is true



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.



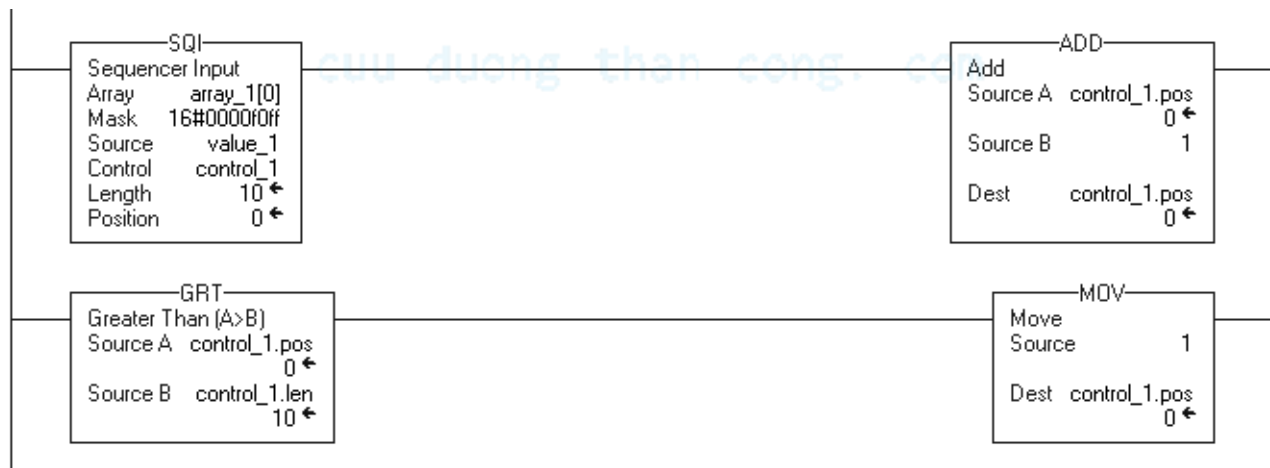
SQI Operand	Example Values (DINTs Displayed In Binary)
Source	xxxxxxxx xxxxxxxx xxxx0101 xxxx1010
Mask	00000000 00000000 00001111 00001111
Array	xxxxxxxx xxxxxxxx xxxx0101 xxxx1010

A 0 in the mask means the bit is not compared (designated by xxxx 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.

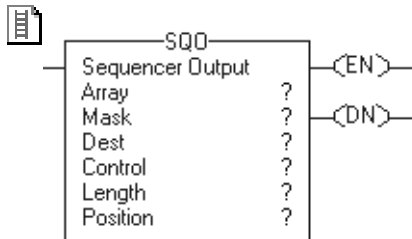
The SQI instruction compares the source value. The ADD instruction increments the sequencer array. The GRT determined whether another value is available to check in the sequencer array. The MOV instruction resets the position value after completely stepping through the sequencer array one time.



## Sequencer Output (SQO)

The SQO instruction sets output conditions for the next step of a sequence pair of SQO/SQI instructions.

### Operands:



### Relay Ladder

Operand	Type	Format	Description
Array	DINT	array tag	sequencer array
			specify the first element of the sequencer array
			<b>do not</b> use CONTROL.POS in the subscript
Mask	SINT	tag	which bits to block or pass
	INT	immediate	
	DINT		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 using another format, precede the value with the correct prefix.

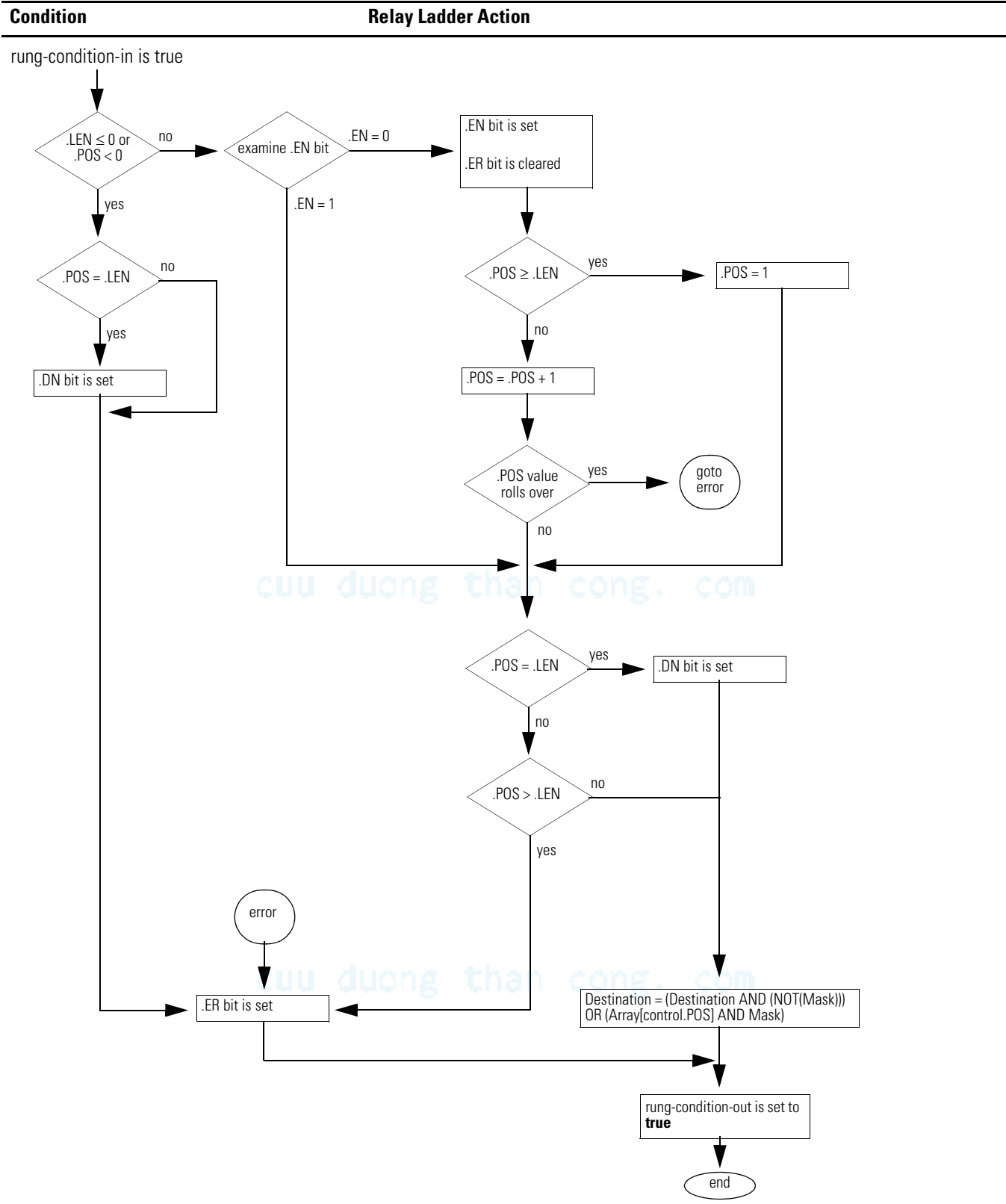
Prefix	Description
16#	hexadecimal for example; 16#0F0F
8#	octal for example; 8#16
2#	binary for example; 2#00110011

**Arithmetic Status Flags** not affected

**Fault Conditions:** none

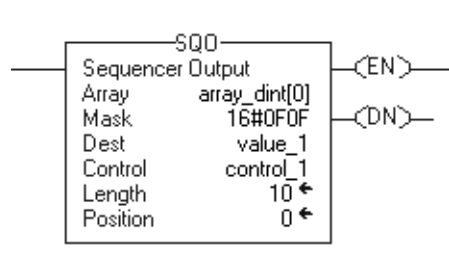
### Execution:

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).

cuu duong than cong. com

### 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,.



cuu duong than cong. com

### Resetting the position of SQO

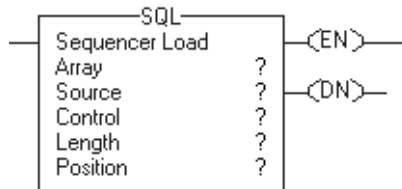
Each time the controller goes from Program to Run mode, the SQO instruction clears (initializes) the .POS value. To reset .POS to the initialization value (.POS = 0), use a RES instruction to clear the position value. This example uses the status of the first-scan bit to clear the .POS value.



## Sequencer Load (SQL)

The SQL instruction loads reference conditions into a sequencer array.

### Operands:



### Relay Ladder

Operand	Type	Format	Description
Array	DINT	array tag	sequencer array
			specify the first element of the sequencer array
			<b>do not</b> use CONTROL.POS in the subscript
Source	SINT	tag	input data to load into the sequencer array
	INT	immediate	
			<b>DINT</b>
			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  $\geq$  .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

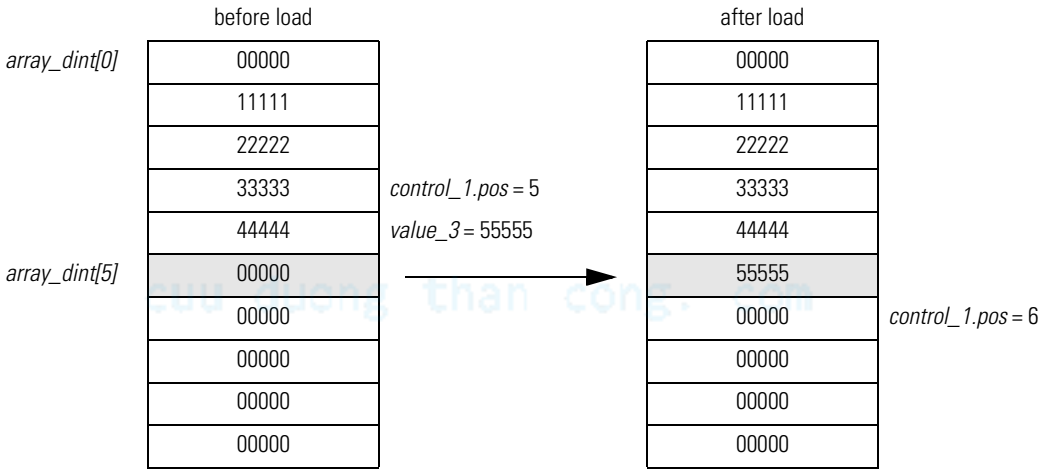
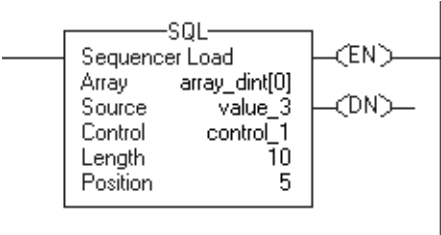
**Execution:**

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.





**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:

cuu duong than cong. com

cuu duong than cong. com

## Program Control Instructions

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

### Introduction

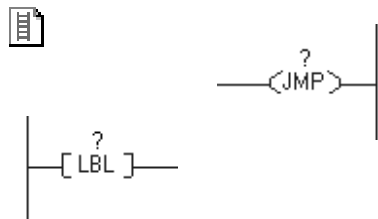
Use the program control instructions to change the flow of logic.

If You Want To	Use This Instruction	Available In These Languages	See Page
Jump over a section of logic that does not always need to be executed.	JMP LBL	relay ladder	436
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	438
Jump to an external routine (SoftLogix5800 controller only)	JXR	relay ladder	449
Mark a temporary end that halts routine execution.	TND	relay ladder structured text	452
Disable all the rungs in a section of logic.	MCR	relay ladder	454
Disable user tasks.	UID	relay ladder structured text	456
Enable user tasks.	UIE	relay ladder structured text	456
Disable a rung.	AFI	relay ladder	458
Insert a placeholder in the logic.	NOP	relay ladder	459
End a transition for a sequential function chart.	EOT	relay ladder structured text	460
Pause a sequential function chart.	SFP	relay ladder structured text	462
Reset a sequential function chart.	SFR	relay ladder structured text	464
Trigger the execution of an event task	EVENT	relay ladder structured text	466

# Jump to Label (JMP) Label (LBL)

The JMP and LBL instructions skip portions of ladder logic.

## Operands:



## Relay Ladder

Operand	Type	Format	Description
JMP instruction			
Label name		label name	enter name for associated LBL instruction
LBL instruction			
Label name		label name	execution jumps to LBL instruction with referenced label name

**Description:** When enabled, the JMP instruction skips to the referenced LBL instruction and the controller continues executing from there. When disabled, the JMP instruction does not affect ladder execution.

The JMP instruction can move ladder execution forward or backward. Jumping forward to a label saves program scan time by omitting a logic segment until it's needed. Jumping backward lets the controller repeat iterations of logic.

Be careful not to jump backward an excessive number of times. The watchdog timer could time out because the controller never reaches the end of the logic, which in turn faults the controller.

### ATTENTION



Jumped logic is not scanned. Place critical logic outside the jumped zone.

The LBL instruction is the target of the JMP instruction that has the same label name. **Make sure the LBL instruction is the first instruction on its rung.**

A label name must be unique within a routine. The name can:

- have as many as 40 characters
- contain letters, numbers, and underscores ( \_ )

**Arithmetic Status Flags:** not affected

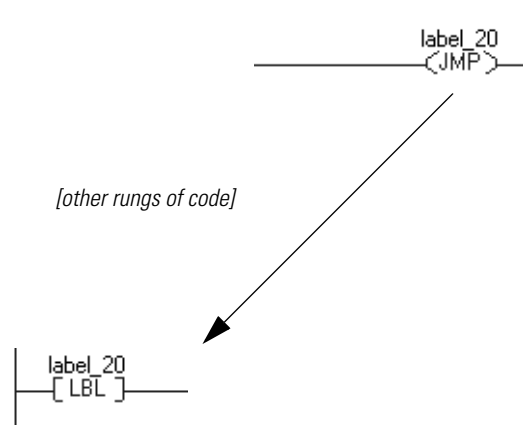
**Fault Conditions:**

A Major Fault Will Occur If	Fault Type	Fault Code
label does not exist	4	42

**Execution:**

Condition:	Relay Ladder Action
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	The rung-condition-out is set to true.  Execution jumps to the rung that contains the LBL instruction with the referenced label name.
postscan	The rung-condition-out is set to false.

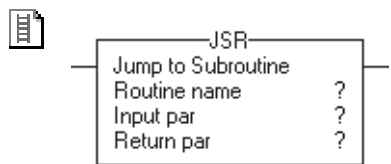
**Example:** When the JMP instruction is enabled, execution jumps over successive rungs of logic until it reaches the rung that contains the LBL instruction with *label\_20*.



# Jump to Subroutine (JSR) Subroutine (SBR) Return (RET)

The JSR instruction jumps execution to a different routine. The SBR and RET instructions are optional instructions that exchange data with the JSR instruction.

## JSR Operands:



## Relay Ladder

Operand	Type	Format	Description
Routine name	ROUTINE	name	routine to execute (that is, subroutine)
Input parameter	BOOL	immediate	data from this routine that you want to copy to a tag in the subroutine <ul style="list-style-type: none"> <li>Input parameters are optional.</li> <li>Enter multiple input parameters, if needed.</li> </ul>
	SINT	tag	
	INT	array tag	
	DINT		
	REAL		
Return parameter	structure		tag in this routine to which you want to copy a result of the subroutine <ul style="list-style-type: none"> <li>Return parameters are optional.</li> <li>Enter multiple return parameters, if needed.</li> </ul>
	BOOL	tag	
	SINT	array tag	
	INT		
	DINT		
	REAL		
	structure		



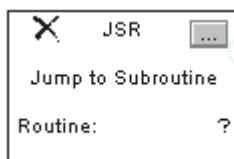
```
JSR(RoutineName, InputCount,
    InputPar, ReturnPar);
```

### Structured Text

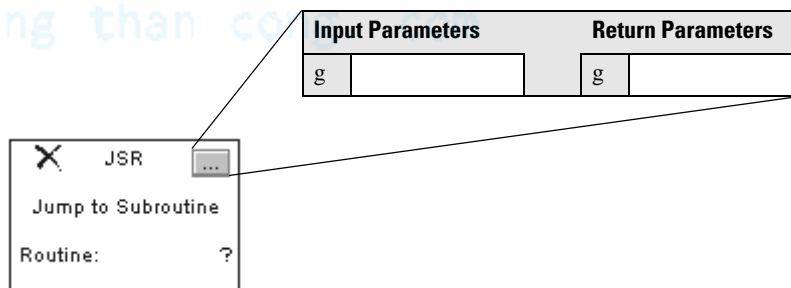
Operand	Type	Format	Description
Routine name	ROUTINE	name	routine to execute (that is, subroutine)
Input count	SINT	immediate	number of input parameters
	INT		
	DINT		
	REAL		
Input parameter	BOOL	immediate	data from this routine that you want to copy to a tag in the subroutine <ul style="list-style-type: none"> <li>Input parameters are optional.</li> <li>Enter multiple input parameters, if needed.</li> </ul>
	SINT	tag	
	INT	array tag	
	DINT		
	REAL		
	structure		
Return parameter	BOOL	tag	tag in this routine to which you want to copy a result of the subroutine <ul style="list-style-type: none"> <li>Return parameters are optional.</li> <li>Enter multiple return parameters, if needed.</li> </ul>
	SINT	array tag	
	INT		
	DINT		
	REAL		
	structure		

(JSR operands continued on next page)

### JSR Operands (continued)



### Function Block



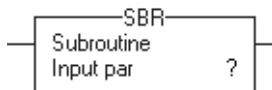
The operands are the same as those for the relay ladder JSR instruction.

**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	tag	tag in this routine into which you want to copy the corresponding input parameter from the JSR instruction
	SINT	array tag	
	INT		
	DINT		
	REAL		
	structure		

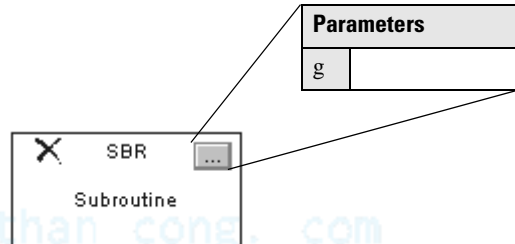
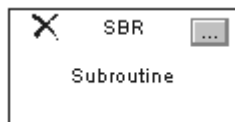


SBR ( InputPar ) ;

**Structured Text**

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

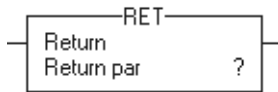
**Function Block**



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



## RET Operands:



### Relay Ladder

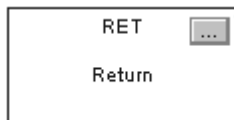
Operand	Type	Format	Description
Return parameter	BOOL	immediate	data from this routine that you want to copy to the corresponding return parameter in the JSR instruction
	SINT	tag	
	INT	array tag	
	DINT		
	REAL		
	structure		



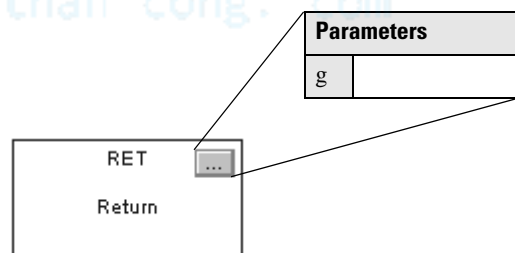
RET(ReturnPar) ;

### Structured Text

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



### Function Block



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

**Description:** The JSR instruction initiates the execution of the specified routine, which is referred to as a subroutine:

- The subroutine executes one time.
- After the subroutine executes, logic execution returns to the routine that contains the JSR instruction.

To program a jump to a subroutine, follow these guidelines:

**IMPORTANT**

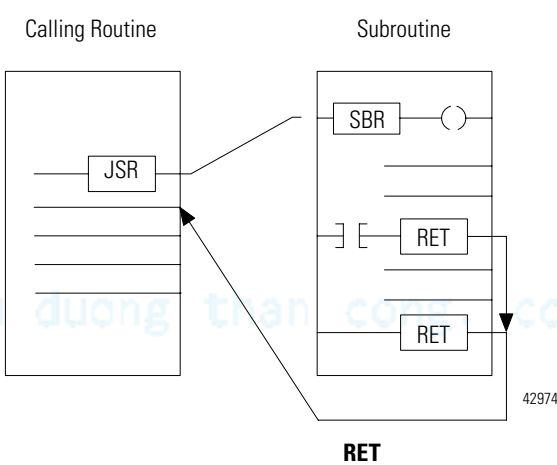
Do not use a JSR instruction to call (execute) the main routine.

- You can put a JSR instruction in the main routine or any other routine.
- If you use a JSR instruction to call the main routine and then put a RET instruction in the main routine, a major fault occurs (type 4, code 31).

The following diagram illustrates how the instructions operate.

**JSR**

1. If you want to copy data to a tag in the subroutine, enter an input parameter.
2. If you want to copy a result of the subroutine to a tag in this routine, enter a return parameter.
3. Enter as many input and return parameters as you need.



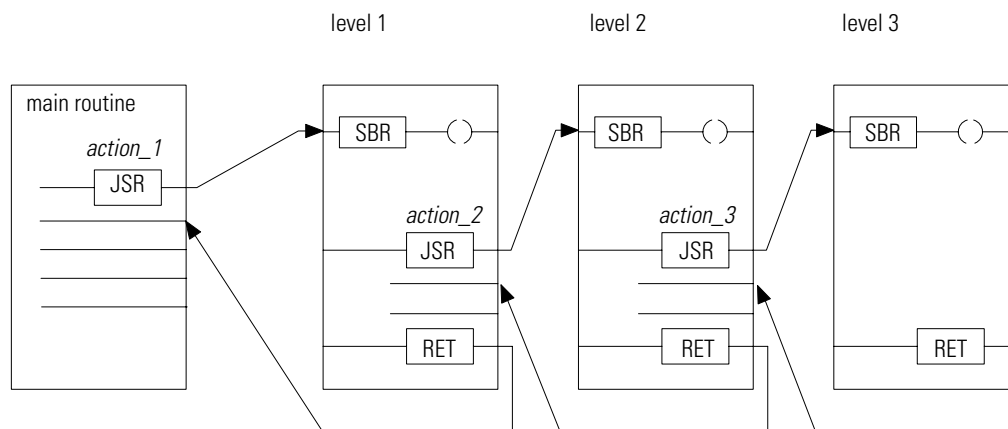
**RET**

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.

**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

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:

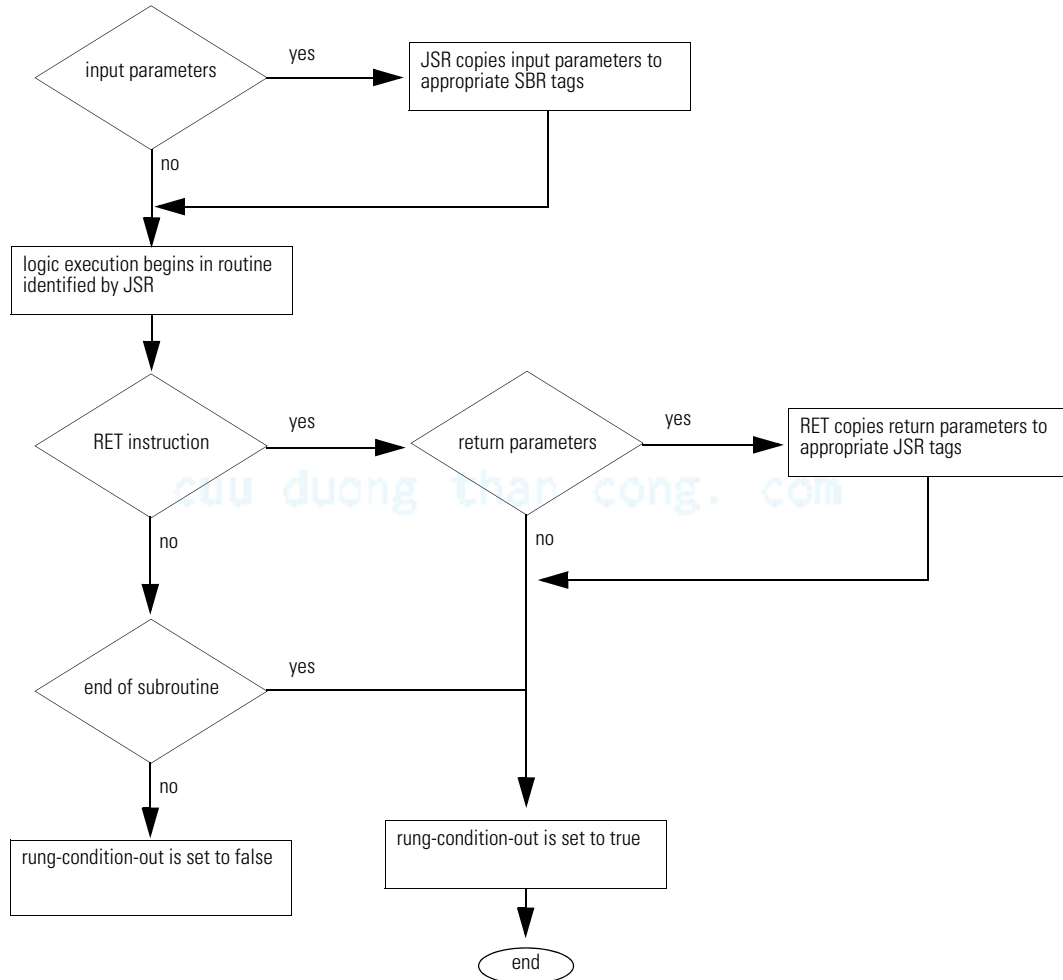
#### Relay Ladder and Structured Text



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.)  <ul style="list-style-type: none"> <li>Release 6.x and earlier, input and return parameters are passed.</li> <li>Release 7.x and later, input and return parameters are not passed.</li> </ul> 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 <i>does not</i> execute.  Outputs in the subroutine remain in their last state.  The rung-condition-out is set to false.	na

Condition	Relay Ladder Action	Structured Text Action
rung-condition-in is true	The instruction executes.  The rung-condition-out is set to true.	na
EnableIn is set	na	EnableIn is always set.  The instruction executes.

instruction execution



postscan	Same action as prescan described above.	Same action as prescan described above.
----------	---	---

### Function Block

Condition:	Action
prescan	No action taken.
instruction first scan	No action taken.
instruction first run	No action taken.
normal execution	<ol style="list-style-type: none"> <li>1. If the routine contains an SBR instruction, the controller first executes the SBR instruction.</li> <li>2. The controller latches all data values in IREFs.</li> <li>3. The controller executes the other function blocks in the order that is determined by their wiring. This includes other JSR instructions.</li> <li>4. The controller writes outputs in OREFs.</li> <li>5. If the routine contains an RET instruction, the controller executes the RET instruction last.</li> </ol>
postscan	<p>The subroutine is called.</p> <p>If the routine is an SFC routine, the routine is initialized the same as it is during prescan.</p>

cuu duong than cong. com

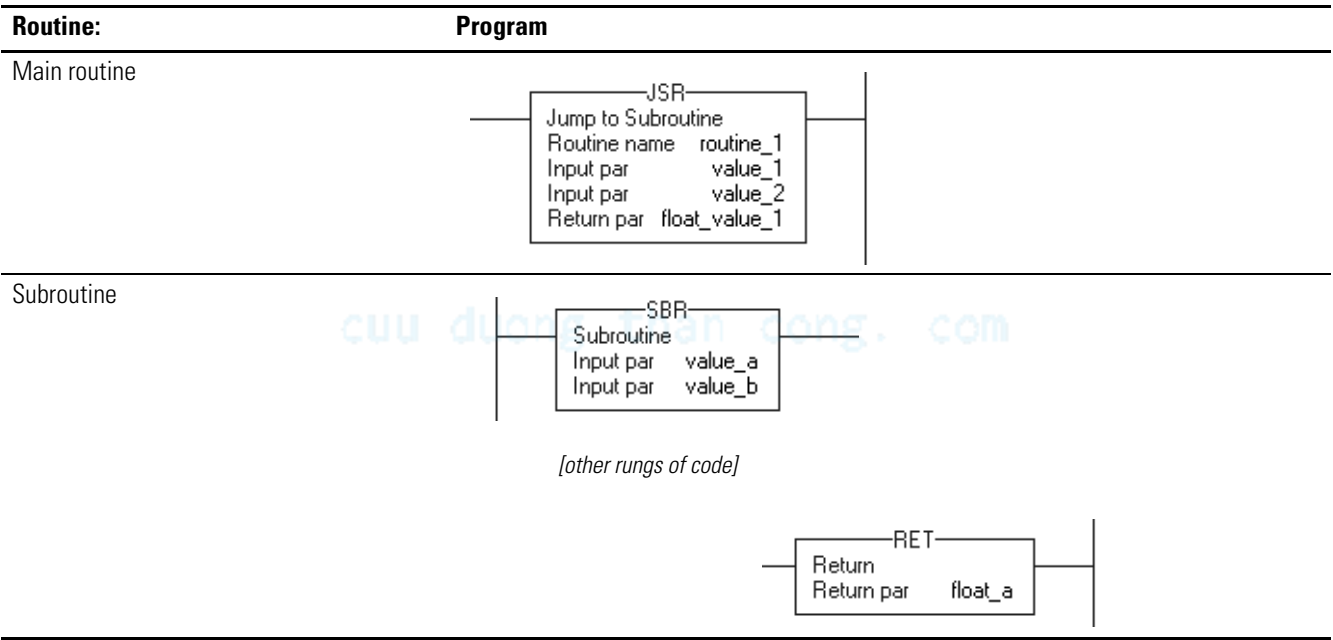
cuu duong than cong. com

**Example 1:** The JSR instruction passes *value\_1* and *value\_2* to *routine\_1*.

The SBR instruction receives *value\_1* and *value\_2* from the JSR instruction and copies those values to *value\_a* and *value\_b*, respectively. Logic execution continues in this routine.

The RET instruction sends *float\_a* to the JSR instruction. The JSR instruction receives *float\_a* and copies the value to *float\_value\_1*. Logic execution continues with the next instruction following the JSR instruction.

Relay Ladder



Structured Text

Routine	Program
Main routine	JSR(routine_1,2,value_1,value_2,float_value_1);
Subroutine	SBR(value_a,value_b);  <statements>;  RET(float_a);

**Example 2:****Relay Ladder**

## MainRoutine

When *abc* is on, *subroutine\_1* executes, calculates the number of cookies, and places a value in *cookies\_1*.

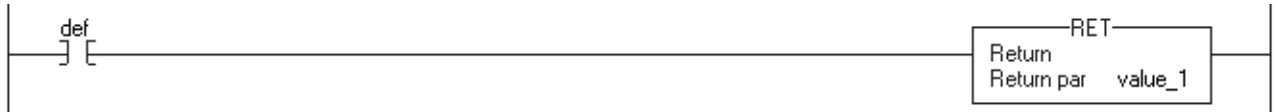


Adds the value in *cookies\_1* to *cookies\_2* and stores the result in *total\_cookies*.

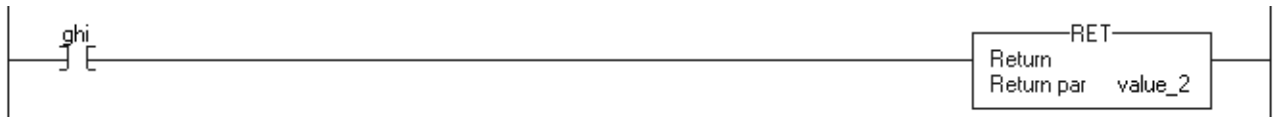


## Subroutine\_1

When *def* is on, the RET instruction returns *value\_1* to the JSR *cookies\_1* parameter and the rest of the subroutine is not scanned.



When *def* is off (previous rung) and *ghi* is on, the RET instruction returns *value\_2* to the JSR *cookies\_1* parameter and the rest of the subroutine is not scanned.

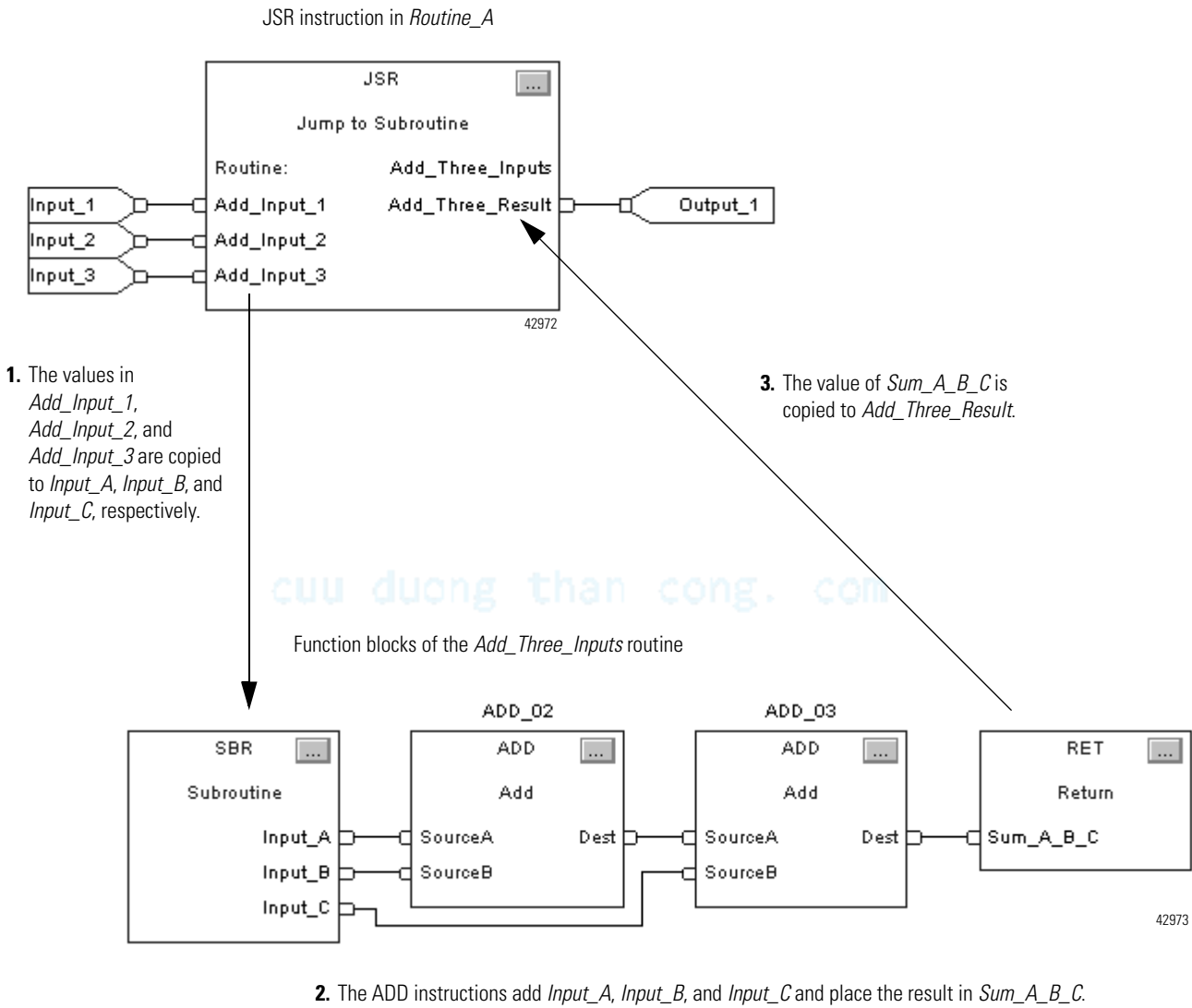


When both *def* and *ghi* are off (previous rungs), the RET instruction returns *value\_3* to the JSR *cookies\_1* parameter.



Example 3:

Function Block





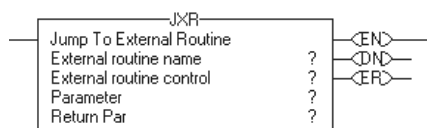
## Jump to External Routine (JXR)

The JXR instruction executes an external routine. This instruction is only supported by the SoftLogix5800 controllers.

### Operands:



### Relay Ladder



Operand	Type	Format	Description
External routine name	ROUTINE	name	external routine to execute
External routine control	EXT_ROUTINE_CONTROL	tag	control structure (see the next page)
Parameter	BOOL	immediate	data from this routine that you want to copy to a variable in the external routine
	SINT	tag	<ul style="list-style-type: none"> <li>Parameters are optional.</li> <li>Enter multiple parameters, if needed.</li> <li>You can have as many as 10 parameters.</li> </ul>
	INT	array tag	
	DINT		
	REAL		
	structure		
Return parameter	BOOL	tag	tag in this routine to which you want to copy a result of the external routine
	SINT		<ul style="list-style-type: none"> <li>The return parameter is optional.</li> <li>You can have only one return parameter</li> </ul>
	INT		
	DINT		
	REAL		

**EXT\_ROUTINE\_CONTROL Structure**

Mnemonic	Data Type	Description	Implementation
ErrorCode	SINT	If an error occurs, this value identifies the error. Valid values are from 0-255.	There are no predefined error codes. The developer of the external routine must provide the error codes.
NumParams	SINT	This value indicates the number of parameters associated with this instruction.	Display only - this information is derived from the instruction entry.
ParameterDefs	EXT_ROUTINE_PARAMETERS[10]	This array contains definitions of the parameters to pass to the external routine. The instruction can pass as many as 10 parameters.	Display only - this information is derived from the instruction entry.
ReturnParamDef	EXT_ROUTINE_PARAMETERS	This value contains definitions of the return parameter from the external routine. There is only one return parameter.	Display only - this information is derived from the instruction entry.
EN	BOOL	When set, the enable bit indicates that the JXR instruction is enabled.	The external routine sets this bit.
ReturnsValue	BOOL	If set, this bit indicates that a return parameter was entered for the instruction. If cleared, this bit indicates that no return parameter was entered for the instruction.	Display only - this information is derived from the instruction entry.
DN	BOOL	The done bit is set when the external routine has executed once to completion.	The external routine sets this bit.
ER	BOOL	The error bit is set if an error occurs. The instruction stops executing until the program clears the error bit.	The external routine sets this bit.
FirstScan	BOOL	This bit identifies whether this is the first scan after switching the controller to Run mode. Use FirstScan to initialize the external routine, if needed.	The controller sets this bit to reflect scan status.
EnableOut	BOOL	Enable output.	The external routine sets this bit.
EnableIn	BOOL	Enable input.	The controller sets this bit to reflect rung-condition-in. The instruction executes regardless of rung condition. The developer of the external routine should monitor this status and act accordingly.
User1	BOOL	These bits are available for the user. The controller does not initialize these bits.	Either the external routine or the user program can set these bits.
User0	BOOL		
ScanType1	BOOL	These bits identify the current scan type:	The controller sets these bits to reflect scan status.
ScanType0	BOOL		
Bit Values: Scan Type:			
00	Normal		
01	Pre Scan		
10	Post Scan (not applicable to relay ladder programs)		

**Description:** Use the Jump to External Routine (JXR) instruction to call the external routine from a ladder routine in your project. The JXR instruction supports multiple parameters so you can pass values between the ladder routine and the external routine.

The JXR instruction is similar to the Jump to Subroutine (JSR) instruction. The JXR instruction initiates the execution of the specified external routine:

- The external routine executes one time.
- After the external routine executes, logic execution returns to the routine that contains the JXR instruction.

**Arithmetic Status Flags:** Arithmetic status flags are not affected.

**Fault Conditions:**

A Major Fault Will Occur If	Fault Type	Fault code:
<ul style="list-style-type: none"> <li>•an exception occurs in the external routine DLL</li> <li>•the DLL could not be loaded</li> <li>•the entry point was not found in the DLL</li> </ul>	4	88

**Execution:** The JXR can be synchronous or asynchronous depending on the implementation of the DLL. The code in the DLL also determines how to respond to scan status, rung-condition-in status, and rung-condition-out status.

For more information on using the JXR instruction and creating external routines, see the SoftLogix5800 System User Manual, publication 1789-UM002.

## Temporary End (TND)

The TND instruction acts as a boundary.

### Operands:



—(TND)—

### Relay Ladder Operands

none



TND ( ) ;

### Structured Text

none

You must enter the parentheses () after the instruction mnemonic, even though there are no operands.

**Description:** When enabled, the TND instruction lets the controller execute logic only up to this instruction.

When enabled, the TND instruction acts as the end of the routine. When the controller scans a TND instruction, the controller moves to the end of the current routine. If the TND instruction is in a subroutine, control returns to the calling routine. If the TND instruction is in a main routine, control returns to the next program within the current task.

**Arithmetic Status Flags:** not affected

**Fault Conditions:** none

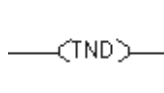
### Execution:

Condition	Relay Ladder Action	Structured Text Action
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	na
EnableIn is set	na	EnableIn is always set. The instruction executes.
instruction execution	The current routine terminates.	The current routine terminates.
postscan	The rung-condition-out is set to false.	No action taken.

**Example:** You can use the TND instruction when debugging or troubleshooting to execute logic up to a certain point. Progressively move the TND instruction through the logic as you debug each new section.

When the TND instruction is enabled, the controller stops scanning the current routine.

### Relay Ladder



### Structured Text

```
TND ( ) ;
```

cuu duong than cong. com

cuu duong than cong. com

**Master Control Reset (MCR)** The MCR instruction, used in pairs, creates a program zone that can disable all rungs within the MCR instructions.

**Operands:**



—(MCR)—

**Relay Ladder**

none

**Description:** When the MCR zone is enabled, the rungs in the MCR zone are scanned for normal true or false conditions. When disabled, the controller still scans rungs within an MCR zone, but scan time is reduced because non-retentive outputs in the zone are disabled. The rung-condition-in is false for all the instructions inside of the disabled MCR zone.

When you program an MCR zone, note that:

- You must end the zone with an unconditional MCR instruction.
- You cannot nest one MCR zone within another.
- Do not jump into an MCR zone. If the zone is false, jumping into the zone activates the zone from the point to which you jumped to the end of the zone.
- If an MCR zone continues to the end of the routine, you do not have to program an MCR instruction to end the zone.

The MCR instruction is not a substitute for a hard-wired master control relay that provides emergency-stop capability. You should still install a hard-wired master control relay to provide emergency I/O power shutdown.

**ATTENTION**



Do not overlap or nest MCR zones. Each MCR zone must be separate and complete. If they overlap or nest, unpredictable machine operation could occur with possible damage to equipment or injury to personnel.

Place critical operations outside the MCR zone. If you start instructions such as timers in a MCR zone, instruction execution stops when the zone is disabled and the timer is cleared.

**Arithmetic Status Flags:** not affected

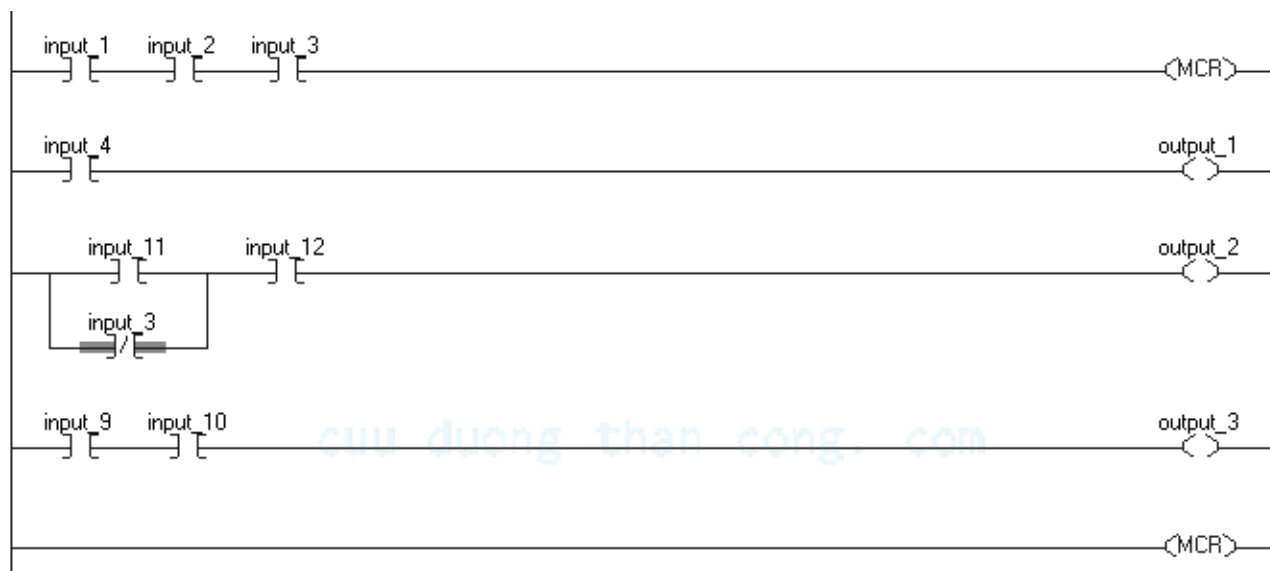
**Fault Conditions:** none

**Execution:**

Condition	Relay Ladder Action
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.  The instructions in the zone are scanned, but the rung-condition-in is false and non-retentive outputs in the zone are disabled.
rung-condition-in is true	The rung-condition-out is set to true.  The instructions in the zone are scanned normally.
postscan	The rung-condition-out is set to false.

**Example:** When the first MCR instruction is enabled (*input\_1*, *input\_2*, and *input\_3* are set), the controller executes the rungs in the MCR zone (between the two MCR instructions) and sets or clears outputs, depending on input conditions.

When the first MCR instruction is disabled (*input\_1*, *input\_2*, and *input\_3* are not all set), the controller executes the rungs in the MCR zone (between the two MCR instructions) and the rung-condition-in goes false for all the rungs in the MCR zone, regardless of input conditions.



## User Interrupt Disable (UID) User Interrupt Enable (UIE)

The UID instruction and the UIE instruction work together to prevent a small number of critical rungs from being interrupted by other tasks.

### Operands:



—UID— —UIE—

### Relay Ladder

none



```
UID ( ) ;  
UIE ( ) ;
```

### Structured Text

none

You must enter the parentheses () after the instruction mnemonic, even though there are no operands.

**Description:** When the rung-condition-in is true, the:

- UID instruction prevents higher-priority tasks from interrupting the current task but *does not* disable execution of a fault routine or the Controller Fault Handler.
- UIE instruction enables other tasks to interrupt the current task.

To prevent a series of rungs from being interrupted:

1. Limit the number of rungs that you *do not* want interrupted to as few as possible. Disabling interrupts for a prolonged period of time can produce communication loss.
2. Above the first rung that you *do not* want interrupted, enter a rung and a UID instruction.
3. After the last rung in the series that you *do not* want interrupted, enter a rung and a UIE instruction.
4. If required, you can nest pairs of UID/UIE instructions.

**Arithmetic Status Flags:** not affected

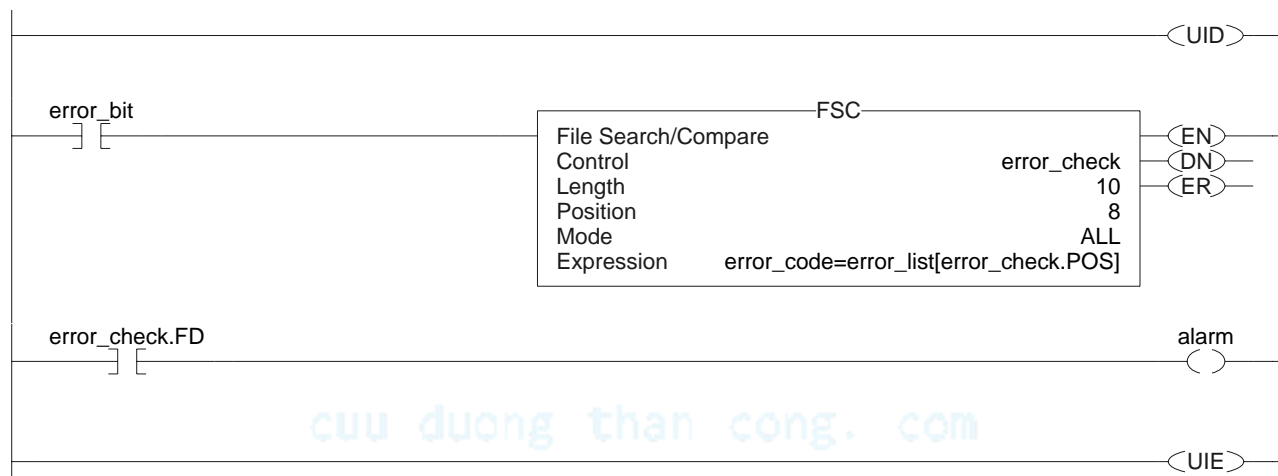
**Fault Conditions:** none



**Execution:**

Condition	Relay Ladder Action	Structured Text Action
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes.  The rung-condition-out is set to true.	na
EnableIn is set	na	EnableIn is always set.  The instruction executes.
instruction execution	The UID instruction prevents interruption by higher-priority tasks.  The UIE instruction enables interruption by higher-priority tasks.	
postscan	The rung-condition-out is set to false.	No action taken.

**Example:** When an error occurs (*error\_bit* is on), the FSC instruction checks the error code against a list of critical errors. If the FSC instruction finds that the error is critical (*error\_check.FD* is on), an alarm is annunciated. The UID and UIE instructions prevent any other tasks from interrupting the error checking and alarming.

**Relay Ladder****Structured Text**

```

UID( ) ;

<statements>

UIE( ) ;
  
```

## Always False Instruction (AFI)

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

### Operands:



—[ AFI ]—

### Relay Ladder

none

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

**Arithmetic Status Flags:** not affected

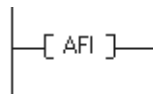
**Fault Conditions:** none

### Execution:

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

**Example:** Use the AFI instruction to temporarily disable a rung while you are debugging a program.

When enabled, the AFI disables all the instructions on this rung.



## No Operation (NOP)

The NOP instruction functions as a placeholder

### Operands:



—[NOP]—

### Relay Ladder

none

**Description:** You can place the NOP instruction anywhere on a rung. When enabled the NOP instruction performs no operation. When disabled, the NOP instruction performs no operation.

**Arithmetic Status Flags:** not affected

**Fault Conditions:** none

### Execution:

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

**Example** This instruction is useful for locating unconditional branches when you place the NOP instruction on the branch.

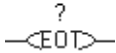
The NOP instruction bypasses the XIC instruction to enable the output.



## End of Transition (EOT)

The EOT instruction returns a boolean state to an SFC transition.

### Operands:



### Relay Ladder

Operand	Type	Format	Description
data bit	BOOL	tag	state of the transition (0=executing, 1=completed)



EOT(data\_bit);

### Structured Text

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

**Description:** Because the EOT instruction returns a boolean state, multiple SFC routines can share the same routine that contains the EOT instruction. If the calling routine is not a transition, the EOT instruction acts as a TND instruction (see page 452).

The Logix implementation of the EOT instruction differs from that in a PLC-5 controller. In a PLC-5 controller, the EOT instruction has no parameters. Instead, the PLC-5 EOT instruction returns rung condition as its state. In a Logix controller, the return parameter returns the transition state since rung condition is not available in all Logix programming languages.

**Arithmetic Status Flags:** not affected

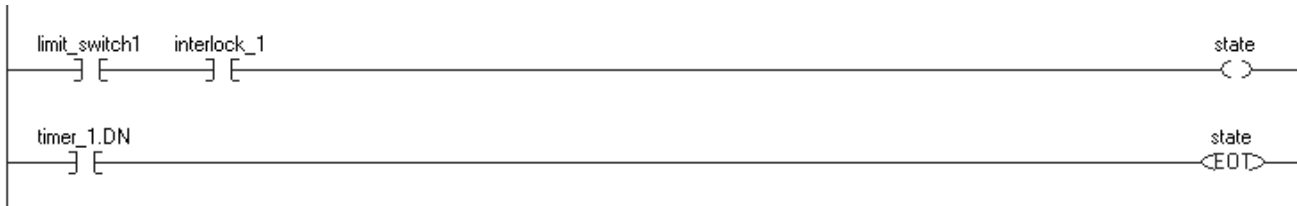
**Fault Conditions:** none

### Execution:

Condition	Relay Ladder Action:	Structured Text Action:
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes.	na
	The rung-condition-out is set to true.	
EnableIn is set	na	EnableIn is always set.
		The instruction executes.
instruction execution	The instruction returns the data bit value to the calling routine.	
postscan	The rung-condition-out is set to false.	No action taken.

**Example:** When both *limit\_switch1* and *interlock\_1* are set, set state. After *timer\_1* completes, EOT returns the value of *state* to the calling routine.

### Relay Ladder



### Structured Text

```
state := limit_switch1 AND interlock_1;
```

```
IF timer_1.DN THEN
```

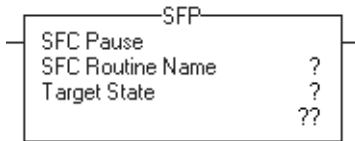
```
    EOT(state);
```

```
END_IF;
```

# SFC Pause (SFP)

The SFP instruction pauses an SFC routine.

## Operands:



## Relay Ladder

Operand	Type:	Format:	Description:
SFCRoutine Name	ROUTINE	name	SFC routine to pause
TargetState	DINT	immediate	select one:
		tag	executing (or enter 0)
			paused (or enter 1)



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

## Structured Text

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

**Description:** The SFP instruction lets you pause an executing SFC routine. If an SFC routine is in the paused state, use the SFP instruction again to change the state and resume execution of the routine.

Also, use the SFP instruction to resume SFC execution after using an SFR instruction (see page 17-464) to reset an SFC routine.

**Arithmetic Status Flags:** not affected

## Fault Conditions:

A Major Fault Will Occur If:	Fault Type	Fault Code
the routine type is not an SFC routine	4	85

cuu duong than cong. com

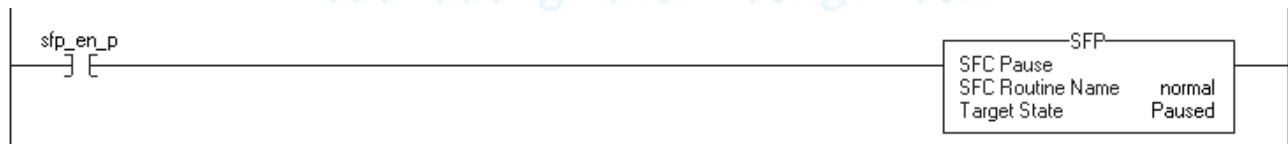
**Execution:**

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

**Example:** If *sfc\_en\_p* is set, pause the SFC routine named *normal*. Restart the SFC when *sfc\_en\_e* is set.

**Relay Ladder**

Pause the SFC routine.



Resume executing the SFC routine.

**Structured Text**

```

Pause the SFC routine: IF (sfp_en_p) THEN

    SFP(normal,paused);

    sfp_en_p := 0;

END_IF;

```

```
Resume executing the SFC routine: IF (sfp_en_e) THEN

    SFP(normal,executing);

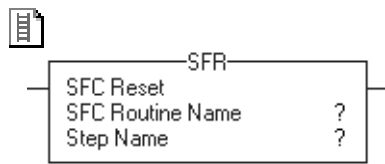
    sfp_en_e := 0;

END_IF;
```

SFC Reset (SFR)

The SFR instruction resets the execution of a SFC routine at a specified step.

Operands:



Relay Ladder Operands

Operand	Type	Format	Description
SFCRoutine Name	ROUTINE	name	SFC routine to reset
Step Name	SFC_STEP	tag	target step where to resume execution



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

Structured Text

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

**Description:** When the SFR instruction is enabled:

- In the specified SFC routine, all stored actions stop executing (reset).
- The SFC begins executing at the specified step.

If the target step is 0, the chart will be reset to its initial step

The Logix implementation of the SFR instruction differs from that in a PLC-5 controller. In the PLC-5 controller, the SFR executed when the rung condition was true. After reset, the SFC would remain paused until the rung containing the SFR became false. This allowed the execution following a reset to be delayed. This pause/un-pause feature of the PLC-5 SFR instruction was decoupled from the rung condition and moved into the SFP instruction.

**Arithmetic Status Flags:** not affected



**Fault Conditions:**

A Major Fault Will Occur If:	Fault Type	Fault Code
the routine type is not an SFC routine	4	85
specified target step does not exist in the SFC routine	4	89

**Execution:**

Condition	Relay Ladder Action	Structured Text Action
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	na
EnableIn is set	na	EnableIn is always set. The instruction executes.
instruction execution	The instruction resets the specified SFC routine.	The instruction resets the specified SFC routine.
postscan	The rung-condition-out is set to false.	No action taken.

**Example:** If a specific condition occurs (*shutdown* is set), restart the SFC at step *initialize*.

**Relay Ladder****Structured Text**

```

IF shutdown THEN

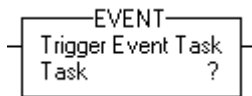
    SFR(mySFC, initialize);

END_IF;

```

## Trigger Event Task (EVENT) The EVENT instruction triggers one execution of an event task.

### Operands:



### Relay Ladder

Operand	Type	Format	Description
Task	TASK	name	event task to execute
The instruction lets you choose other types of tasks, but it does not execute them.			



```
EVENT ( task_name ) ;
```

### Structured Text

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

**Description:** Use the EVENT instruction to programmatically execute an event task:

- Each time the instruction executes, it triggers the specified event task.
- Make sure that you give the event task enough time to complete its execution before you trigger it again. If not, an overlap occurs.
- If you execute an EVENT instruction while the event task is already executing, the controller increments the overlap counter but it does not trigger the event task.

## Programmatically Determine if an EVENT Instruction Triggered a Task

To determine if an EVENT instruction triggered an event task, use a Get System Value (GSV) instruction to monitor the Status attribute of the task.

### Status Attribute of the TASK Object

Attribute	Data Type	Instruction	Description								
Status	DINT	GSV	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.								
		SSV									
			<table><tr><th>To determine if:</th><th>Examine this bit:</th></tr><tr><td>An EVENT instruction triggered the task (event task only).</td><td>0</td></tr><tr><td>A timeout triggered the task (event task only).</td><td>1</td></tr><tr><td>An overlap occurred for this task.</td><td>2</td></tr></table>	To determine if:	Examine this bit:	An EVENT instruction triggered the task (event task only).	0	A timeout triggered the task (event task only).	1	An overlap occurred for this task.	2
		To determine if:	Examine this bit:								
		An EVENT instruction triggered the task (event task only).	0								
A timeout triggered the task (event task only).	1										
An overlap occurred for this task.	2										

The controller does not clear the bits of the Status attribute once they are set.

- To use a bit for new status information, you must manually clear the bit.
- Use a Set System Value (SSV) instruction to set the attribute to a different value.

**Arithmetic Status Flags:** not affected

**Fault Conditions:** none

**Execution:**

Condition:	Relay Ladder Action	Structured Text Action
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	na
EnableIn is set	na	EnableIn is always set. The instruction executes.
instruction execution	The instruction triggers one execution of the specified event task	
postscan	The rung-condition-out is set to false.	No action taken.

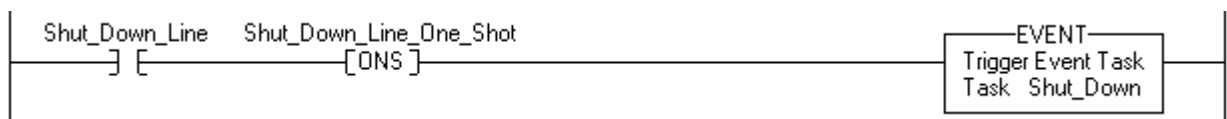
**Example 1:** A controller uses multiple programs but a common shut down procedure. Each program uses a program-scoped tag named *Shut\_Down\_Line* that turns on if the program detects a condition that requires a shut down. The logic in each program executes as follows:

If *Shut\_Down\_Line* = on (conditions require a shut down) then

Execute the *Shut\_Down* task one time

## Relay Ladder

### Program A



### Program B



## Structured Text

### Program A

```
IF Shut_Down_Line AND NOT Shut_Down_Line_One_Shot THEN
    EVENT (Shut_Down);
END_IF;
Shut_Down_Line_One_Shot := Shut_Down_Line;
```

### Program B

```
IF Shut_Down_Line AND NOT Shut_Down_Line_One_Shot THEN
    EVENT (Shut_Down);
END_IF;
Shut_Down_Line_One_Shot := Shut_Down_Line;
```

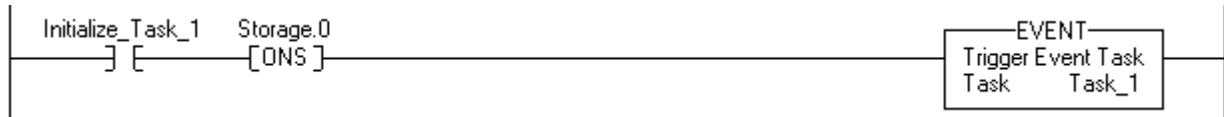
**Example 2:** The following example uses an EVENT instruction to initialize an event task. (Another type of event normally triggers the event task.)

### Continuous task

If *Initialize\_Task\_1* = 1 then

The ONS instruction limits the execution of the EVENT instruction to one scan.

The EVENT instruction triggers an execution of *Task\_1* (event task).



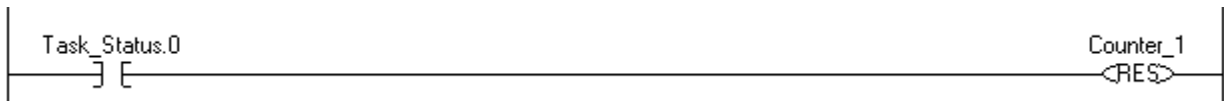
### Task\_1 (event task)

The GSV instruction sets *Task\_Status* (DINT tag) = Status attribute for the event task. In the Instance Name attribute, THIS means the TASK object for the task that the instruction is in (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.

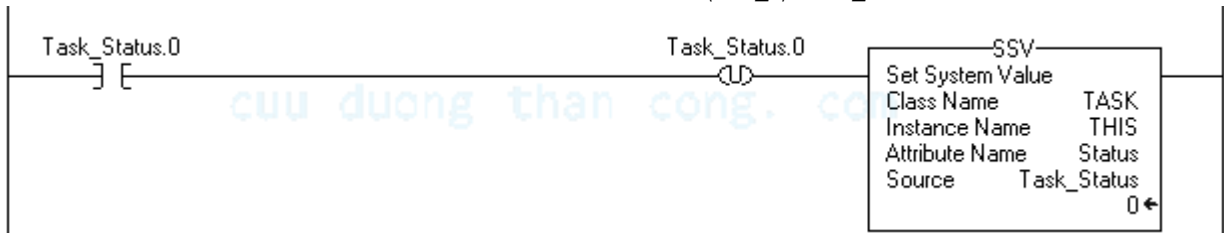


The controller does not clear the bits of the Status attribute once they are set. To use a bit for new status information, you must manually clear the bit.

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:

cuu duong than cong. com

cuu duong than cong. com

## For/Break Instructions

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

### Introduction

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

If You Want To	Use This Instruction	Available In These Languages	See Page
Repeatedly execute a routine.	FOR	relay ladder	472
	FOR...DO <sup>(1)</sup>	structured text	
Terminate the repeated execution of a routine.	BRK	relay ladder	475
	EXIT <sup>(1)</sup>	structured text	
Return to the FOR instruction.	RET	relay ladder	476

<sup>(1)</sup> Structured text only.

cuu duong than cong. com

cuu duong than cong. com

## For (FOR)

The FOR instruction executes a routine repeatedly.

### Operands:



FOR	
For	
Routine name	?
Index	?
	??
Initial value	?
Terminal value	?
Step size	?

### Relay Ladder

Operand	Type	Format	Description
Routine name	ROUTINE	routine name	routine to execute
Index	DINT	tag	counts how many times the routine has been executed
Initial value	SINT	immediate	value at which to start the index
	INT	tag	
	DINT		
Terminal value	SINT	immediate	value at which to stop executing the routine
	INT	tag	
	DINT		
Step size	SINT	immediate	amount to add to the index each time the FOR instruction executes the routine
	INT	tag	
	DINT		



```
FOR count:= initial_value TO
final_value BY increment DO
    <statement>;
END_FOR;
```

### Structured Text

Use the FOR...DO construct. See Appendix B 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.

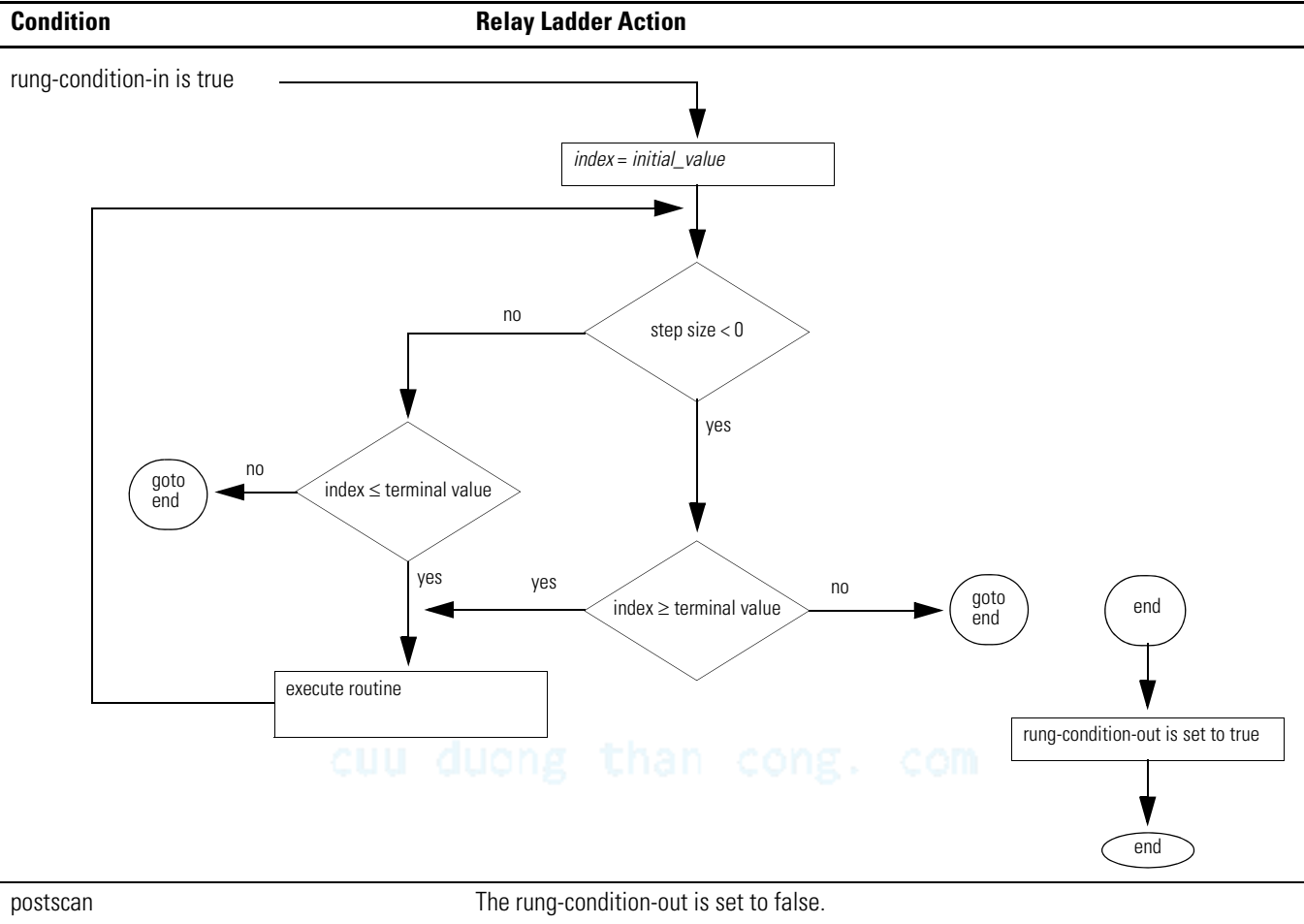
**Arithmetic Status Flags:** not affected

**Fault Conditions:**

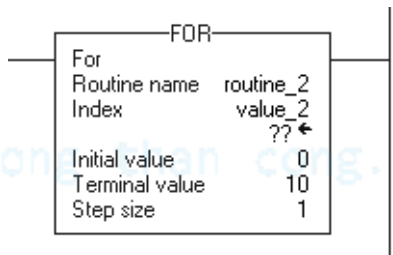
A Major Fault Will Occur If	Fault Type	Fault Code
main routine contains a RET instruction	4	31

**Execution:**

Condition	Relay Ladder Action
prescan	<p>The rung-condition-out is set to false.</p> <p>The controller executes the subroutine once.</p> <p>If recursive FOR instructions 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.</p>
rung-condition-in is false	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:



—(BRK)—

### 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.

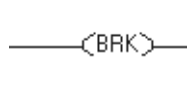
**Arithmetic Status Flags:** not affected

**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.

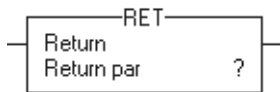
**Example:** When enabled, the BRK instruction stops executing the current routine and returns to the instruction that follows the calling FOR instruction.



## 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:

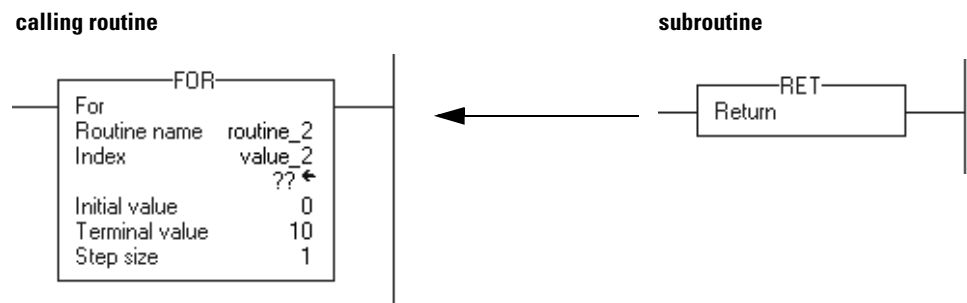
A Major Fault Will Occur If	Fault Type	Fault Code
main routine contains a RET instruction	4	31

### 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.



cuu duong than cong. com

cuu duong than cong. com

## Notes:

cuu duong than cong. com

cuu duong than cong. com

## Special Instructions

### (FBC, DDT, DTR, PID)

#### Introduction

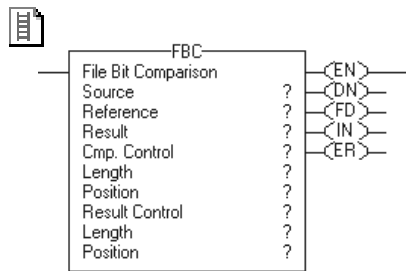
The special instructions perform application-specific operations.

If You Want To	Use This Instruction	Available In These Languages	See Page
Compare data against a known, good reference and record any mismatches.	FBC	relay ladder	480
Compare data against a known, good reference, record any mismatches, and update the reference to match the source.	DDT	relay ladder	488
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	496
Control a PID loop.	PID	relay ladder structured text	499

# 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  <b>do not</b> use CONTROL.POS in the subscript
Reference	DINT	array tag	array to compare to the source  <b>do not</b> use CONTROL.POS in the subscript
Result	DINT	array tag	array to store the result  <b>do not</b> 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.

## Selecting the Search Mode

If You Want To Detect	Select This Mode
One mismatch at a time	Set the .IN bit in the compare CONTROL structure.  Each time the rung-condition-in goes from false to true, the FBC instruction searches for the next mismatch between the Source and Reference arrays. Upon finding a mismatch, the instruction sets the .FD bit, records the position of the mismatch, and stops executing.
All mismatches	Clear the .IN bit in the compare CONTROL structure.  Each time the rung-condition-in goes from false to true, the FSC instruction searches for all mismatches between the Source and Reference arrays.

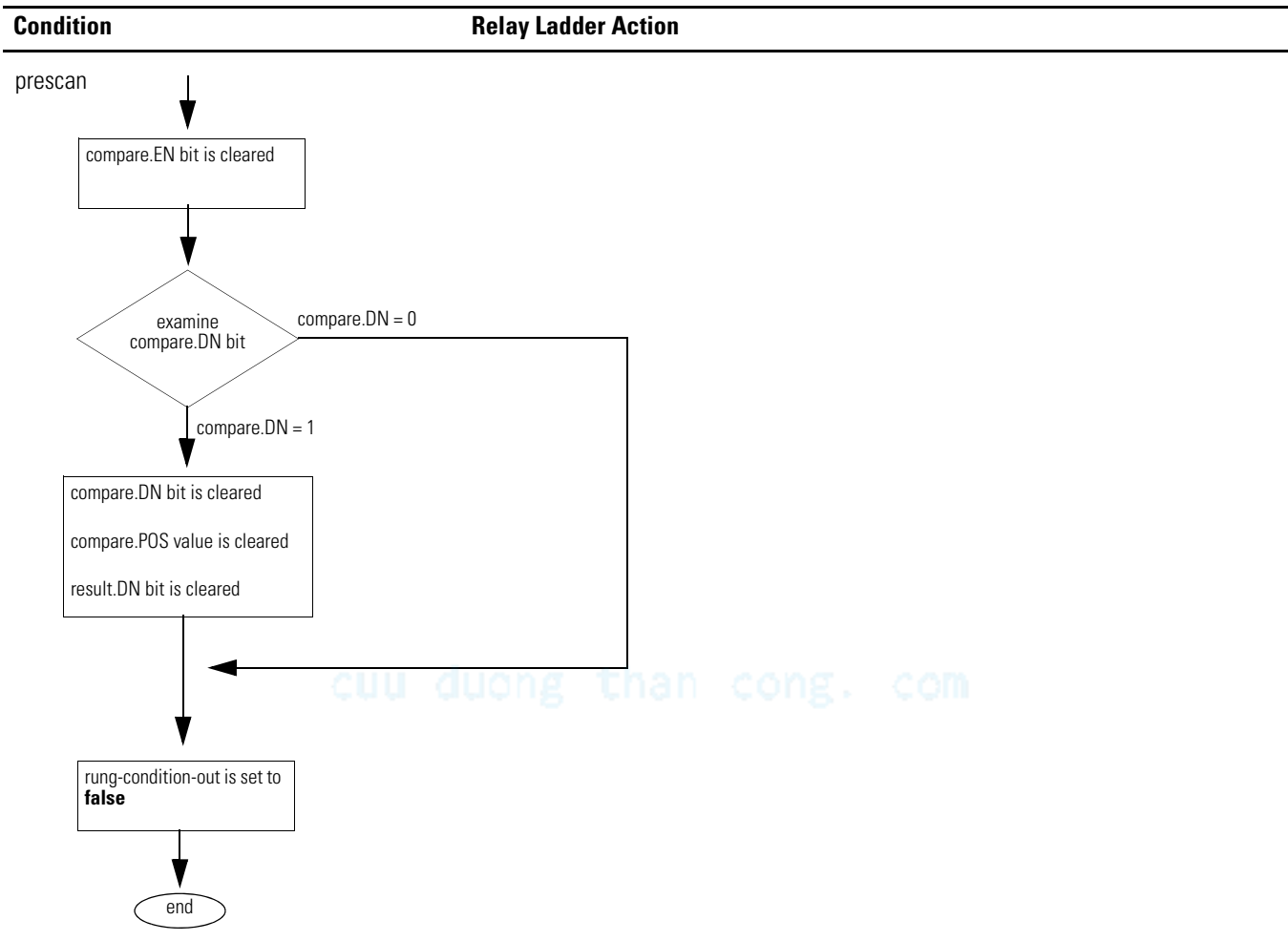
**Arithmetic Status Flags:** not affected

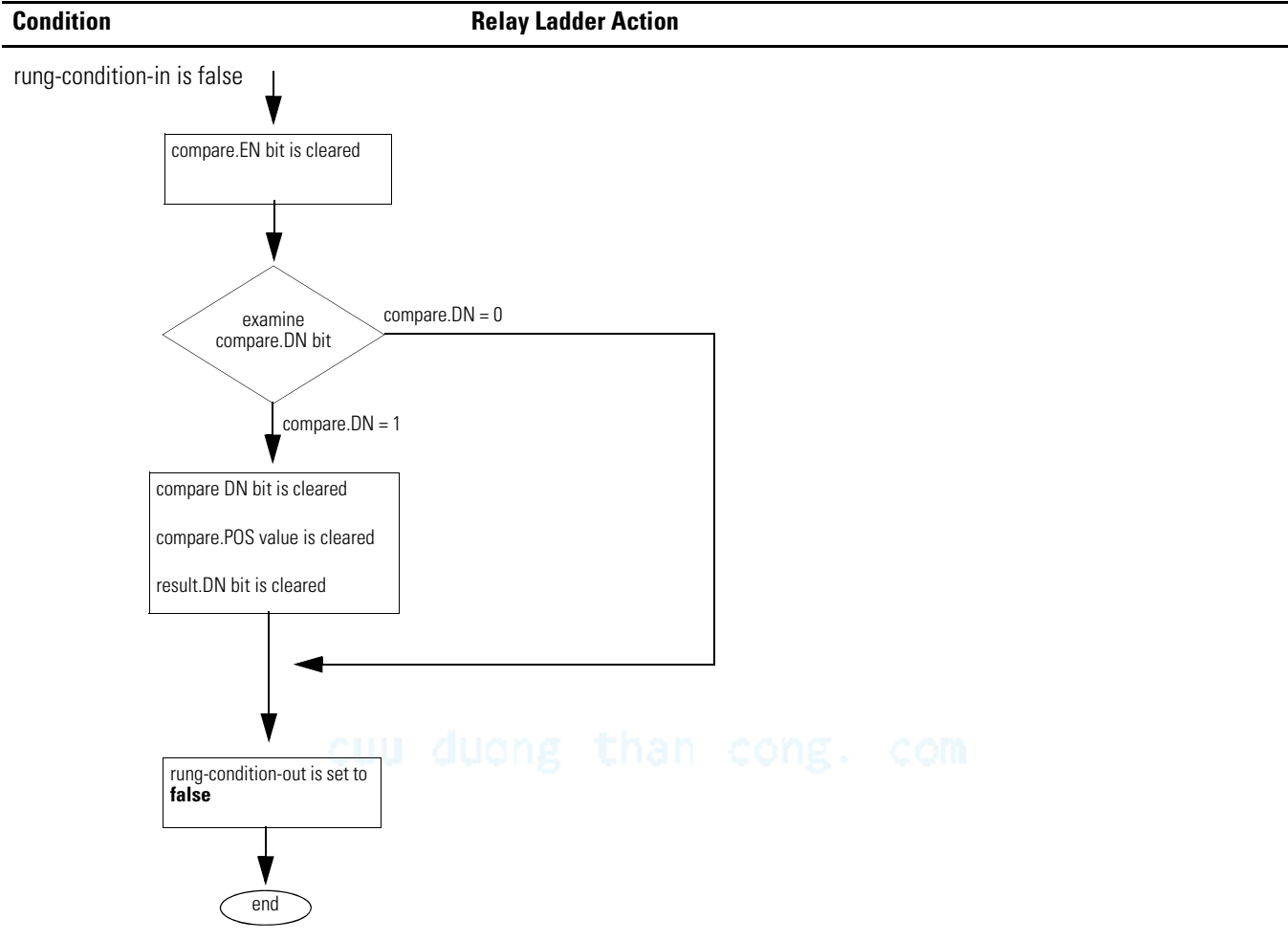
### Fault Conditions:

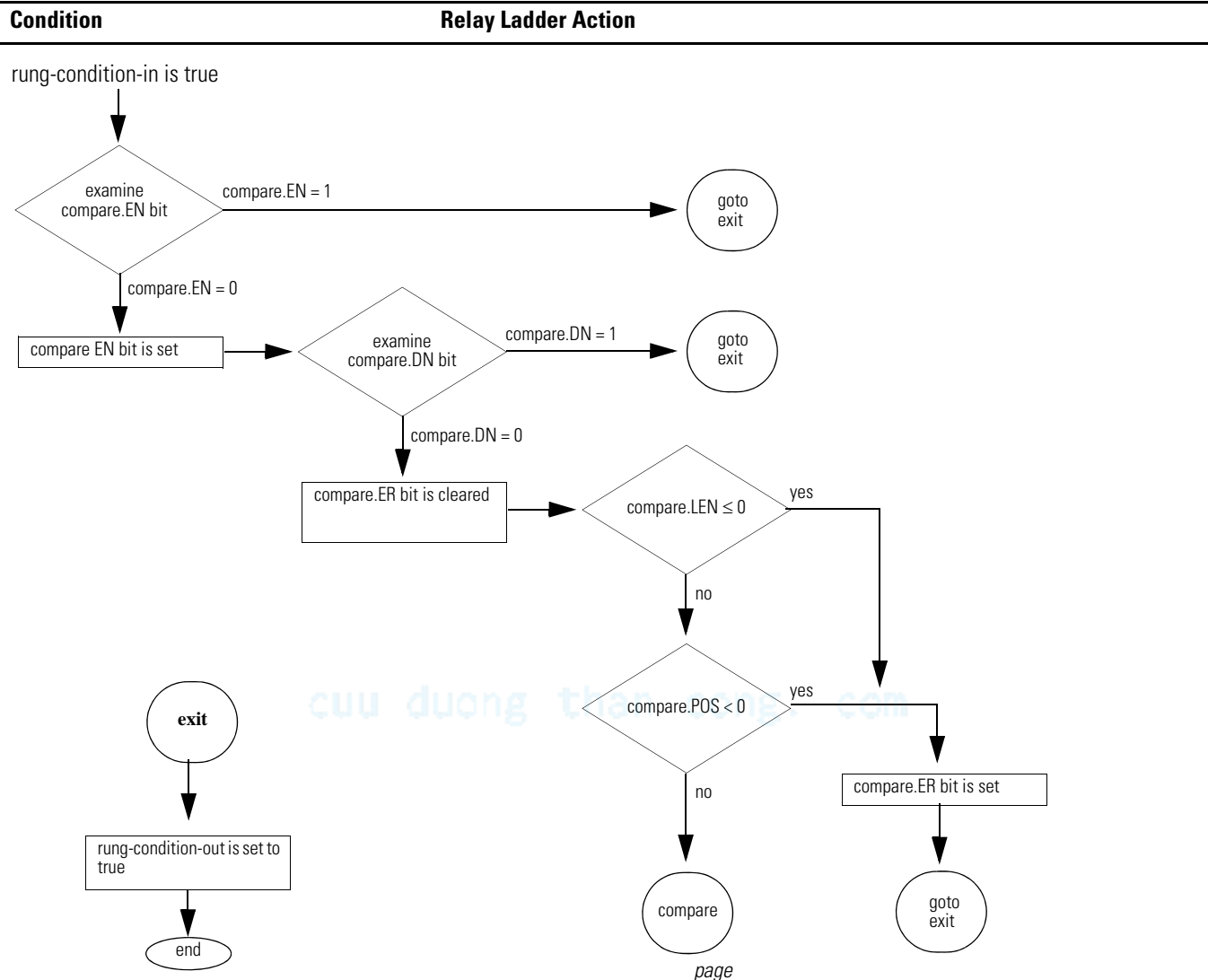
A Major Fault Will Occur If:	Fault Type	Fault Code
Result.POS > size of Result array	4	20

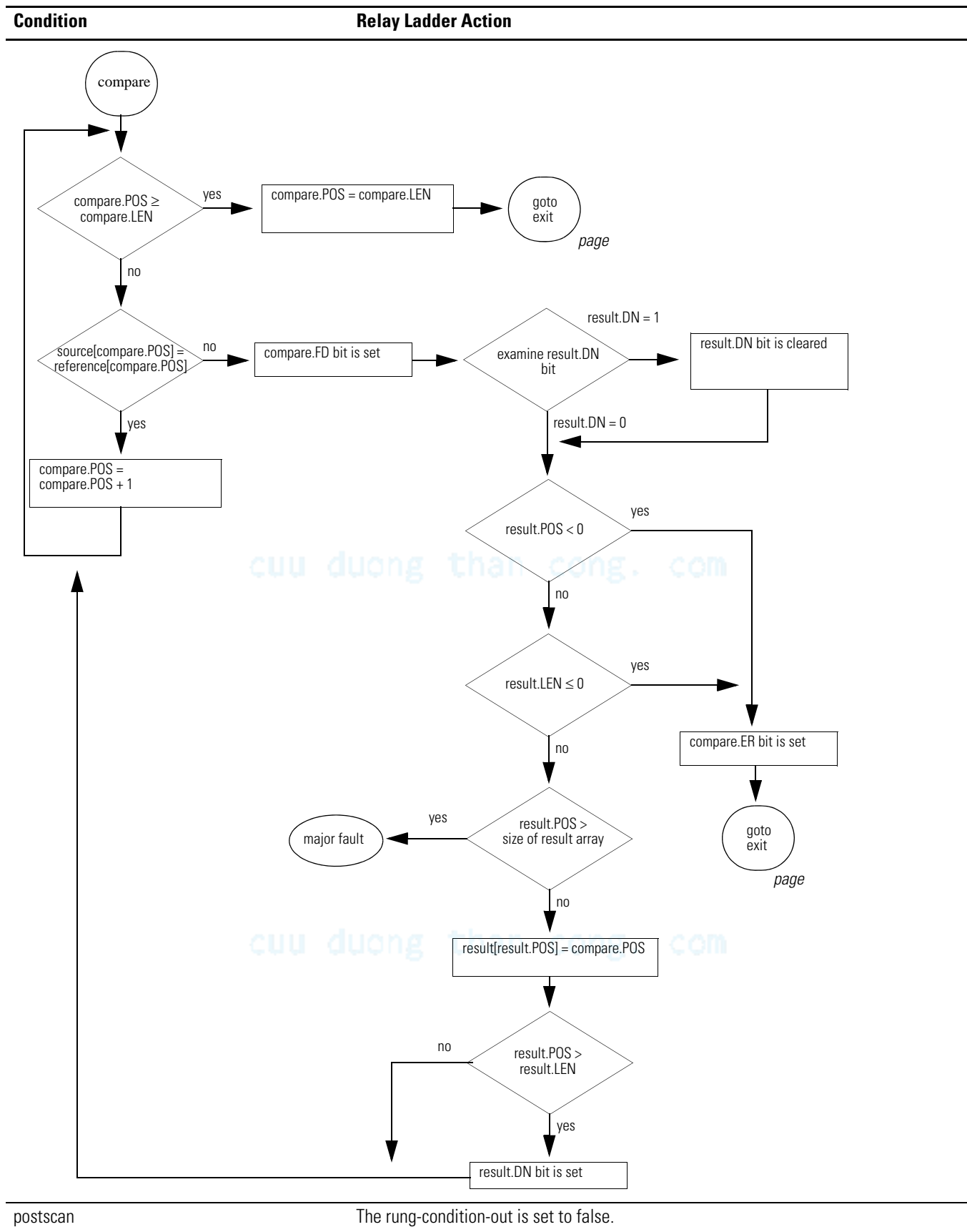
cuu duong than cong. com

cuu duong than cong. com

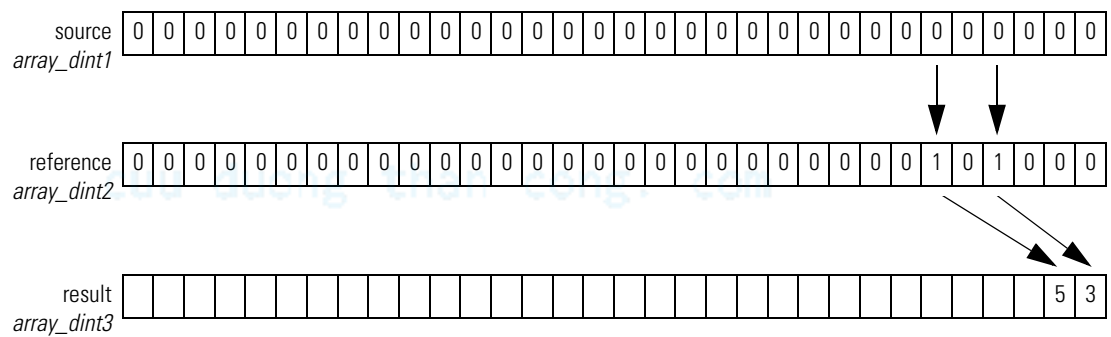
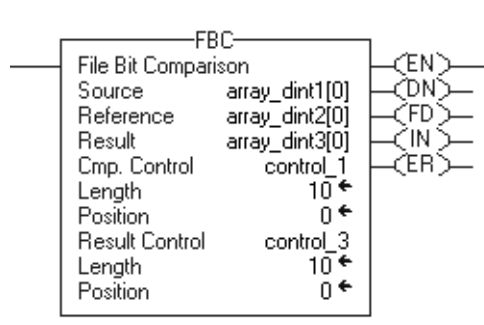
**Execution:**







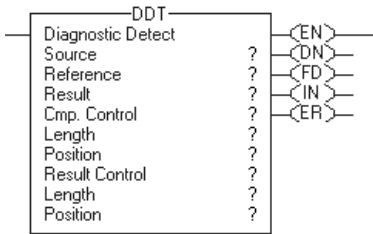
**Example:** When enabled, the FBC instruction compares the source *array\_dint1* to the reference *array\_dint2* and stores the locations of any mismatches in the result *array\_dint3*.



# 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 <b>do not</b> use CONTROL.POS in the subscript
Reference	DINT	array tag	array to compare to the source <b>do not</b> use CONTROL.POS in the subscript
Result	DINT	array tag	array to store the results <b>do not</b> 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.

## Selecting the search mode

If You Want To Detect	Select This Mode
One mismatch at a time	Set the .IN bit in the compare CONTROL structure.  Each time the rung-condition-in goes from false to true, the DDT instruction searches for the next mismatch between the Source and Reference arrays. Upon finding a mismatch, the instruction sets the .FD bit, records the position of the mismatch, and stops executing.
All mismatches	Clear the .IN bit in the compare CONTROL structure.  Each time the rung-condition-in goes from false to true, the DDT instruction searches for all mismatches between the Source and Reference arrays.

**Arithmetic Status Flags:** not affected

### Fault Conditions:

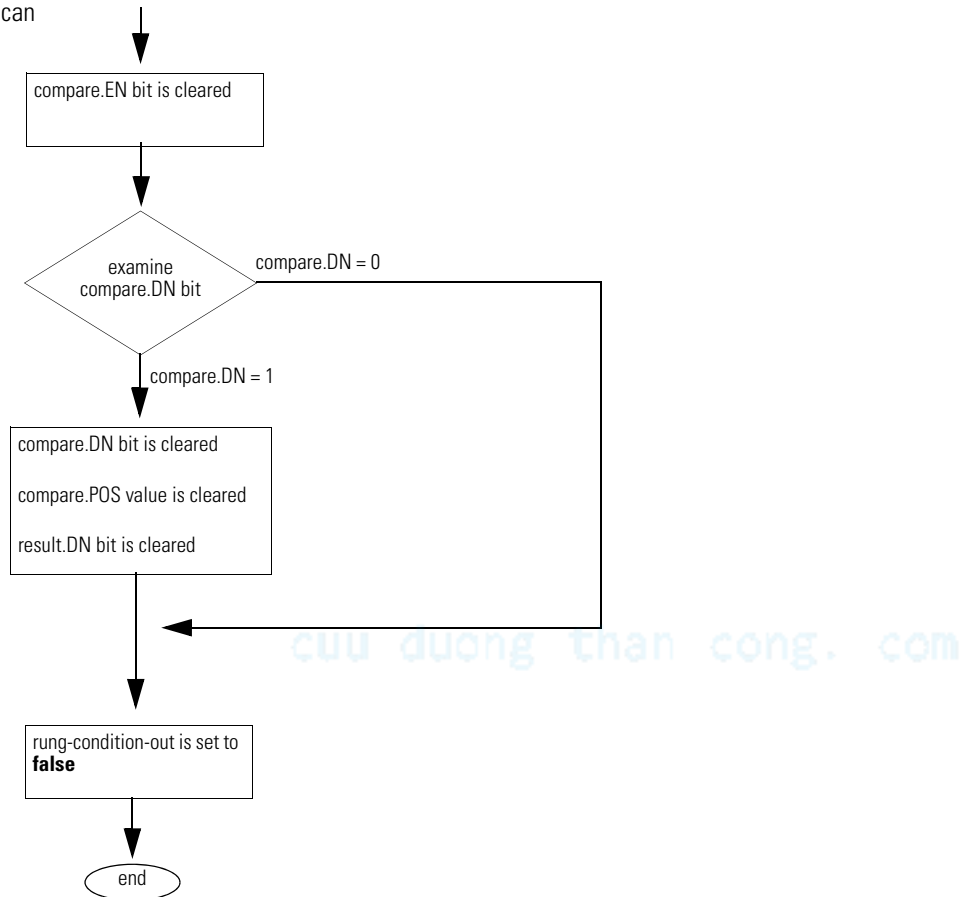
A Major Fault Will Occur If	Fault Type:	Fault Code
Result.POS > size of Result array	4	20

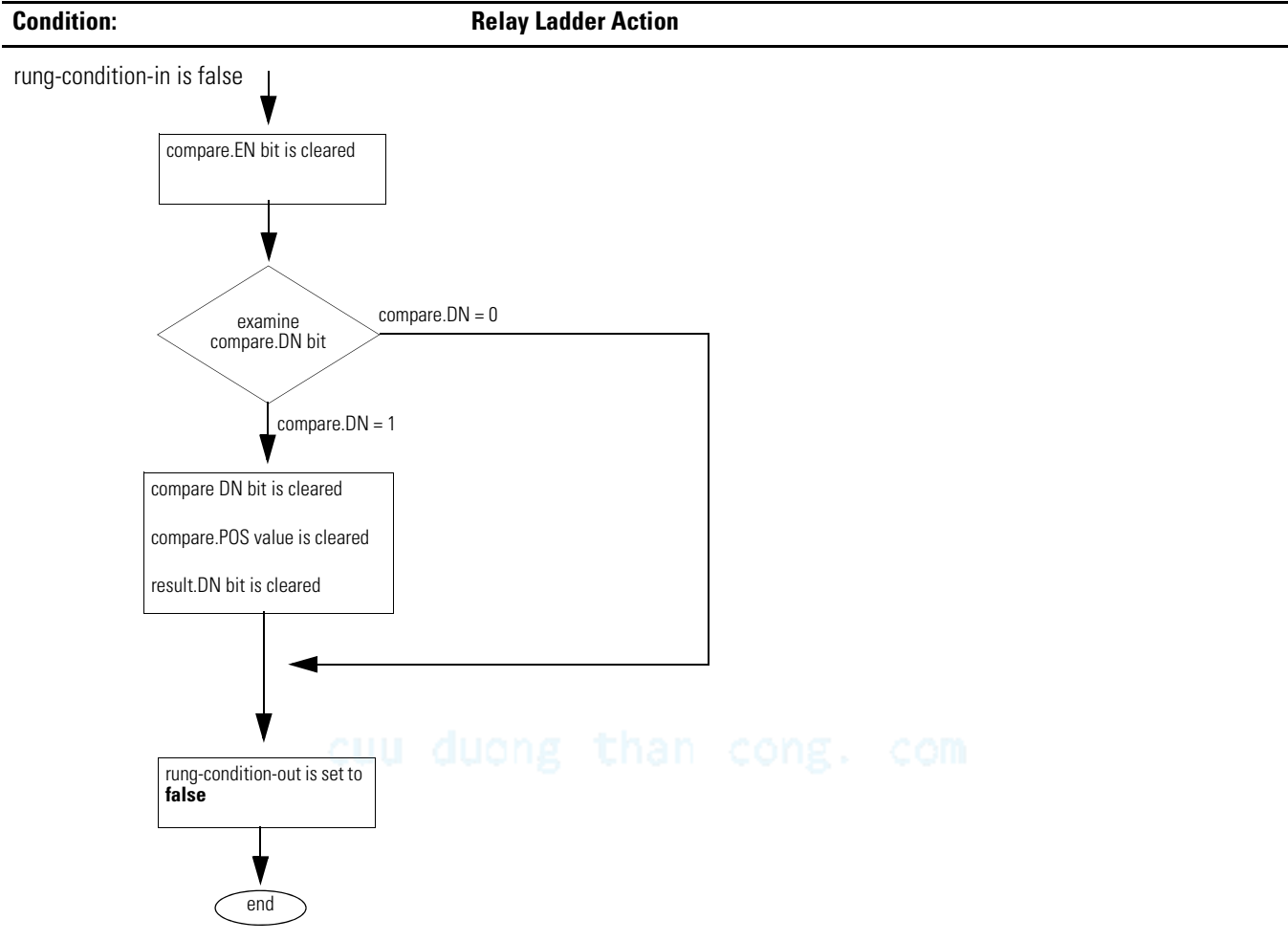
cuu duong than cong. com

cuu duong than cong. com

**Execution:****Condition:                      Relay Ladder Action**

prescan



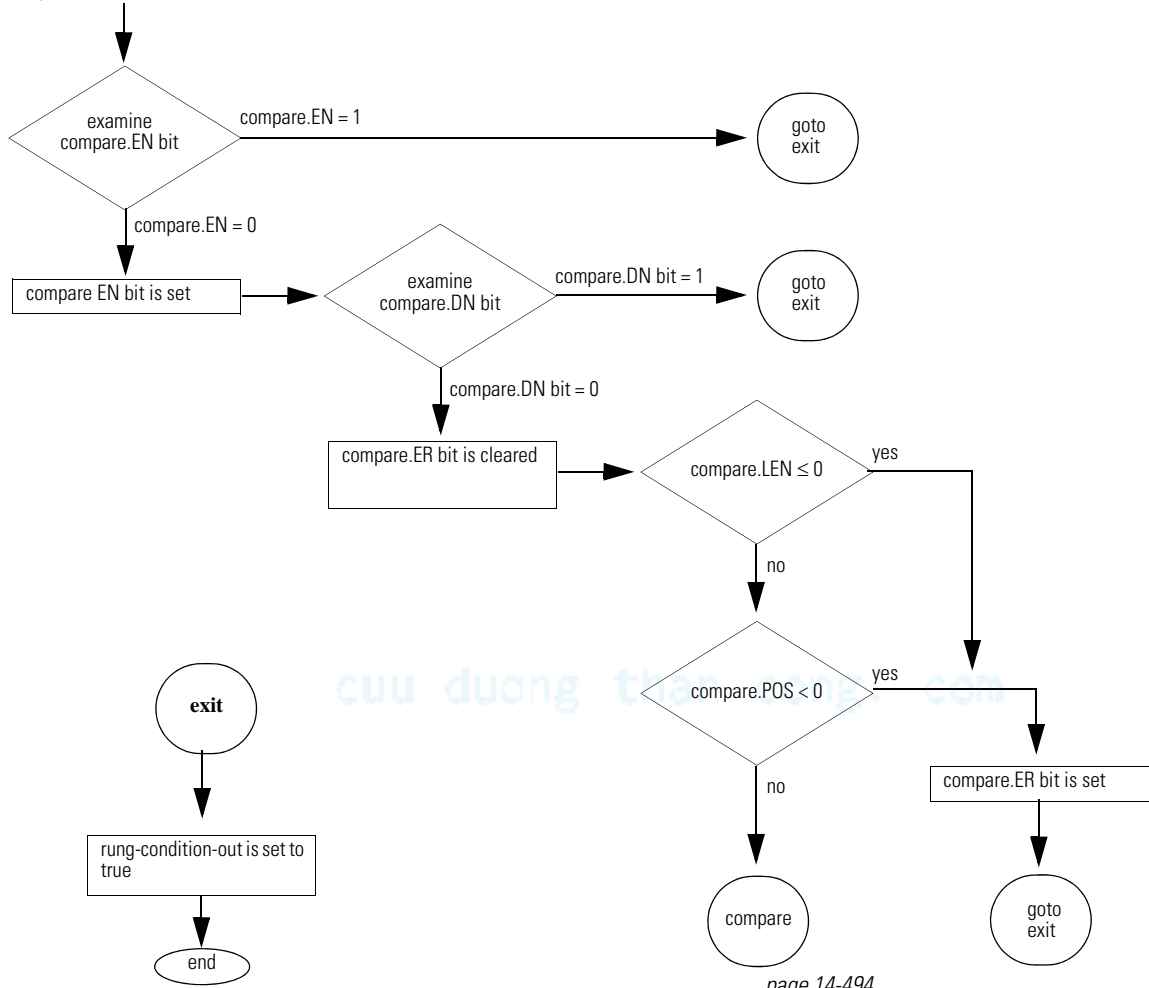


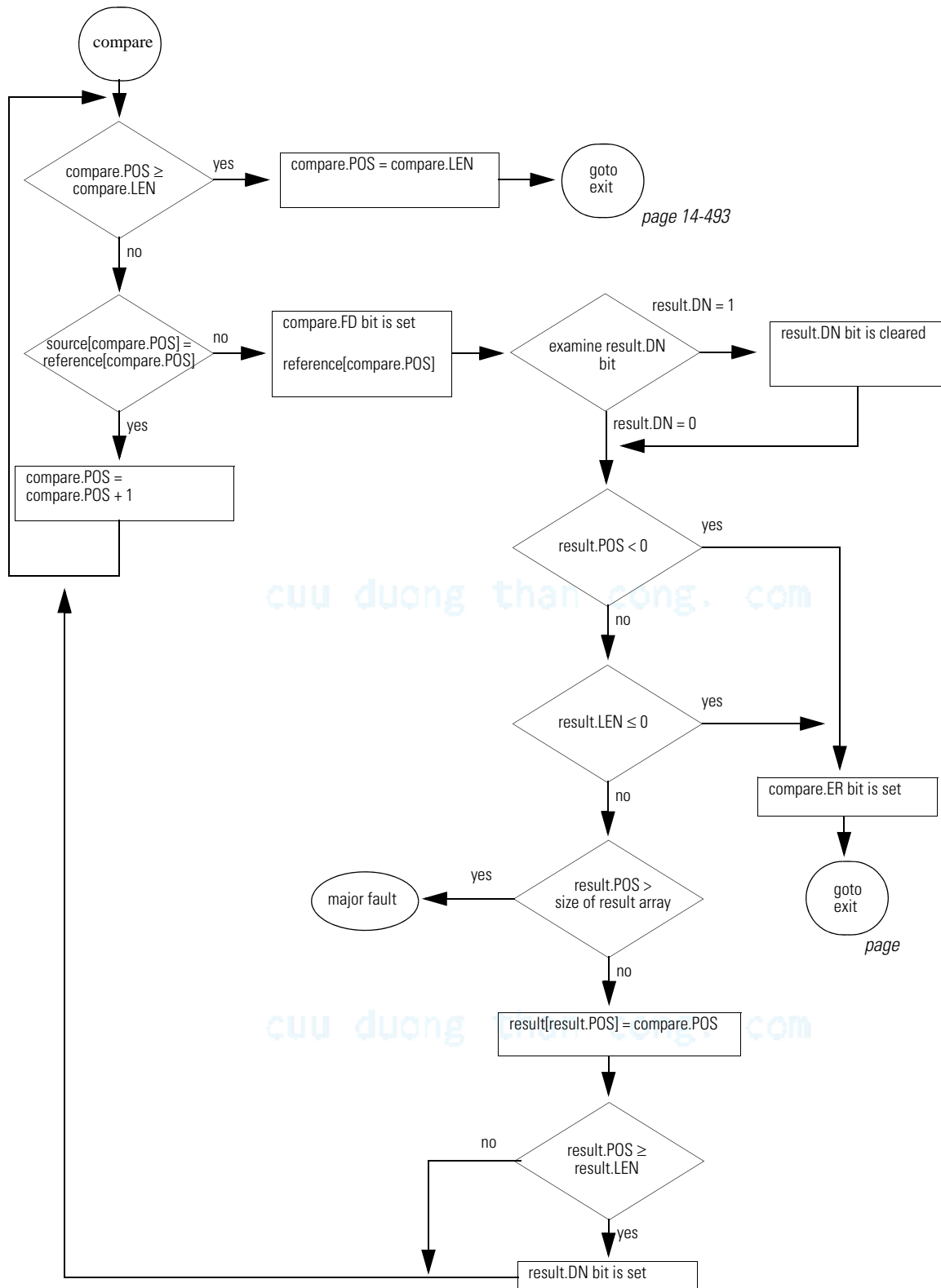
---

**Condition: Relay Ladder Action**


---

rung-condition-in is true

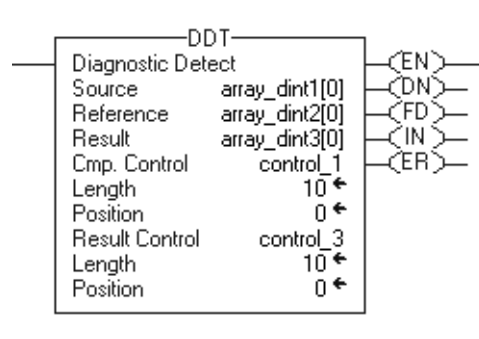


**Condition:****Relay Ladder Action**

postscan

The rung-condition-out is set to false.

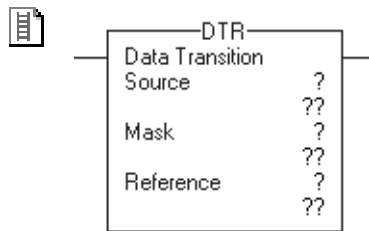
**Example:** When enabled, the DDT instruction compares the source *array\_dint1* to the reference *array\_dint2* and stores the locations of any mismatches in the result *array\_dint3*. The controller also changes the mismatched bits in the reference *array\_dint2* to match the source *array\_dint1*.



# 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	array to compare to the reference
		tag	
Mask	DINT	immediate	which bits to block or pass
		tag	
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

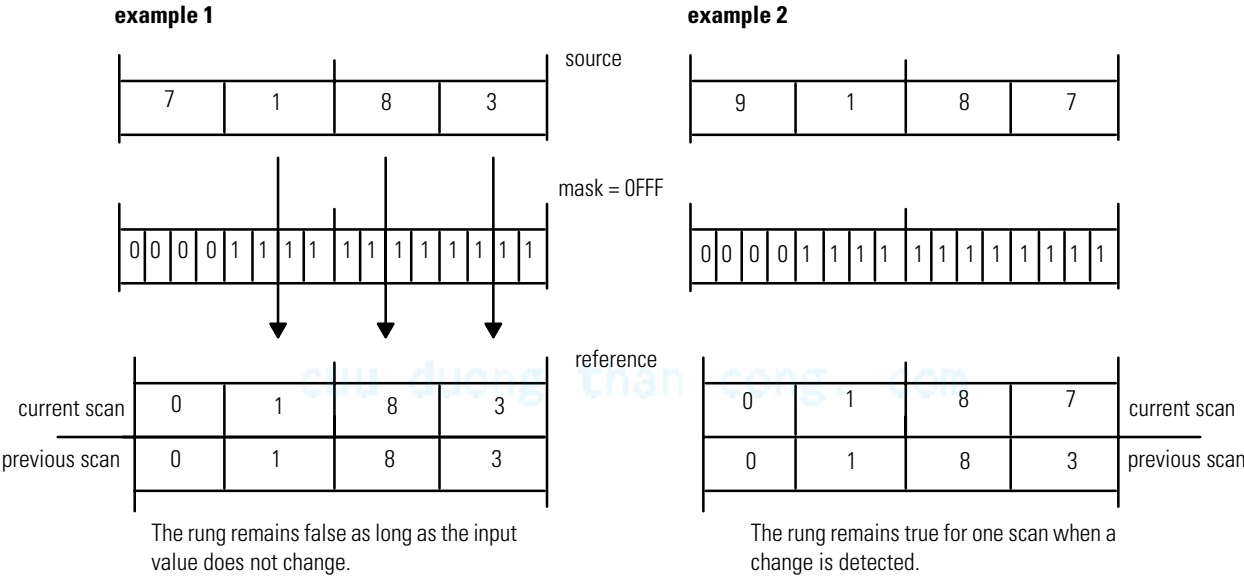
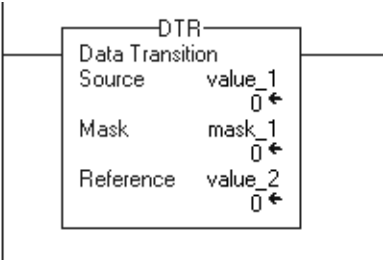
**Arithmetic Status Flags:** not affected

**Fault Conditions:** none

**Execution:** [duong than cong. com](http://duongthancong.com)

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.
rung-condition-in is true	<pre> graph TD     Start([rung-condition-in is true]) --&gt; Decision{masked source = reference}     Decision -- no --&gt; Box1[reference is set equal to masked source rung-condition-out is set to <b>true</b>]     Decision -- yes --&gt; Box2[rung-condition-out is set to <b>false</b>]     Box1 --&gt; End([end])     Box2 --&gt; End           </pre>
postscan	The rung-condition-out is set to false.

**Example:** When enabled, the DTR instruction masks *value\_1*. If there is a difference in the two values, the rung-condition-out is set to true.



13385

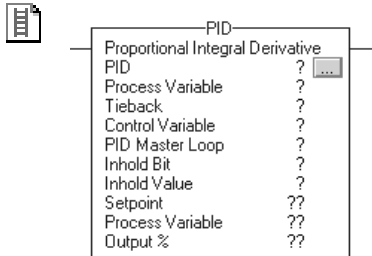
A 0 in the mask leaves the bit unchanged.

## Proportional Integral Derivative (PID)

The PID instruction controls a process variable such as flow, pressure, temperature, or level.

### Operands:

### Relay Ladder



Operand	Type	Format	Description
PID	PID	structure	PID structure
Process variable	SINT	tag	value you want to control
	INT		
	DINT		
	REAL		
Tieback	SINT	immediate	( <i>optional</i> ) output of a hardware hand/auto station which is bypassing the output of the controller
	INT	tag	Enter 0 if you don't want to use this parameter.
	DINT		
	REAL		
Control variable	SINT	tag	value which goes to the final control device (valve, damper, etc.)
	INT		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.
	DINT		
	REAL		
PID master loop	PID	structure	( <i>optional</i> ) PID tag for the master PID If you are performing cascade control and this PID is a slave loop, enter the name of the master PID. Enter 0 if you don't want to use this parameter.
Inhold bit	BOOL	tag	( <i>optional</i> ) current status of the inhold bit from a 1756 analog output channel to support bumpless restart Enter 0 if you don't want to use this parameter.
Inhold value	SINT	tag	( <i>optional</i> ) data readback value from a 1756 analog output channel to support bumpless restart
	INT		Enter 0 if you don't want to use this parameter.
	DINT		
	REAL		

Operand	Type	Format	Description
Setpoint			displays current value of the setpoint
Process variable			displays current value of the scaled Process Variable
Output %			displays current output percentage value



```
PID(PID,ProcessVariable,
Tieback,ControlVariable,
PIDMasterLoop,InholdBit,
InHoldValue);
```

### Structured Text

The operands are the same as those for the relay ladder PID instruction. However, you specify the Setpoint, Process Variable, and Output % by accessing the .SP, .PV.and .OUT members of the PID structure, rather than by including values in the operand list.

cuu duong than cong. com

cuu duong than cong. com

## 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.																																																
		<table><tr><th>This Bit</th><th>Is This Member</th></tr><tr><td>31</td><td>.EN</td></tr><tr><td>30</td><td>.CT</td></tr><tr><td>29</td><td>.CL</td></tr><tr><td>28</td><td>.PVT</td></tr><tr><td>27</td><td>.DOE</td></tr><tr><td>26</td><td>.SWM</td></tr><tr><td>25</td><td>.CA</td></tr><tr><td>24</td><td>.MO</td></tr><tr><td>23</td><td>.PE</td></tr><tr><td>22</td><td>.NDF</td></tr><tr><td>21</td><td>.NOBC</td></tr><tr><td>20</td><td>.NOZC</td></tr><tr><td></td><td></td></tr><tr><th>This Bit:</th><th>Is This Member, Which the PID Instruction Sets</th></tr><tr><td>15</td><td>.INI</td></tr><tr><td>14</td><td>.SPOR</td></tr><tr><td>13</td><td>.OLL</td></tr><tr><td>12</td><td>.OLH</td></tr><tr><td>11</td><td>.EWD</td></tr><tr><td>10</td><td>.DVNA</td></tr><tr><td>09</td><td>.DVPA</td></tr><tr><td>08</td><td>.PVLA</td></tr><tr><td>07</td><td>.PVHA</td></tr></table>	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
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 %																																																
.MAXS	REAL	maximum engineering unit scaling value																																																
.MINS	REAL	minimum engineering unit scaling value																																																

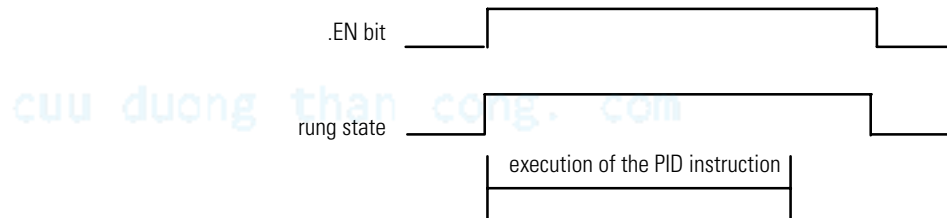
Mnemonic:	Data Type	Description
.DB	REAL	deadband engineering units
.SO	REAL	set output %
.MAXO	REAL	maximum output limit (% of output)
.MINO	REAL	minimum output limit (% of output)
.UPD	REAL	loop update time (seconds)
.PV	REAL	scaled PV value
.ERR	REAL	scaled error value
.OUT	REAL	output %
.PVH	REAL	process variable high alarm limit
.PVL	REAL	process variable low alarm limit
.DVP	REAL	positive deviation alarm limit
.DVN	REAL	negative deviation alarm limit
.PVDB	REAL	process variable alarm deadband
.DVDB	REAL	deviation alarm deadband
.MAXI	REAL	maximum PV value (unscaled input)
.MINI	REAL	minimum PV value (unscaled input)
.TIE	REAL	tieback value for manual control
.MAXCV	REAL	maximum CV value (corresponding to 100%)
.MINCV	REAL	minimum CV value (corresponding to 0%)
.MINTIE	REAL	minimum tieback value (corresponding to 100%)
.MAXTIE	REAL	maximum tieback value (corresponding to 0%)

Mnemonic:	Data Type	Description	
.DATA	REAL[17]	The .DATA member stores:	
		Element:	Description
		.DATA[0]	integral accumulation
		.DATA[1]	derivative smoothing temporary value
		.DATA[2]	previous .PV value
		.DATA[3]	previous .ERR value
		.DATA[4]	previous valid .SP value
		.DATA[5]	percent scaling constant
		.DATA[6]	.PV scaling constant
		.DATA[7]	derivative scaling constant
		.DATA[8]	previous .KP value
		.DATA[9]	previous .KI value
		.DATA[10]	previous .KD value
		.DATA[11]	dependent gain .KP
		.DATA[12]	dependent gain .KI
		.DATA[13]	dependent gain .KD
		.DATA[14]	previous .CV value
		.DATA[15]	.CV descaling constant
.DATA[16]	tieback descaling constant		
.EN	BOOL	enabled	
.CT	BOOL	cascade type (0=slave; 1=master)	
.CL	BOOL	cascade loop (0=no; 1=yes)	
.PVT	BOOL	process variable tracking (0=no; 1=yes)	
.DOE	BOOL	derivative of (0=PV; 1=error)	
.SWM	BOOL	software manual mode (0=no-auto; 1=yes- sw manual)	
.CA	BOOL	control action (0 means E=SP-PV; 1 means E=PV-SP)	
.MO	BOOL	station mode (0=automatic; 1>manual)	
.PE	BOOL	PID equation (0=independent; 1=dependent)	
.NDF	BOOL	no derivative smoothing (0=derivative smoothing filter enabled; 1=derivative smoothing filter disabled)	
.NOBC	BOOL	no bias back calculation (0=bias back calculation enabled; 1=bias back calculation disabled)	
.NOZC	BOOL	no zero crossing deadband (0=deadband is zero crossing; 1=deadband is not zero crossing)	
.INI	BOOL	PID initialized (0=no; 1=yes)	
.SPOR	BOOL	setpoint out of range (0=no; 1=yes)	
.OLL	BOOL	CV is below minimum output limit (0=no; 1=yes)	
.OLH	BOOL	CV is above maximum output limit (0=no; 1=yes)	

Mnemonic:	Data Type	Description
.EWD	BOOL	error is within deadband (0=no; 1=yes)
.DVNA	BOOL	deviation is alarmed low (0=no; 1=yes)
.DVPA	BOOL	deviation is alarmed high (0=no; 1=yes)
.PVLA	BOOL	PV is alarmed low (0=no; 1=yes)
.PVHA	BOOL	PV is alarmed high (0=no; 1=yes)

**Description:** The PID instruction typically receives the process variable (PV) from an analog input module and modulates a control variable output (CV) on an analog output module in order to maintain the process variable at the desired setpoint.

The .EN bit indicates execution status. The .EN bit is set when the rung-condition-in transitions from false to true. The .EN bit is cleared when the rung-condition-in becomes false. The PID instruction does not use a .DN bit. The PID instruction executes every scan as long as the rung-condition-in is true.



**Arithmetic Status Flags:** not affected

#### Fault Conditions:

##### IMPORTANT

These faults were major faults in the PLC-5 controller.

A Minor Fault Will Occur If	Fault Type	Fault Code
.UPD $\leq$ 0	4	35
setpoint out of range	4	36

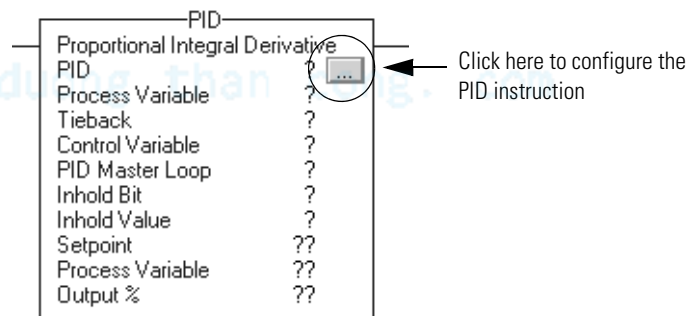


**Execution:**

Condition	Action	Action
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes.	na
	The rung-condition-out is set to true.	
EnableIn is set	na	EnableIn is always set.
		The instruction executes.
instruction execution	The instruction executes the PID loop.	The instruction executes the PID loop.
postscan	The rung-condition-out is set to false.	No action taken.

**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 effect as soon as you click on another field, click OK, click Apply, or press Enter.

In This Field	Specify:
Setpoint (SP)	Enter a setpoint value (.SP).
Set output %	Enter a set output percentage (.SO).  In software manual mode, this value is used for the output.  In auto mode, this value displays the output %.
Output bias	Enter an output bias percentage (.BIAS).
Proportional gain ( $K_p$ )	Enter the proportional gain (.KP).  For independent gains, it's the proportional gain (unitless).  For dependent gains, it's the controller gain (unitless).
Integral gain ( $K_i$ )	Enter the integral gain (.KI).  For independent gains, it's the integral gain (1/sec). For dependent gains, it's the reset time (minutes per repeat).
Derivative time ( $K_d$ )	Enter the derivative gain (.KD).  For independent gains, it's the derivative gain (seconds).  For dependent gains, it's the rate time minutes).
Manual mode	Select either manual (.MO) or software manual (.SWM).  Manual mode overrides software manual mode if both are selected.

## 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).
CV low limit	Enter a low limit for the control variable (.MINO).
Deadband value	Enter a deadband value (.DB).
No derivative smoothing	Enable or disable this selection (.NDF).
No bias calculation	Enable or disable this selection (.NOBC).
No zero crossing in deadband	Enable or disable this selection (.NOZC).
PV tracking	Enable or disable this selection (.PVT).
Cascade loop	Enable or disable this selection (.CL).
Cascade type	If cascade loop is enabled, select either slave or master (.CT).

## Specifying Alarms

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

In This Field	Specify
PV high	Enter a PV high alarm value (.PVH).
PV low	Enter a PV low alarm value (.PVL).
PV deadband	Enter a PV alarm deadband value (.PVDB).
positive deviation	Enter a positive deviation value (.DVP).
negative deviation	Enter a negative deviation value (.DVN).
deviation deadband	Enter a deviation alarm deadband value (.DVDB).

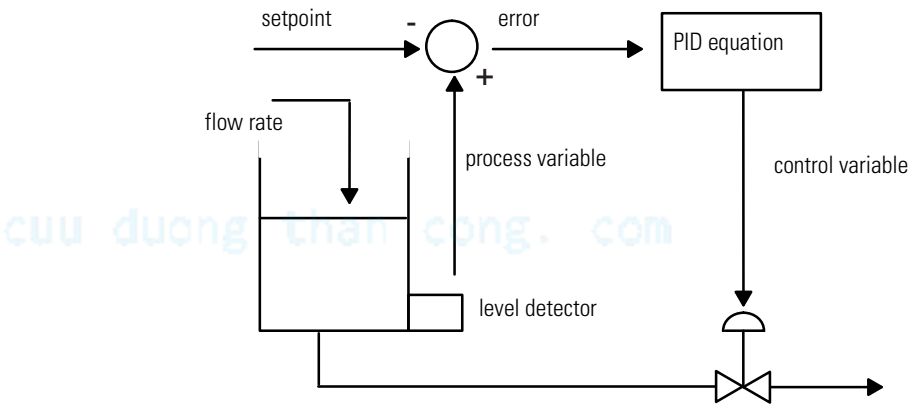
# Specifying Scaling

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

In this field	Specify
PV unscaled maximum	Enter a maximum PV value (.MAXI) that equals the maximum unscaled value received from the analog input channel for the PV value.
PV unscaled minimum	Enter a minimum PV value (.MINI) that equals the minimum unscaled value received from the analog input channel for the PV value.
PV engineering units maximum	Enter the maximum engineering units corresponding to .MAXI (.MAXS)
PV engineering units minimum	Enter the minimum engineering units corresponding to .MINI (.MINS)
CV maximum	Enter a maximum CV value corresponding to 100% (.MAXCV).
CV minimum	Enter a minimum CV value corresponding to 0% (.MINCV).
Tieback maximum	Enter a maximum tieback value (.MAXTIE) that equals the maximum unscaled value received from the analog input channel for the tieback value.
Tieback minimum	Enter a minimum tieback value (.MINTIE) that equals the minimum unscaled value received from the analog input channel for the tieback value.
PID Initialized	If you change scaling constants during Run mode, turn this off to reinitialize internal descaling values (.INI).

# Using PID Instructions

PID closed-loop control holds a process variable at a desired set point. The following figure shows a flow-rate/fluid level example:



14271

In the above example, the level in the tank is compared against the setpoint. If the level is higher than the setpoint, the PID equation increases the control variable and causes the outlet valve from the tank to open; thereby decreasing the level in the tank.

The PID equation used in the PID instruction is a positional form equation with the option of using either independent gains or dependent gains. When using independent gains, the proportional, integral, and derivative gains only affect their specific proportional, integral, or derivative terms respectively. When using dependent gains, the proportional gain is replaced with a controller gain which affects all three terms. You can use either form of equation to perform the same type of control. The two equation types are merely provided to let you use the equation type with which you are most familiar.

Gains Option	Derivative Of	Equation
Dependent gains (ISA standard)	error (E)	$CV = K_C \left[ E + \frac{1}{T_i} \int_0^t E dt + T_d \frac{dE}{dt} \right] + BIAS$
	process variable (PV)	$E = SP - PV$ $CV = K_C \left[ E + \frac{1}{T_i} \int_0^t E dt + T_d \frac{dPV}{dt} \right] + BIAS$
		$E = PV - SP$ $CV = K_C \left[ E + \frac{1}{T_i} \int_0^t E dt + T_d \frac{dPV}{dt} \right] + BIAS$
Independent gains	error (E)	$CV = K_P E + K_i \int_0^t E dt + K_d \frac{dE}{dt} + BIAS$
	process variable (PV)	$E = SP - PV$ $CV = K_P E + K_i \int_0^t E dt + K_d \frac{dPV}{dt} + BIAS$
		$E = PV - SP$ $CV = K_P E + K_i \int_0^t E dt + K_d \frac{dPV}{dt} + BIAS$

Where:

Variable	Description
$K_p$	proportional gain (unitless) $K_p = K_c$ unitless
$K_i$	integral gain (seconds <sup>-1</sup> ) To convert between $K_i$ (integral gain) and $T_i$ (reset time), use: $K_i = \frac{K_c}{60T_i}$
$K_d$	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)	<p>also known as set output mode</p> <p>lets the user set the output % from the software</p> <p>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.</p>
manual (.MO)	<p>takes the tieback value, as an input, and adjusts its internal variables to generate the same value at the output</p> <p>The tieback input to the PID instruction is scaled to 0-100% according to the values of .MINTIE and .MAXTIE and is used as the output of the loop. The tieback input typically comes from the output of a hardware hand/auto station which is bypassing the output from the controller.</p> <p><b>Note:</b> Manual mode overrides software manual mode if both mode bits are set on.</p>

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

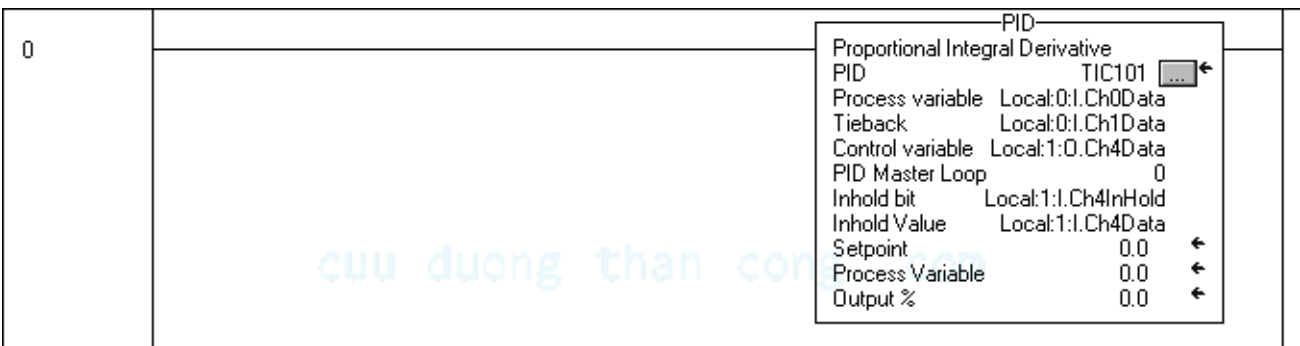
The PID instruction and the sampling of the process variable need to be updated at a periodic rate. This update time is related to the physical process you are controlling. For very slow loops, such as temperature loops, an update time of once per second or even longer is usually sufficient to obtain good control. Somewhat faster loops, such as pressure or flow loops, may require an update time such as once every 250 milliseconds. Only rare cases, such as tension control

on an unwinder spool, require loop updates as fast as every 10 milliseconds or faster.

Because the PID instruction uses a time base in its calculation, you need to synchronize execution of this instruction with sampling of the process variable (PV).

The easiest way to execute the PID instruction is to put the PID instruction in a periodic task. Set the loop update time (.UPD) equal to the periodic task rate and make sure that the PID instruction is executed every scan of the periodic task

### Relay Ladder



### Structured Text

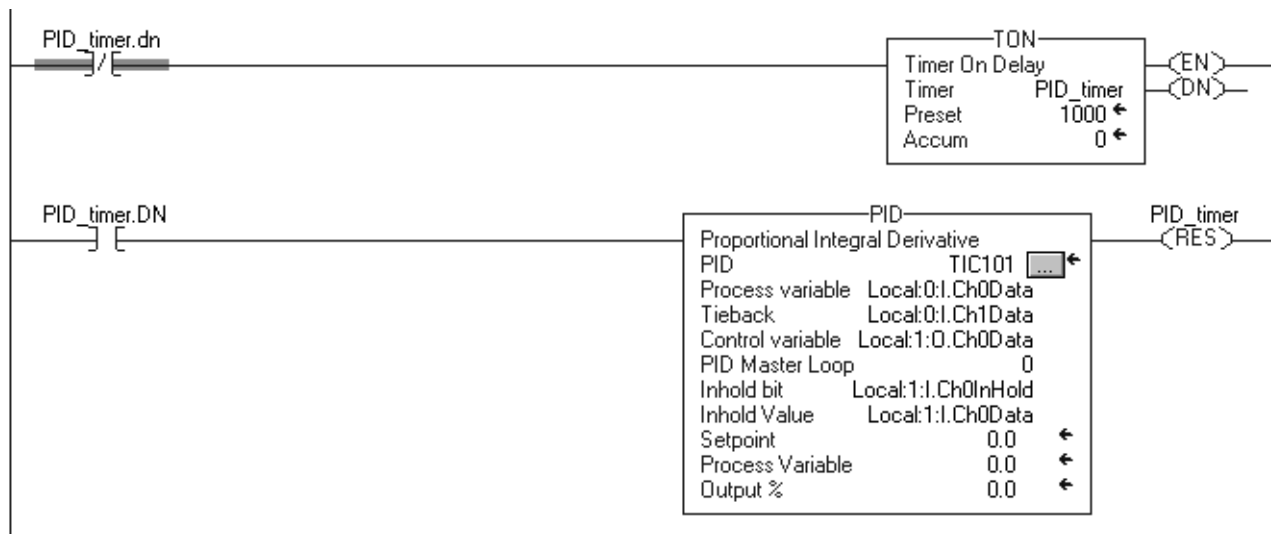
```
PID(TIC101,Local:0:I.Ch0Data,Local:0:I.Ch1Data,
    Local:1:O.Ch4Data,0,Local:1:I.Ch4InHold,
    Local:1:I.Ch4Data);
```

When using a periodic task, make sure that the analog input used for the process variable is updated to the processor at a rate that is significantly faster than the rate of the periodic task. Ideally, the process variable should be sent to the processor at least five to ten times faster than the periodic task rate. This minimizes the time difference between actual samples of the process variable and execution of the PID loop. For example, if the PID loop is in a 250 millisecond periodic task, use a loop update time of 250 milliseconds (.UPD = .25), and configure the analog input module to produce data at least about every 25 to 50 msecs.

Another, somewhat less accurate, method of executing a PID instruction is to place the instruction in a continuous task and use a timer done bit to trigger execution of the PID instruction.



## Relay Ladder



## Structured Text

```

PID_timer.pre := 1000
TONR(PID_timer);

IF PID_timer.DN THEN

    PID(TIC101,Local:0:I.Ch0Data,Local:0:I.Ch1Data,
        Local:1:O.Ch0Data,0,Local:1:I.Ch0InHold,
        Local:1:I.Ch0Data);

END_IF;

```

In this method, the loop update time of the PID instruction should be set equal to the timer preset. As in the case of using a periodic task, you should set the analog input module to produce the process variable at a significantly faster rate than the loop update time. You should only use the timer method of PID execution for loops with loop update times that are at least several times longer than the worst-case execution time for your continuous task.

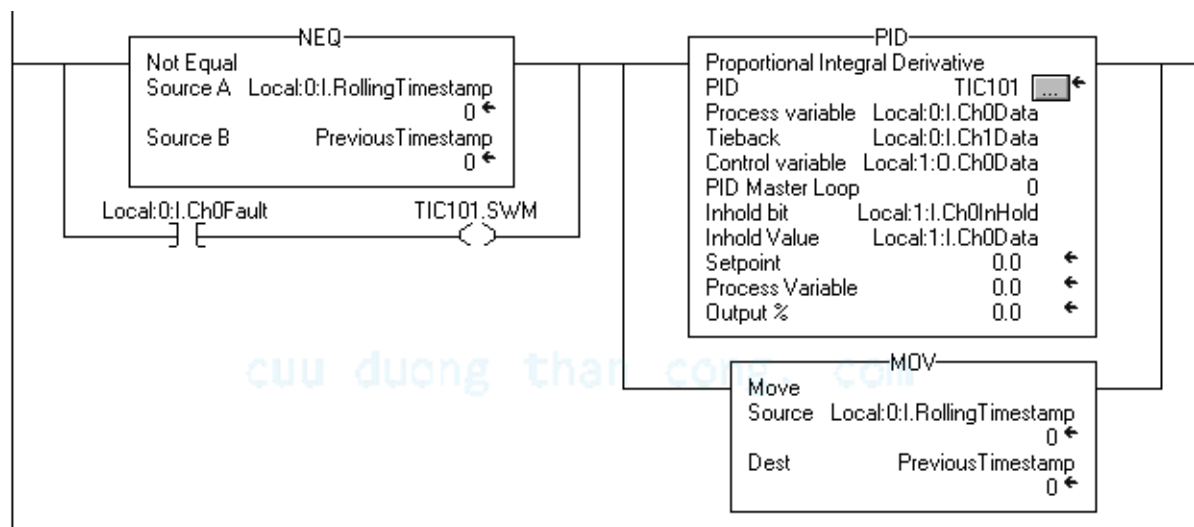
The most accurate way to execute a PID instruction is to use the real time sampling (RTS) feature of the 1756 analog input modules. The analog input module samples its inputs at the real time sampling rate you configure when you set up the module. When the module's real time sample period expires, it updates its inputs and updates a rolling timestamp (represented by the .RollingTimestamp member of the analog input data structure) produced by the module.

The timestamp ranges from 0-32767 milliseconds. Monitor the timestamp. When it changes, a new process variable sample has been received. Every time a timestamp changes, execute the PID instruction once. Because the process variable sample is driven by the analog input module, the input sample time is very accurate, and the loop update time used by the PID instruction should be set equal to the RTS time of the analog input module.

To make sure that you do not miss samples of the process variable, execute your logic at a rate faster than the RTS time. For example, if the RTS time is 250 msecs, you could put the PID logic in a periodic task that runs every 100 msecs to make sure that you never miss a sample. You could even place the PID logic in a continuous task, as long as you make sure that the logic would be updated more frequently than once every 250 milliseconds.

An example of the RTS method of execution is shown below. The execution of the PID instruction depends on receiving new analog input data. If the analog input module fails or is removed, the controller stops receiving rolling timestamps and the PID loop stops executing. You should monitor the status bit of the PV analog input and if it shows bad status, force the loop into software manual mode and execute the loop every scan. This lets operator still manually change the output of the PID loop.

### Relay Ladder



## Structured Text

```

IF (Local:0:I.Ch0Fault) THEN
    TIC101.SWM [:=] 1;
ELSE
    TIC101.SWM := 0;
END_IF;

IF (Local:0:I.RollingTimestamp<>PreviousTimestamp) OR
    (Local:0:I.Ch0Fault) THEN

    PreviousTimestamp := Local:0:I.RollingTimestamp;

    PID(TIC101,Local:0:I.Ch0Data,Local:0:I.Ch1Data,
        Local:1:0.Ch0Data,0,Local:1:I.Ch0InHold,
        Local:1:I.Ch0Data);

END_IF;

```

## Bumpless restart

The PID instruction can interact with the 1756 analog output modules to support a bumpless restart when the controller changes from Program to Run mode or when the controller powers up.

When a 1756 analog output module loses communications with the controller or senses that the controller is in Program mode, the analog output module sets its outputs to the fault condition values you specified when you configured the module. When the controller then returns to Run mode or re-establishes communications with the analog output module, you can have the PID instruction automatically reset its control variable output equal to the analog output by using the Inhold bit and Inhold Value parameters on the PID instruction.

To set a bumpless restart:

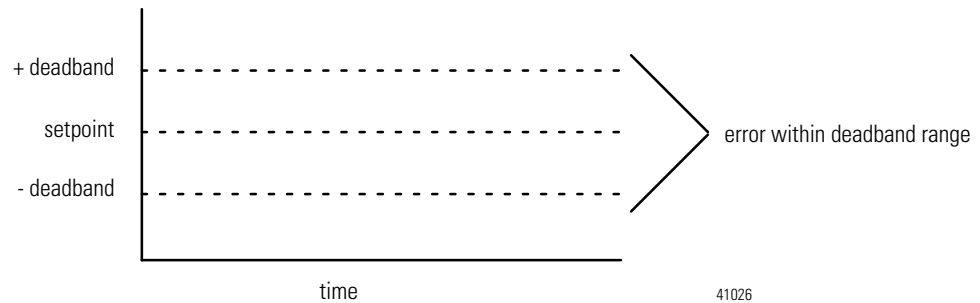
Do This	Details:
Configure the 1756 analog output module's channel which receives the control variable from the PID instruction	<p>Select the "hold for initialization" check box on the properties page for the specific channel of the module.</p> <p>This tells the analog output module that when the controller returns to Run mode or re-establishes communications with the module, the module should hold the analog output at its current value until the value sent from the controller matches (within 0.1% of span) the current value used by the output channel. The controller's output will ramp to the currently held output value by making use of the .BIAS term. This ramping is similar to auto bumpless transfer.</p>
Enter the Inhold bit tag and Inhold Value tag in the PID instruction	<p>The 1756 analog output module returns two values for each channel in its input data structure. The InHold status bit (.Ch2InHold, for example), when true, indicates that the analog output channel is holding its value. The Data readback value (.Ch2Data, for example) shows the current output value in engineering units.</p> <p>Enter the tag of the InHold status bit as the InHold bit parameter of the PID instruction. Enter the tag of the Data readback value as the Inhold Value parameter.</p> <p>When the Inhold bit goes true, the PID instruction moves the Inhold Value into the Control variable output and re-initializes to support a bumpless restart at that value. When the analog output module receives this value back from the controller, it turns off the InHold status bit, which allows the PID instruction to start controlling normally.</p>

## Derivative Smoothing

The derivative calculation is enhanced by a derivative smoothing filter. This first order, low pass, digital filter helps to minimize large derivative term spikes caused by noise in the PV. This smoothing becomes more aggressive with larger values of derivative gain. You can disable derivative smoothing if your process requires very large values of derivative gain ( $K_d > 10$ , for example). To disable derivative smoothing, select the "No derivative smoothing" option on the Configuration tab or set the .NDF bit in the PID structure.

## 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 0 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.

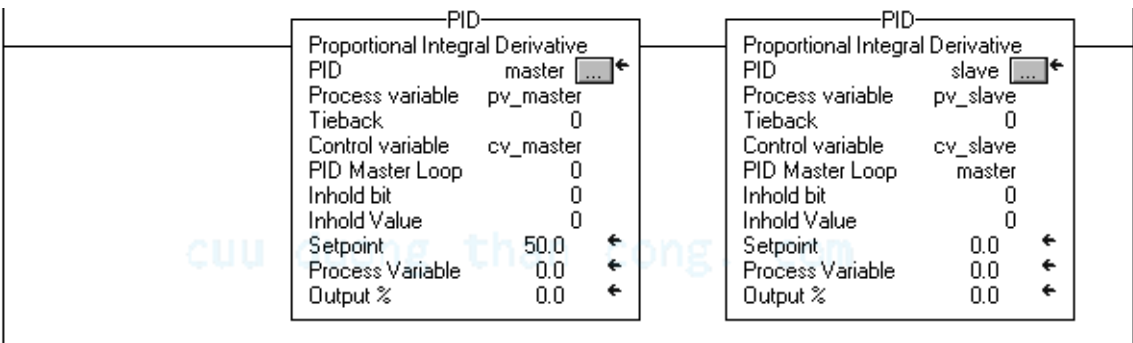
The feedforward value represents a disturbance fed into the PID instruction before the disturbance has a chance to change the process variable. Feedforward is often used to control processes with a transportation lag. For example, a feedforward value representing “cold water poured into a warm mix” could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

## Cascading loops

The PID cascades two loops by assigning the output in percent of the master loop to the setpoint of the slave loop. The slave loop automatically converts the output of the master loop into the correct engineering units for the setpoint of the slave loop, based on the slave loop's values for .MAXS and .MINS.

## Relay Ladder



## Structured Text

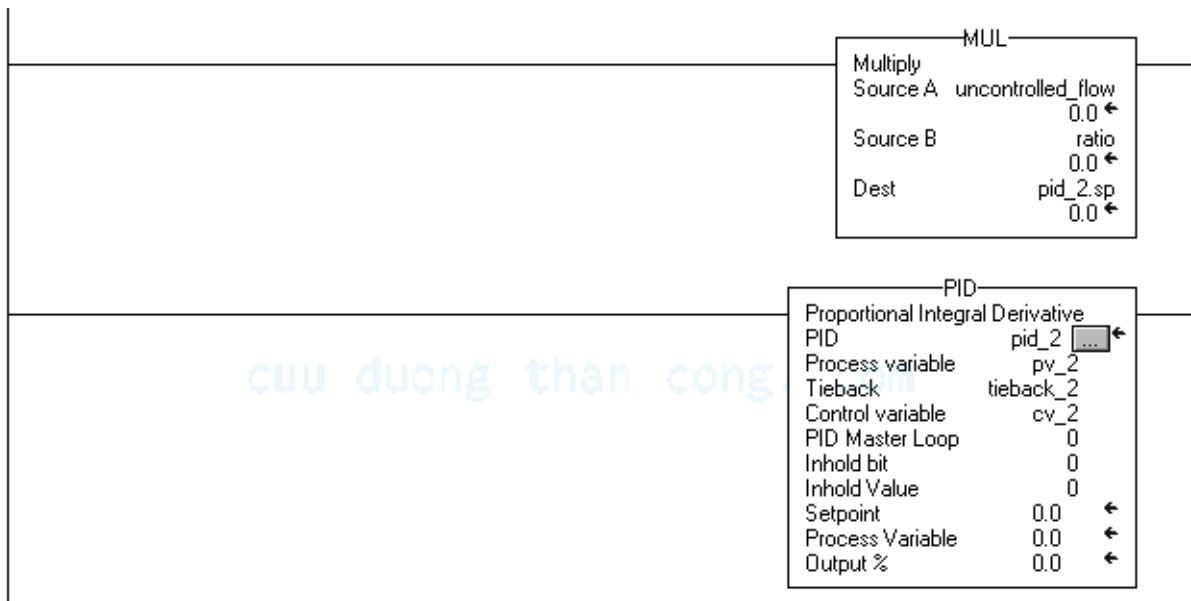
```
PID(master,pv_master,0,cv_master,0,0,0);  
  
PID (slave,pv_slave,0,cv_slave,master,0,0);
```

## 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
```

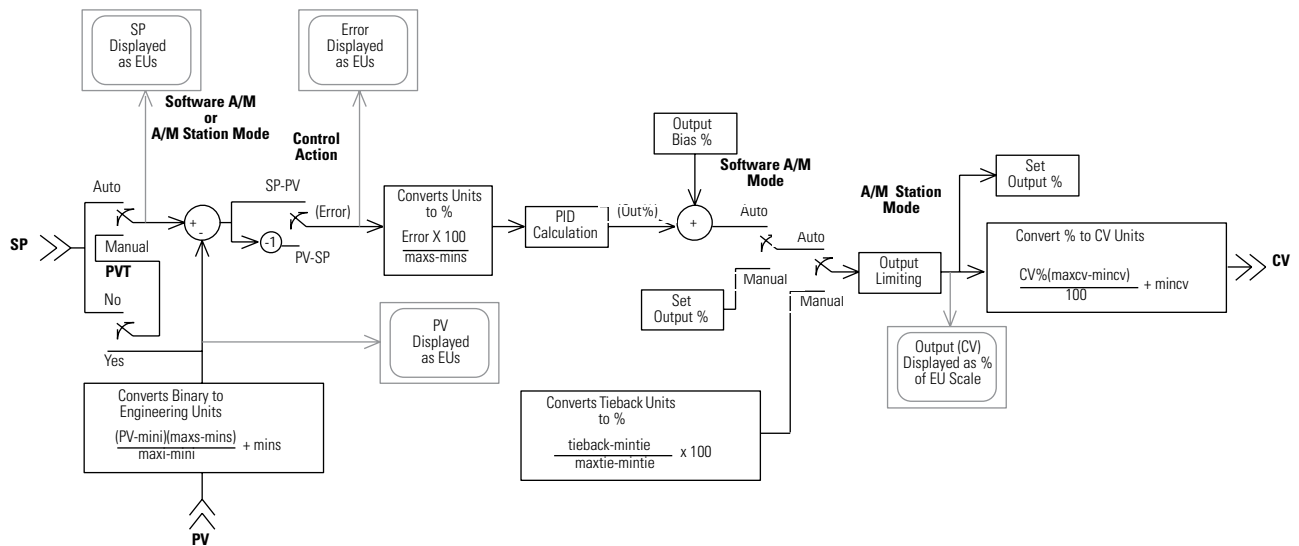
```
PID(pid_2,pv_2,tieback_2,cv_2,0,0,0);
```

For This Multiplication Parameter	Enter This Value
destination	controlled value
source A	uncontrolled value
source B	ratio

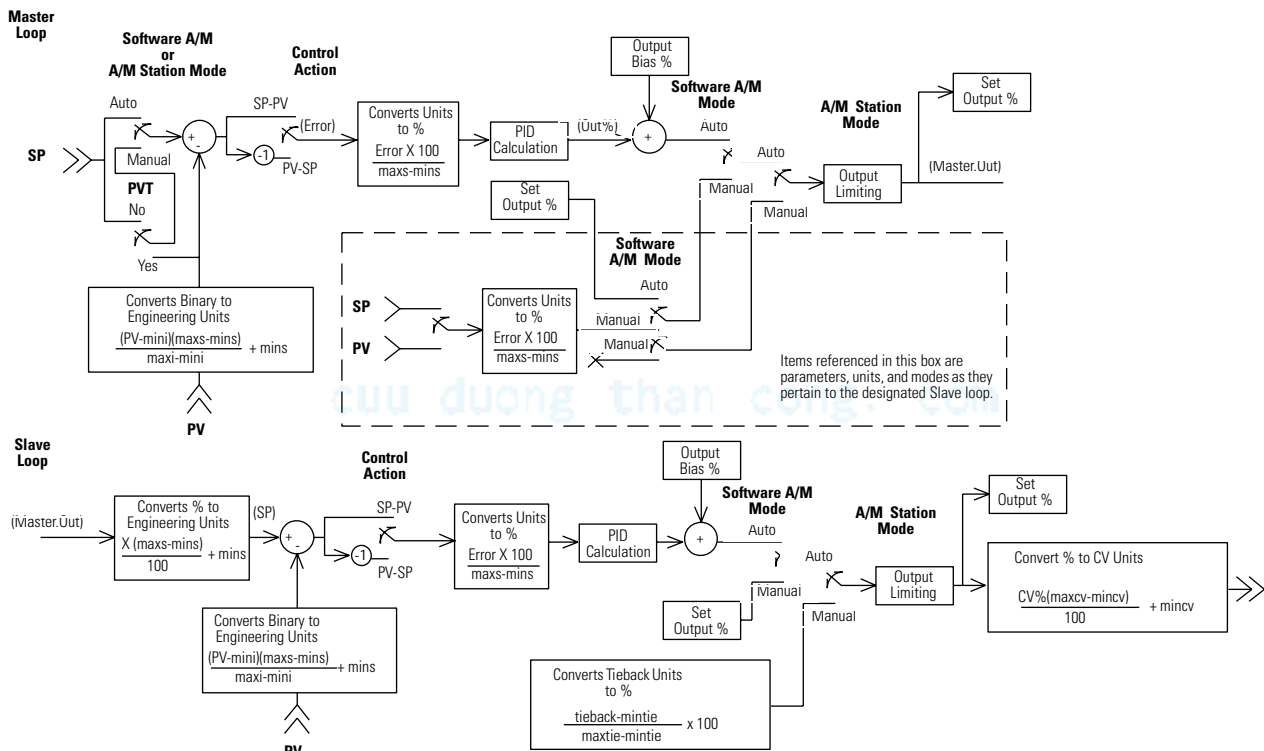
## PID Theory

The following figures show the process flow for a PID instructions.

### PID Process



### PID Process With Master/slave Loops





## Trigonometric Instructions

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

### Introduction

The trigonometric instructions evaluate arithmetic operations using trigonometric operations.

If You Want To	Use This Instruction	Available In These Languages	See Page
Take the sine of a value.	SIN	relay ladder	522
		structured text	
		function block	
Take the cosine of a value.	COS	relay ladder	525
		structured text	
		function block	
Take the tangent of a value.	TAN	relay ladder	529
		structured text	
		function block	
Take the arc sine of a value.	ASN	relay ladder	532
	ASIN <sup>(1)</sup>	structured text	
		function block	
Take the arc cosine of a value.	ACS	relay ladder	536
	ACOS <sup>(1)</sup>	structured text	
		function block	
Take the arc tangent of a value.	ATN	relay ladder	540
	ATAN <sup>(1)</sup>	structured text	
		function block	

<sup>(1)</sup> Structured text only.

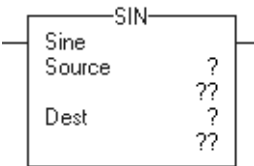
You can mix data types, but loss of accuracy and rounding error might occur and the instruction takes more time to execute. Check the overflow status bit (S:V) to see whether the result was truncated.

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	immediate	find the sine of this value
	INT	tag	
	DINT		
	REAL		
Destination	SINT	tag	tag to store the result
	INT		
	DINT		
	REAL		

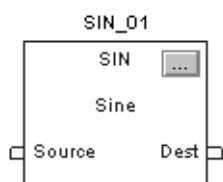


```
dest := SIN(source);
```

## Structured Text

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

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



## Function Block

Operand	Type	Format	Description
SIN tag	FBD_MATH_ADVANCED	structure	SIN structure

## FBD\_MATH\_ADVANCED Structure

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

**Description:** The Source must be greater than or equal to  $-205887.4 (-2\pi \times 2^{15})$  and less than or equal to  $205887.4 (2\pi \times 2^{15})$ . The resulting value in the Destination is always greater than or equal to -1 and less than or equal to 1.

**Arithmetic Status Flags:** Arithmetic status flags are affected.

**Fault Conditions:** none

**Execution:**



## Relay Ladder

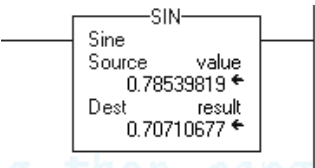
Condition:	Action:
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	The controller calculates the sine of the Source and places the result in the Destination. The rung-condition-out is set to true.
postscan	The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

Function Block

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

**Example:** Calculate the sine of *value* and place the result in *result*.

Relay Ladder



Structured Text

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

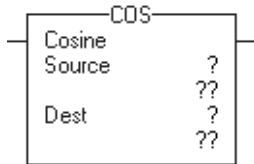
Function Block



## Cosine (COS)

The COS instruction takes the cosine of the Source value (in radians) and stores the result in the Destination.

### Operands:



### Relay Ladder

Operand	Type	Format	Description
Source	SINT	immediate	find the cosine of this value
	INT	tag	
	DINT		
	REAL		
Destination	SINT	tag	tag to store the result
	INT		
	DINT		
	REAL		

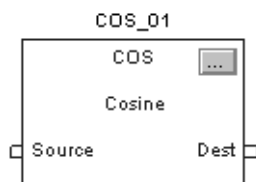


```
dest := COS(source);
```

### Structured Text

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

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



### Function Block

Operand	Type	Format	Description
COS tag	FBD_MATH_ADVANCED	structure	COS structure

**FBD\_MATH\_ADVANCED Structure**

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

**Description:** The Source must be greater than or equal to  $-205887.4 (-2\pi \times 2^{15})$  and less than or equal to  $205887.4 (2\pi \times 2^{15})$ . The resulting value in the Destination is always greater than or equal to -1 and less than or equal to 1.

**Arithmetic Status Flags:** Arithmetic status flags are affected.

**Fault Conditions:** none

**Execution:**

**Relay Ladder**

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

**Function Block**

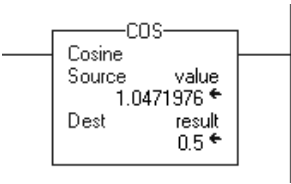
Condition	Action
prescan	No action taken.
instruction first scan	No action taken.
instruction first run	No action taken.
EnableIn is cleared	EnableOut is cleared.
EnableIn is set	The instruction executes. EnableOut is set.
postscan	No action taken.

cuu duong than cong. com

cuu duong than cong. com

**Example:** Calculate the cosine of *value* and place the result in *result*.

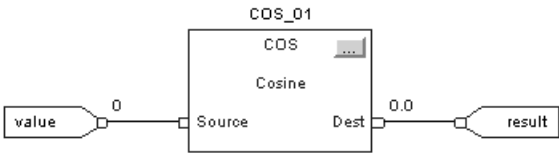
**Relay Ladder**



**Structured Text**

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

**Function Block**



cuu duong than cong. com

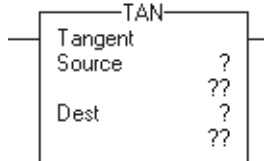
cuu duong than cong. com



## 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	immediate	find the tangent of this value
	INT	tag	
	DINT		
	REAL		
Destination	SINT	tag	tag to store the result
	INT		
	DINT		
	REAL		



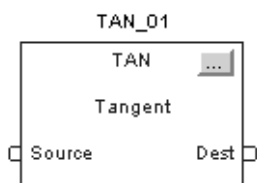
```
dest := TAN(source);
```

### Structured Text

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

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

### Function Block



Operand	Type	Format	Description
TAN tag	FBD_MATH_ADVANCED	structure	TAN structure

**FBD\_MATH\_ADVANCED Structure**

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

**Description:** The Source must be greater than or equal to  $-102943.7(-2\pi \times 2^{14})$  and less than or equal to  $102943.7(2\pi \times 2^{14})$ .

**Arithmetic Status Flags:** Arithmetic status flags are affected.

**Fault Conditions:** none

**Execution:** [duong than cong. com](https://fb.com/tailieudientucntt)

**Relay Ladder**

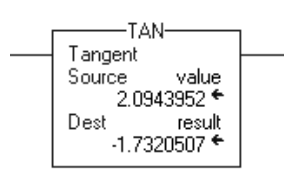
Condition	Action
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	The controller calculates the tangent of the Source and places the result in the Destination.  The rung-condition-out is set to true.
postscan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

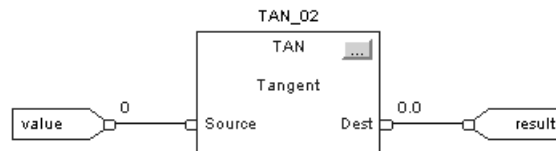
### Relay Ladder



### Structured Text

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

### Function Block



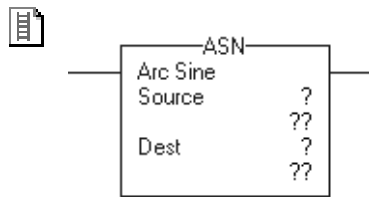
cuu duong than cong. com

cuu duong than cong. com

# 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	immediate	find the arc sine of this value
	INT	tag	
	DINT		
	REAL		
Destination	SINT	tag	tag to store the result
	INT		
	DINT		
	REAL		

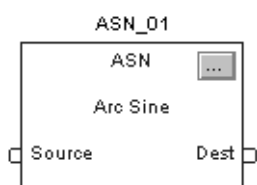


dest := ASIN(source);

## Structured Text

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

See Appendix B 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

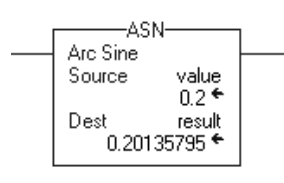
Condition	Action
prescan	No action taken.
instruction first scan	No action taken.
instruction first run	No action taken.
EnableIn is cleared	EnableOut is cleared.
EnableIn is set	The instruction executes.
	EnableOut is set.
postscan	No action taken.

cuu duong than cong. com

cuu duong than cong. com

**Example:** Calculate the arc sine of *value* and place the result in *result*.

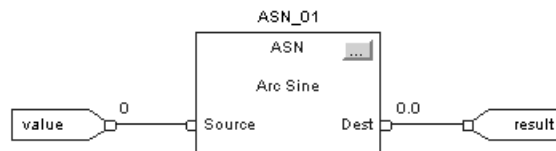
### Relay Ladder



### Structured Text

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

### Function Block



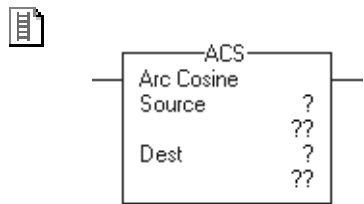
cuu duong than cong. com

cuu duong than cong. com

# 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	immediate	find the arc cosine of this value
	INT	tag	
	DINT		
	REAL		
Destination	SINT	tag	tag to store the result
	INT		
	DINT		
	REAL		

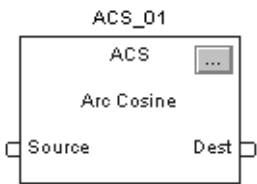


dest := ACOS(source);

## Structured Text

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

See Appendix B 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  $\pi$  (where  $\pi = 3.141593$ ).

**Arithmetic Status Flags:** Arithmetic status flags are affected.

**Fault Conditions:** none

**Execution:**

**Relay Ladder**

Condition	Action
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	The controller 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.

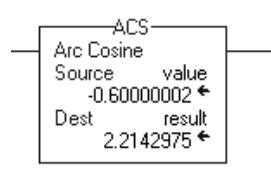
Condition:	Action
EnableIn is cleared	EnableOut is cleared.
EnableIn is set	The instruction executes.
	EnableOut is set.
postscan	No action taken.

cuu duong than cong. com

cuu duong than cong. com

**Example:** Calculate the arc cosine of *value* and place the result in *result*.

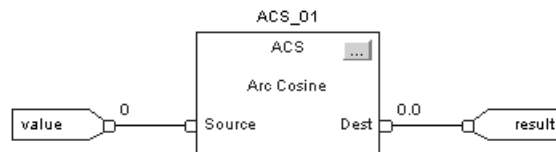
### Relay Ladder



### Structured Text

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

### Function Block



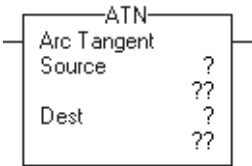
cuu duong than cong. com

cuu duong than cong. com

# 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	immediate	find the arc tangent of this value
	INT	tag	
	DINT		
	REAL		
Destination	SINT	tag	tag to store the result
	INT		
	DINT		
	REAL		



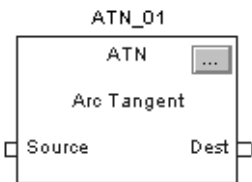
```
dest := ATAN(source);
```

## Structured Text

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

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

## Function Block



Operand	Type	Format	Description
ATN tag	FBD_MATH_ADVANCED	structure	ATN structure

**FBD\_MATH\_ADVANCED Structure**

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

**Description:** The resulting value in the Destination is always greater than or equal to  $-\pi/2$  and less than or equal to  $\pi/2$  (where  $\pi = 3.141593$ ).

**Arithmetic Status Flags:** Arithmetic status flags are affected.

**Fault Conditions:** none

**Execution:** [cuu duong than cong. com](https://fb.com/tailieudientucntt)

**Relay Ladder**

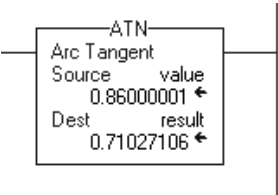
Condition	Action
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	The controller calculates the arc tangent of the Source and places the result in the Destination. The rung-condition-out is set to true.
postscan	The rung-condition-out is set to false.

**Function Block**

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

**Example:** Calculate the arc tangent of *value* and place the result in *result*.

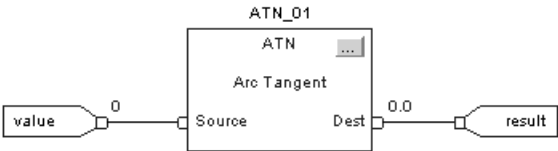
**Relay Ladder**



**Structured Text**

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

**Function Block**



cuu duong than cong. com

cuu duong than cong. com

**Notes:**

cuu duong than cong. com

cuu duong than cong. com

cuu duong than cong. com

cuu duong than cong. com



## Advanced Math Instructions

### (LN, LOG, XPY)

### Introduction

The advanced math instructions include these instructions:

If You Want To	Use This Instruction	Available In These Languages	See Page
Take the natural log of a value.	LN	relay ladder	546
		structured text	
		function block	
Take the log base 10 of a value.	LOG	relay ladder	549
		structured text	
		function block	
Raise a value to the power of another value.	XPY	relay ladder	552
		structured text <sup>(1)</sup>	
		function block	

<sup>(1)</sup> There is no equivalent structured text instruction. Use the operator in an expression.

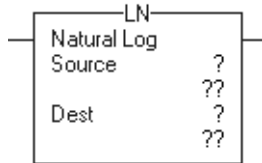
You can mix data types, but loss of accuracy and rounding error might occur and the instruction takes more time to execute. Check the S:V bit to see whether the result was truncated.

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	immediate	find the natural log of this value
	INT	tag	
	DINT		
	REAL		
Destination	SINT	tag	tag to store the result
	INT		
	DINT		
	REAL		

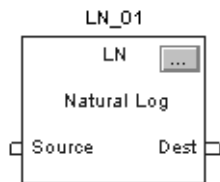


```
dest := LN(source);
```

## Structured Text

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

See Appendix B 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

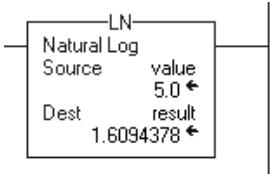
Condition	Action
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	The controller calculates the natural log of the Source and places the result in the Destination. The rung-condition-out is set to true.
postscan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



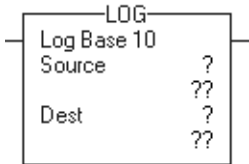
cuu duong than cong. com

cuu duong than cong. com

# 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	immediate	find the log of this value
	INT	tag	
	DINT		
	REAL		
Destination	SINT	tag	tag to store the result
	INT		
	DINT		
	REAL		

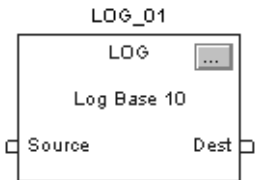


```
dest := LOG(source);
```

## Structured Text

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

See Appendix B 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**

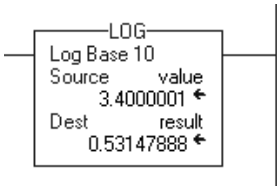
Condition	Action
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	The controller calculates the log of the Source and places the result in the Destination.  The rung-condition-out is set to true.
postscan	The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

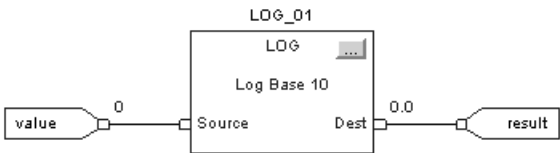
**Relay Ladder**



**Structured Text**

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

**Function Block**



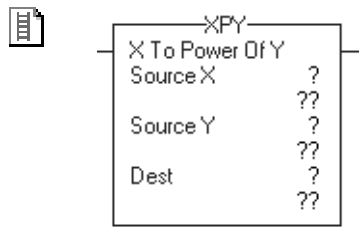
cuu duong than cong. com

cuu duong than cong. com

# 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	immediate	base value
	INT	tag	
	DINT		
	REAL		
Source Y	SINT	immediate	exponent
	INT	tag	
	DINT		
	REAL		
Destination	SINT	tag	tag to store the result
	INT		
	DINT		
	REAL		

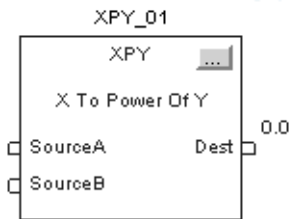


```
dest := sourceX ** sourceY;
```

## Structured Text

Use two, adjacent multiply signs “\*\*” as an operator within an expression. This expression takes *sourceX* to the power of *sourceY* and stores the result in *dest*.

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



## Function Block

Operand	Type	Format	Description
XPY tag	FBD_MATH	structure	XPY structure



## FBD\_MATH Structure

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

**Description:** If Source X is negative, Source Y must be an integer value or a minor fault will occur.

The XPY instruction uses this algorithm:  $Destination = X^{**}Y$

The controller evaluates  $x^0=1$  and  $0^x=0$ .

**Arithmetic Status Flags:** Arithmetic status flags are affected.

### Fault Conditions:

A Minor Fault Will Occur If	Fault Type	Fault Code
Source X is negative and Source Y is not an integer value	4	4

### Execution:



### Relay Ladder

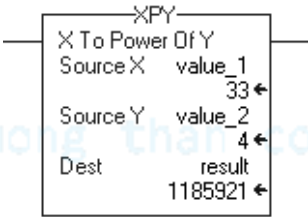
Condition	Action
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	The controller takes Source X to the power of Source Y and places the result in the Destination. The rung-condition-out is set to true.
postscan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

Function Block

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

**Example:** The XPY instruction takes *value\_1* to the power of *value\_2* and places the result in *result*.

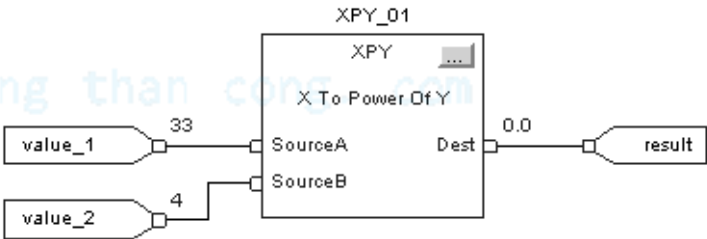
Relay Ladder



Structured Text

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

Function Block



## Math Conversion Instructions

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

### Introduction

The math conversion instructions convert values.

If You Want To	Use This Instruction	Available In These Languages	See Page
Convert radians to degrees.	DEG	relay ladder	556
		structured text	
		function block	
Convert degrees to radians.	RAD	relay ladder	559
		structured text	
		function block	
Convert an integer value to a BCD value.	TOD	relay ladder	562
		function block	
Convert a BCD value to an integer value.	FRD	relay ladder	565
		function block	
Remove the fractional part of a value	TRN	relay ladder	567
	TRUNC <sup>(1)</sup>	structured text	
		function block	

<sup>(1)</sup> Structured text only.

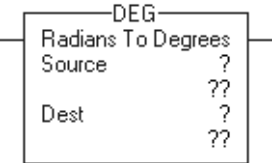
You can mix data types, but loss of accuracy and rounding error might occur and the instruction takes more time to execute. Check the S:V bit to see whether the result was truncated.

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	immediate	value to convert to degrees
	INT	tag	
	DINT		
	REAL		
Destination	SINT	tag	tag to store the result
	INT		
	DINT		
	REAL		

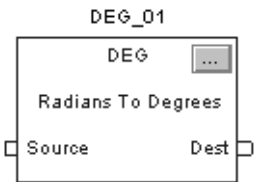


```
dest := DEG(source);
```

## Structured Text

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

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



## Function Block

Operand	Type	Format	Description
DEG tag	FBD_MATH_ADVANCED	structure	DEG structure

### FBD\_MATH\_ADVANCED Structure

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

**Description:** The DEG instruction uses this algorithm:  
 $\text{Source} \times 180 / \pi$  (where  $\pi = 3.141593$ )

**Arithmetic Status Flags:** Arithmetic status flags are affected.

**Fault Conditions:** none

**Execution:** [fb.com/tailieudientucntt](https://fb.com/tailieudientucntt)



### Relay Ladder

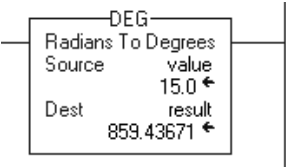
Condition	Action
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	The controller converts the Source to degrees and places the result in the Destination. The rung-condition-out is set to true.
postscan	The rung-condition-out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

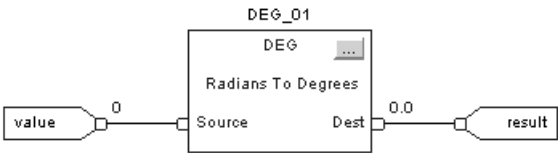
**Relay Ladder**



**Structured Text**

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

**Function Block**



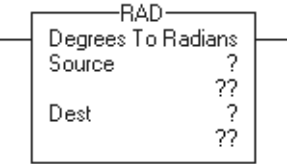
cuu duong than cong. com

cuu duong than cong. com

# 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	immediate	value to convert to radians
	INT	tag	
	DINT		
	REAL		
Destination	SINT	tag	tag to store the result
	INT		
	DINT		
	REAL		

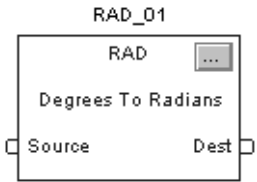


```
dest := RAD(source);
```

## Structured Text

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

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



## Function Block

Operand	Type	Format	Description
RAD tag	FBD_MATH_ADVANCED	structure	RAD structure

**FBD\_MATH\_ADVANCED Structure**

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

**Description:** The RAD instruction uses this algorithm:  
 $\text{Source} \times \pi / 180$  (where  $\pi = 3.141593$ )

**Arithmetic Status Flags:** Arithmetic status flags are affected.

**Fault Conditions:** none

**Execution:** [cuuduongthancong.com](http://cuuduongthancong.com)

**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.

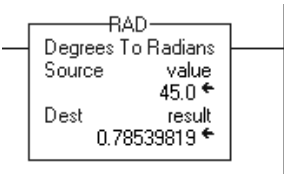
**Function Block**

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*.

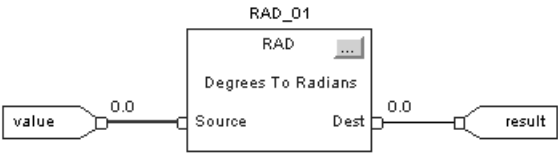
**Relay Ladder**



**Structured Text**

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

**Function Block**



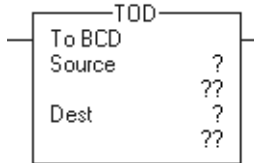
cuu duong than cong. com

cuu duong than cong. com

## Convert to BCD (TOD)

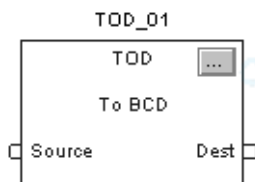
The TOD instruction converts a decimal value ( $0 \leq \text{Source} \leq 99,999,999$ ) to a BCD value and stores the result in the Destination.

### Operands:



### Relay Ladder

Operand	Type	Format	Description
Source	SINT	immediate	value to convert to decimal
	INT	tag	
	DINT		A SINT or INT tag converts to a DINT value by zero-fill.
Destination	SINT	tag	stores the result
	INT		
	DINT		



### 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:

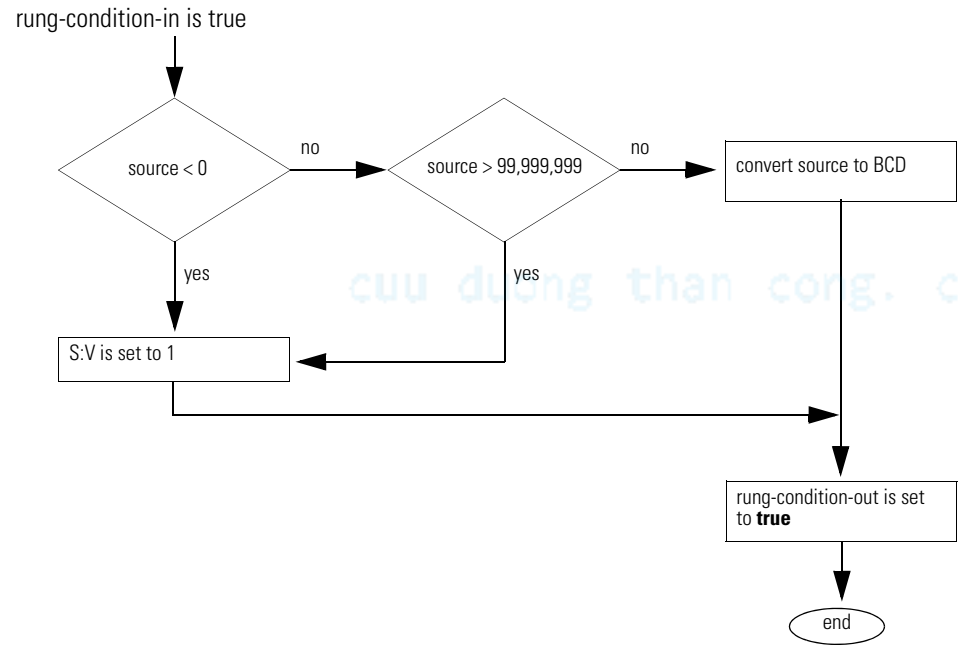
A Minor Fault Will Occur If	Fault Type	Fault Code
Source < 0	4	4

### 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.

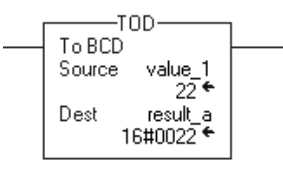
### Function Block

Condition	Action
prescan	No action taken.
instruction first scan	No action taken.
instruction first run	No action taken.

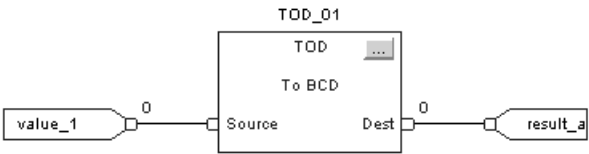
Condition	Action
EnableIn is cleared	EnableOut is cleared.
EnableIn is set	The instruction executes.
	EnableOut is set.
postscan	No action taken.

**Example:** The TOD instruction converts *value\_1* to a BCD value and places the result in *result\_a*.

**Relay Ladder**



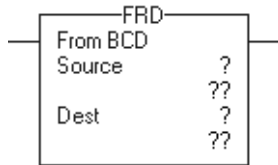
**Function Block**



## Convert to Integer (FRD)

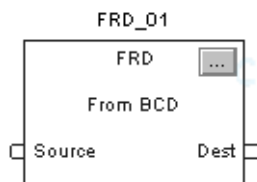
The FRD instruction converts a BCD value (Source) to a decimal value and stores the result in the Destination.

### Operands:



### Relay Ladder

Operand	Type	Format	Description
Source	SINT	immediate	value to convert to decimal
	INT	tag	
	DINT		A SINT or INT tag converts to a DINT value by zero-fill.
Destination	SINT	tag	stores the result
	INT		
	DINT		



### 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

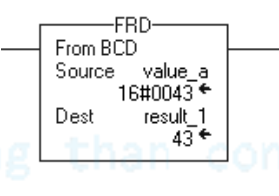
Condition	Action
prescan	The rung-condition-out is set to false.
rung-condition-in is false	The rung-condition-out is set to false.
rung-condition-in is true	The controller converts the Source to a decimal value and places the result in the Destination. The rung-condition-out is set to true.
postscan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

## Function Block

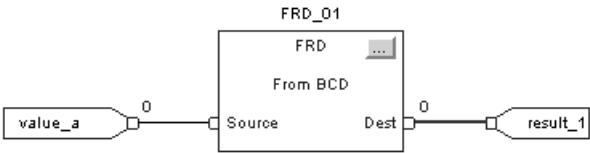
Condition	Action
prescan	No action taken.
instruction first scan	No action taken.
instruction first run	No action taken.
EnableIn is cleared	EnableOut is cleared.
EnableIn is set	The instruction executes. EnableOut is set.
postscan	No action taken.

**Example:** The FRD instruction converts *value\_a* to a decimal value and places the result in *result\_1*.

## Relay Ladder



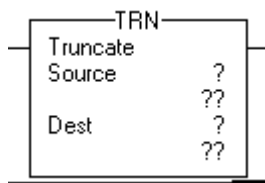
## Function Block



## 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	value to truncate
		tag	
Destination	SINT	tag	tag to store the result
	INT		
	DINT		
	REAL		



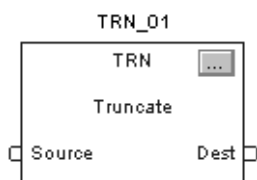
```
dest := TRUNC(source);
```

### Structured Text

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

See Appendix B 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.

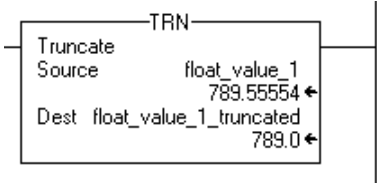
### Function Block

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*.

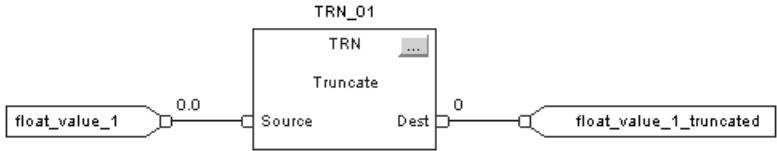
### Relay Ladder



### Structured Text

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

### Function Block



cuu duong than cong. com

cuu duong than cong. com

## Notes:

cuu duong than cong. com

cuu duong than cong. com

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

### Introduction

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

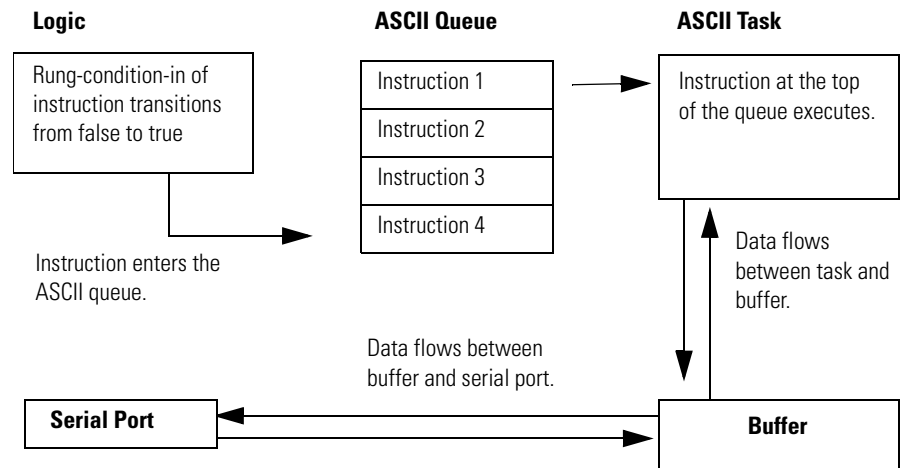
#### IMPORTANT

To use the ASCII serial port instructions, you must configure the serial port of the controller. See the *Logix5000 Controllers Common Procedures*, publication 1756-PM001.

If You Want To	For Example	Use This Instruction	Available In These Languages	See Page
determine when the buffer contains termination characters	check for data that contains termination characters	ABL	relay ladder structured text	575
count the characters in the buffer	check for the required number of characters before reading the buffer	ACB	relay ladder structured text	578
clear the buffer	<ul style="list-style-type: none"> <li>remove old data from the buffer at start-up</li> </ul>	ACL	relay ladder	581
clear out ASCII Serial Port instructions that are currently executing or are in the queue	<ul style="list-style-type: none"> <li>synchronize the buffer with a device</li> </ul>		structured text	
obtain the status of the serial port control lines	cause a modem to hang up	AHL	relay ladder	583
turn on or off the DTR signal			structured text	
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	587
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	591
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	595
send characters	send messages that use a variety of termination characters	AWT	relay ladder structured text	600

## Instruction Execution

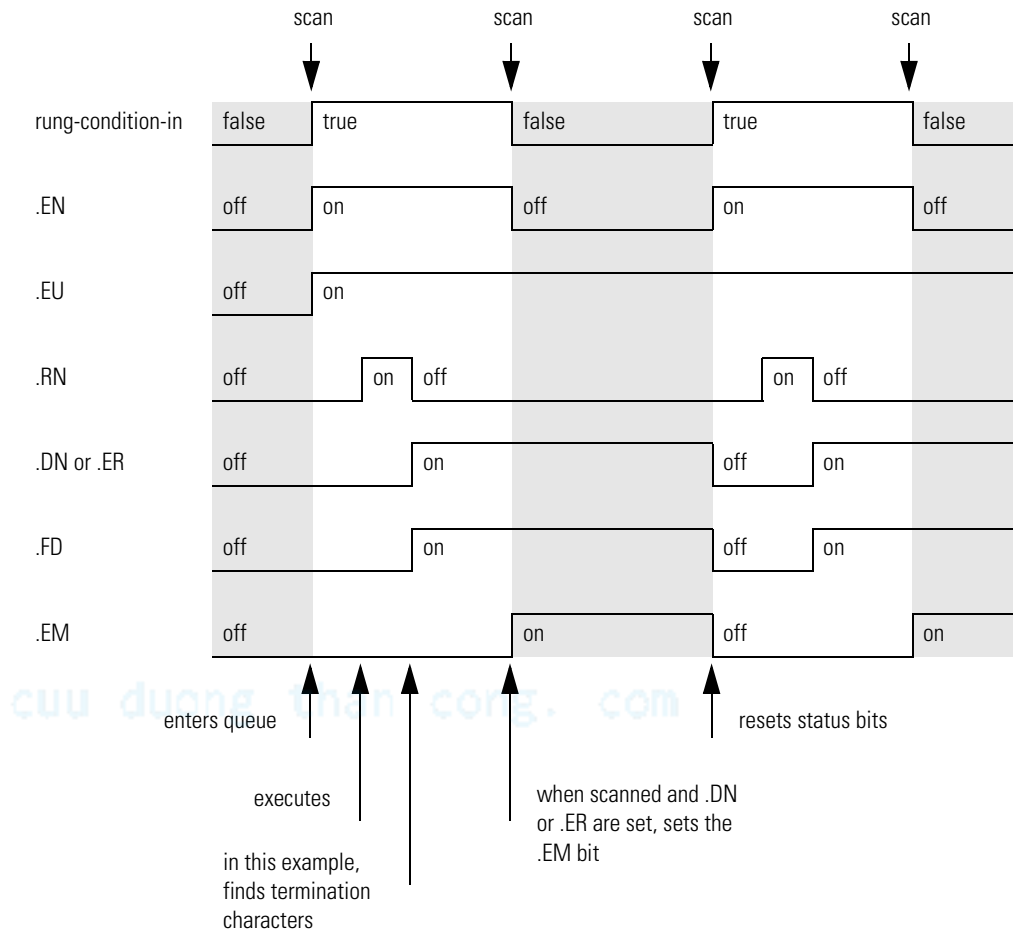
ASCII serial port instructions execute asynchronous to the scan of the logic:



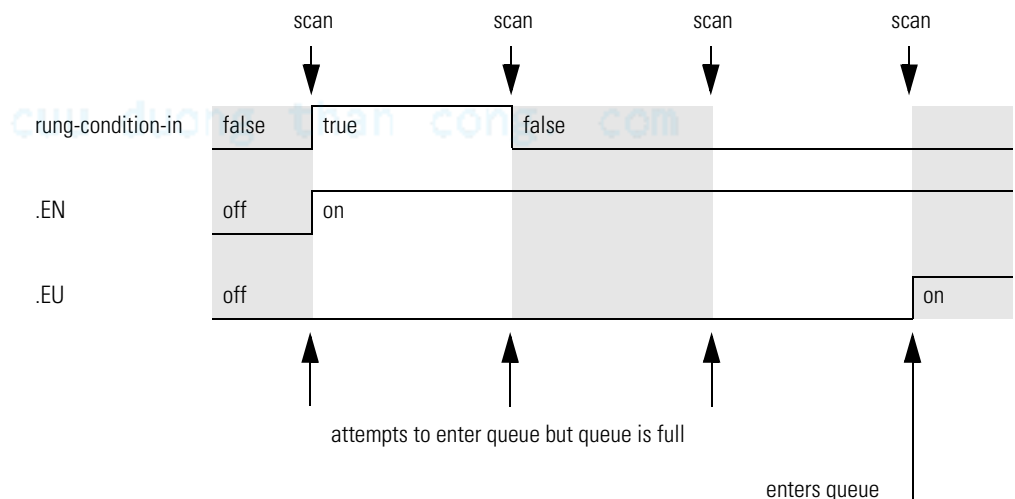
Each ASCII serial port instruction (except ACL) uses a SERIAL\_PORT\_CONTROL structure to perform the following functions:

- control the execution of the instruction
- provide status information about the instruction

The following timing diagram depicts the changes in the status bits as an ABL instruction tests the buffer for termination characters.



The ASCII queue holds up to 16 instructions. When the queue is full, an instruction tries to enter the queue on each subsequent scan of the instruction, as depicted below:



### ASCII Error Codes

If an ASCII serial port instruction fails to execute, the ERROR member of its SERIAL\_PORT\_CONTROL structure will contain one of the following hexadecimal error codes:

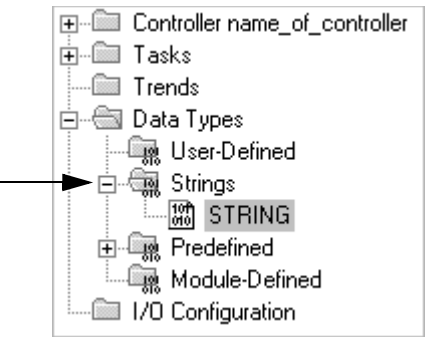
This Hex Code	Indicates That the
16#2	Modem went offline.
16#3	CTS signal was lost during communication.
16#4	Serial port was in system mode.
16#A	Before the instruction executed, the .UL bit was set. This prevents the execution of the instruction.
16#C	The controller changed from Run mode to Program mode. This stops the execution of an ASCII serial port instruction and clears the queue.
16#D	In the Controller Properties dialog box, User Protocol tab, the buffer size or echo mode parameters were changed and applied. This stops the execution of an ASCII serial port instruction and clears the queue.
16#E	ACL instruction executed.
16#F	Serial port configuration changed from User mode to System mode. This stops the execution of an ASCII serial port instruction and clears the ASCII serial port instruction queue.
16#51	The LEN value of the string tag is either negative or greater than the DATA size of the string tag.
16#54	The Serial Port Control Length is greater than the size of the buffer.
16#55	The Serial Port Control Length is either negative or greater than the size of the Source or Destination.

### String Data Types

You store ASCII characters in tags that use a string data type.

- You can use the default STRING data type. It stores up to 82 characters.
- You can create a new string data type that stores less or more characters.

To create a new string data type, see *Logix5000 Controllers Common Procedures*, publication 1756-PM001.



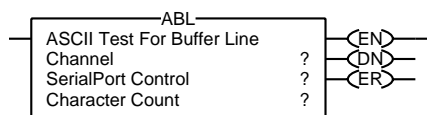
Each string data type contains the following members:

Name	Data Type	Description	Notes
LEN	DINT	number of characters in the string	<p>The LEN automatically updates to the new count of characters whenever you:</p> <ul style="list-style-type: none"> <li>• use the String Browser dialog box to enter characters</li> <li>• use instructions that read, convert, or manipulate a string</li> </ul> <p>The LEN shows the length of the current string. The DATA member may contain additional, old characters, which are not included in the LEN count.</p>
DATA	SINT array	ASCII characters of the string	<ul style="list-style-type: none"> <li>• To access the characters of the string, address the name of the tag. For example, to access the characters of the <i>string_1</i> tag, enter <i>string_1</i>.</li> <li>• Each element of the DATA array contains one character.</li> <li>• You can create new string data types that store less or more characters.</li> </ul>

## 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	0
		tag	
Serial Port	SERIAL_PORT_	tag	tag that controls the operation
Control	CONTROL		
Character Count	DINT	immediate	0
			During execution, displays the number of characters in the buffer, including the first set of termination characters.



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

### Structured Text

The operands are the same as those for the relay ladder ABL instruction. You access the Character Count value via the .POS member of the SERIAL\_PORT\_CONTROL structure.

**SERIAL\_PORT\_CONTROL Structure**

<b>Mnemonic</b>	<b>Data Type</b>	<b>Description</b>
.EN	BOOL	The enable bit indicates that the instruction is enabled.
.EU	BOOL	The queue bit indicates that the instruction entered the ASCII queue.
.DN	BOOL	The done bit indicates when the instruction is done, but it is asynchronous to the logic scan.
.RN	BOOL	The run bit indicates that the instruction is executing.
.EM	BOOL	The empty bit indicates that the instruction is done, but it is synchronous to the logic scan.
.ER	BOOL	The error bit indicates when the instruction fails (errors).
.FD	BOOL	The found bit indicates that the instruction found the termination character or characters.
.POS	DINT	The position determines the number of characters in the buffer, up to and including the first set of termination characters. The instruction only returns this number after it finds the termination character or characters.
.ERROR	DINT	The error contains a hexadecimal value that identifies the cause of an error.

**Description** The ABL instruction searches the buffer for the first set of termination characters. If the instruction finds the termination characters, it:

- sets the .FD bit
- counts the characters in the buffer up to and including the first set of termination characters

The Controller Properties dialog box, User Protocol tab, defines the ASCII characters that the instruction considers as the termination characters.

To program the ABL instruction, follow these guidelines:

1. Configure the serial port of the controller for user mode and define the characters that serve as the termination characters.
2. This is a transitional instruction:
  - In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.
  - In structured text, condition the instruction so that it only executes on a transition.

**Arithmetic Status Flags:** not affected

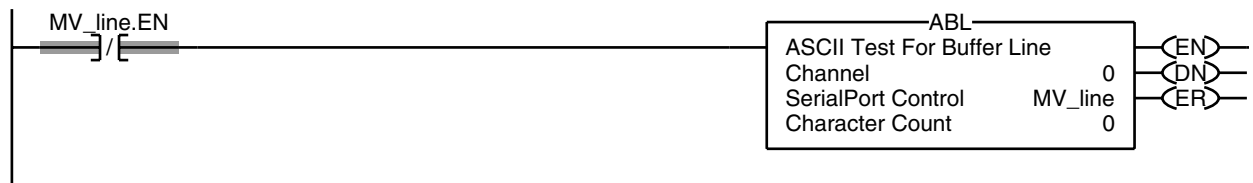
**Fault Conditions:** none



**Execution:**

Condition	Relay Ladder Action	Structured Text Action
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes when rung-condition-in toggles from cleared to set.  The rung-condition-out is set to true.	na
EnableIn is set	na	EnableIn is always set.  The instruction executes.
instruction execution	The instruction counts the characters in the buffer.  The .EN bit is set.  The remaining status bits, except .UL, are cleared.  The instruction attempts to enter the ASCII queue.	
postscan	The rung-condition-out is set to false.	No action taken.

**Example:** Continuously test the buffer for the termination characters.

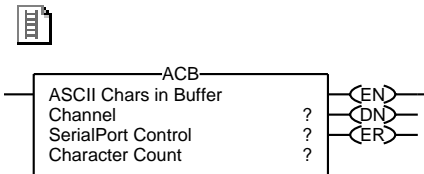
**Relay Ladder****Structured Text**

```
ABL(0,MV_line);
```

# ASCII Chars in Buffer (ACB)

The ACB instruction counts the characters in the buffer.

## Operands:



## Relay Ladder

Operand	Type	Format	Enter
Channel	DINT	immediate	0
		tag	
Serial Port	SERIAL_PORT_	tag	tag that controls the operation
Control	CONTROL		
Character Count	DINT	immediate	0

During execution, displays the number of characters in the buffer.



```

ACB(Channel
    SerialPortControl);

```

## Structured Text

The operands are the same as those for the relay ladder ACB instruction. However, you specify the Character Count value by accessing the .POS member of the SERIAL\_PORT\_CONTROL structure, rather than by including the value in the operand list.

## SERIAL\_PORT\_CONTROL Structure

Mnemonic	Data Type	Description
.EN	BOOL	The enable bit indicates that the instruction is enabled.
.EU	BOOL	The queue bit indicates that the instruction entered the ASCII queue.
.DN	BOOL	The done bit indicates when the instruction is done, but it is asynchronous to the logic scan.
.RN	BOOL	The run bit indicates that the instruction is executing.
.EM	BOOL	The empty bit indicates that the instruction is done, but it is synchronous to the logic scan.
.ER	BOOL	The error bit indicates when the instruction fails (errors).
.FD	BOOL	The found bit indicates that the instruction found a character.
.POS	DINT	The position determines the number of characters in the buffer, up to and including the first set of termination characters.
.ERROR	DINT	The error contains a hexadecimal value that identifies the cause of an error.

**Description:** The ACB instruction counts the characters in the buffer.

To program the ACB instruction, follow these guidelines:

1. Configure the serial port of the controller for user mode.
2. This is a transitional instruction:
  - In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.
  - In structured text, condition the instruction so that it only executes on a transition.

**Arithmetic Status Flags:** not affected

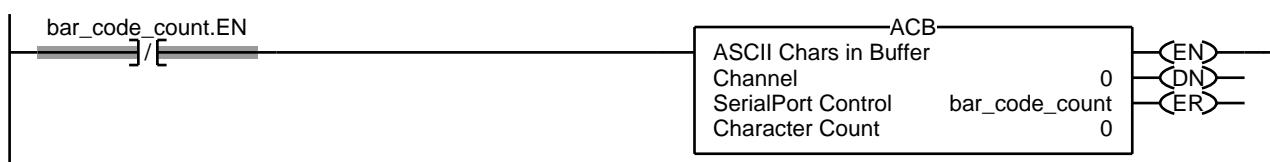
**Fault Conditions:** none

### Execution:

Condition	Relay Ladder Action	Structured Text Action
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes when rung-condition-in toggles from cleared to set.	na
	The rung-condition-out is set to true.	
EnableIn is set	na	EnableIn is always set.
		The instruction executes.
instruction execution	The instruction counts the characters in the buffer.	
	The .EN bit is set.	
	The remaining status bits, except .UL, are cleared.	
	The instruction attempts to enter the ASCII queue.	
postscan	The rung-condition-out is set to false.	No action taken.

**Example:** Continuously count the characters in the buffer.

### Relay Ladder



### Structured Text

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

cuu duong than cong. com

cuu duong than cong. com

## ASCII Clear Buffer (ACL)

The ACL instruction immediately clears the buffer and ASCII queue.

### Operands:



ACL	
ASCII Clear Buffer	?
Channel	?
Clear Serial Port Read	?
Clear Serial Port Write	?

### Relay Ladder

Operand	Type	Format	Enter
Channel	DINT	immediate	0
		tag	
Clear Serial Port Read	BOOL	immediate	To empty the buffer and remove ARD and ARL instructions from the queue, enter Yes.
		tag	
Clear Serial Port Write	BOOL	immediate	To remove AWA and AWT instructions from the queue, enter Yes.
		tag	



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

### Structured Text

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

**Description:** The ACL instruction immediately performs one or both of the following actions:

- clears the buffer of characters and clears the ASCII queue of read instructions
- clears the ASCII queue of write instructions

To program the ACL instruction, follow these guidelines:

1. Configure the serial port of the controller:

If Your Application	Then
uses ARD or ARL instructions	Select User mode
<i>does not</i> use ARD or ARL instructions	Select either System or User mode

2. To determine if an instruction was removed from the queue or aborted, examine the following of the appropriate instruction:
  - .ER bit is set
  - .ERROR member is 16#E

**Arithmetic Status Flags:** not affected

**Fault Conditions:** none

### Execution:

Condition	Relay Ladder Action	Structured Text Action
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes.  The rung-condition-out is set to true.	na
EnableIn is set	na	EnableIn is always set.  The instruction executes.
instruction execution	The instruction clears the specified instructions and buffer(s).	
postscan	The rung-condition-out is set to false.	No action taken.

**Example:** When the controller enters Run mode, clear the buffer and the ASCII queue.

### Relay Ladder



### Structured Text

```

osri_1.InputBit := S:FS;
OSRI(osri_1);

IF (osri_1.OutputBit) THEN
    ACL(0,0,1);
END_IF;
```

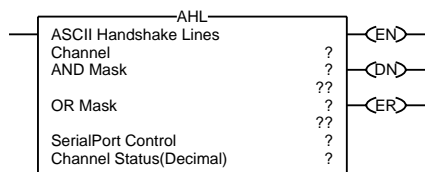
## ASCII Handshake Lines (AHL)

The AHL instruction obtains the status of control lines and turns on or off the DTR and RTS signals.

### Operands:



### Relay Ladder



Operand	Type	Format	Enter
Channel	DINT	immediate	0
		tag	
ANDMask	DINT	immediate	Refer to the description.
		tag	
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



```
AHL(Channel, ANDMask, ORMask,
      SerialPortControl);
```

### Structured Text

The operands are the same as those for the relay ladder AHL instruction. However, you specify the Channel Status value by accessing the .POS member of the SERIAL\_PORT\_CONTROL structure, rather than by including the value in the operand list.

**SERIAL\_PORT\_CONTROL Structure**

Mnemonic	Data Type	Description
.EN	BOOL	The enable bit indicates that the instruction is enabled.
.EU	BOOL	The queue bit indicates that the instruction entered the ASCII queue.
.DN	BOOL	The done bit indicates when the instruction is done, but it is asynchronous to the logic scan.
.RN	BOOL	The run bit indicates that the instruction is executing.
.EM	BOOL	The empty bit indicates that the instruction is done, but it is synchronous to the logic scan.
.ER	BOOL	The error bit indicates when the instruction fails (errors).
.FD	BOOL	The found bit does not apply to this instruction.
.POS	DINT	The position stores the status of the control lines.
.ERROR	DINT	The error contains a hexadecimal value that identifies the cause of an error.

**Description:** The AHL instruction can:

- obtain the status of the control lines of the serial port
- turn on or off the data terminal ready (DTR) signal
- turn on or off the request to send signal (RTS)

To program the AHL instruction, follow these guidelines:

1. Configure the serial port of the controller:

If Your Application	Then
uses ARD or ARL instructions	Select User mode
<i>does not</i> use ARD or ARL instructions	Select either System or User mode

2. Use the following table to select the correct values for the ANDMask and ORMask operands:

To Turn DTR	And Turn RTS:	Enter This ANDMask Value	And Enter This ORMask Value
off	off	3	0
	on	1	2
	unchanged	1	0
on	off	2	1
	on	0	3
	unchanged	0	1
unchanged	off	2	0
	on	0	2
	unchanged	0	0



3. This is a transitional instruction:

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

**Arithmetic Status Flags:** not affected

### Fault Conditions:

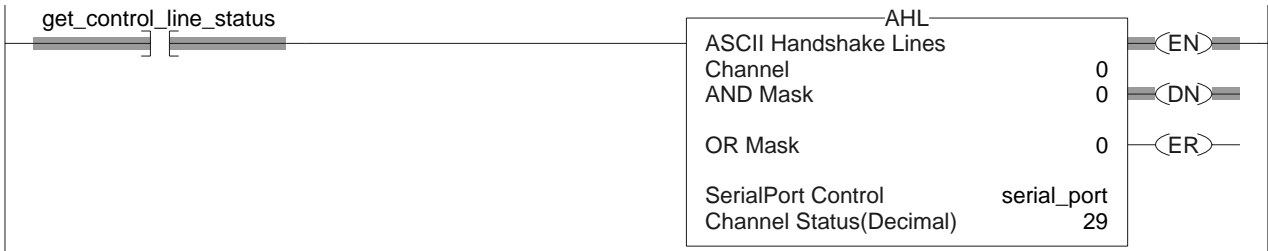
Type	Code	Cause	Recovery Method
4	57	The AHL instruction failed to execute because the serial port is set to no handshaking.	Either: <ul style="list-style-type: none"> <li>• Change the Control Line setting of the serial port.</li> <li>• Delete the AHL instruction.</li> </ul>

### Execution:

Condition	Relay Ladder Action	Structured Text Action
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes when rung-condition-in toggles from cleared to set.  The rung-condition-out is set to true.	na
EnableIn is set	na	EnableIn is always set.  The instruction executes.
instruction execution	The instruction obtains the control line status and turns on or off DTR and RTS signals.  The .EN bit is set.  The remaining status bits, except .UL, are cleared.  The instruction attempts to enter the ASCII queue.	
postscan	The rung-condition-out is set to false.	No action taken.

**Example:** When *get\_control\_line\_status* becomes set, obtain the status of the control lines of the serial port and store the status in the Channel Status operand. To view the status of a specific control line, monitor the SerialPortControl tag and expand the POS member.

Relay Ladder



Structured Text

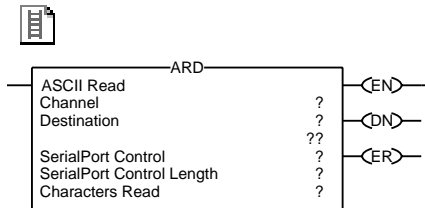
```
osri_1.InputBit := get_control_line_status;  
OSRI(osri_1);  
  
IF (osri_1.OutputBit) THEN  
    AHL(0,0,0,serial_port);  
END_IF;
```

## ASCII Read (ARD)

The ARD instruction removes characters from the buffer and stores them in the Destination.

### Operands:

### Relay Ladder



Operand	Type	Format	Enter	Notes
Channel	DINT	immediate	0	
		tag		
Destination	<b>string</b>	tag	tag into which the characters are moved (read):	<ul style="list-style-type: none"> <li>If you want to compare, convert, or manipulate the characters, use a string data type.</li> <li>String data types are: <ul style="list-style-type: none"> <li>default STRING data type</li> <li>any new string data type that you create</li> </ul> </li> </ul>
	SINT			
	INT		<ul style="list-style-type: none"> <li>For a string data type, enter the name of the tag.</li> </ul>	
	DINT		<ul style="list-style-type: none"> <li>For a SINT, INT, or DINT array, enter the first element of the array.</li> </ul>	
Serial Port	SERIAL_PORT_	tag	tag that controls the operation	
Control	CONTROL			
Serial Port	DINT	immediate	number of characters to move to the destination (read)	<ul style="list-style-type: none"> <li>The Serial Port Control Length must be less than or equal to the size of the Destination.</li> <li>If you want to set the Serial Port Control Length equal to the size of the Destination, enter 0.</li> </ul>
Control Length				
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**

<b>Mnemonic</b>	<b>Data Type</b>	<b>Description</b>
.EN	BOOL	The enable bit indicates that the instruction is enabled.
.EU	BOOL	The queue bit indicates that the instruction entered the ASCII queue.
.DN	BOOL	The done bit indicates when the instruction is done, but it is asynchronous to the logic scan.
.RN	BOOL	The run bit indicates that the instruction is executing.
.EM	BOOL	The empty bit indicates that the instruction is done, but it is synchronous to the logic scan.
.ER	BOOL	The error bit indicates when the instruction fails (errors).
.FD	BOOL	The found bit does not apply to this instruction.
.LEN	DINT	The length indicates the number of characters to move to the destination (read).
.POS	DINT	The position displays the number of characters that were read.
.ERROR	DINT	The error contains a hexadecimal value that identifies the cause of an error.

**Description:** The ARD instruction removes the specified number of characters from the buffer and stores them in the Destination.

- The ARD instruction continues to execute until it removes the specified number of characters (Serial Port Control Length).
- While the ARD instruction is executing, no other ASCII Serial Port instruction executes.

To program the ARD instruction, follow these guidelines:

1. Configure the serial port of the controller for user mode.
2. Use the results of an ACB instruction to trigger the ARD instruction. This prevents the ARD instruction from holding up the ASCII queue while it waits for the required number of characters.
3. This is a transitional instruction:
  - In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.
  - In structured text, condition the instruction so that it only executes on a transition. See .
4. To trigger a subsequent action when the instruction is done, examine the EM bit.

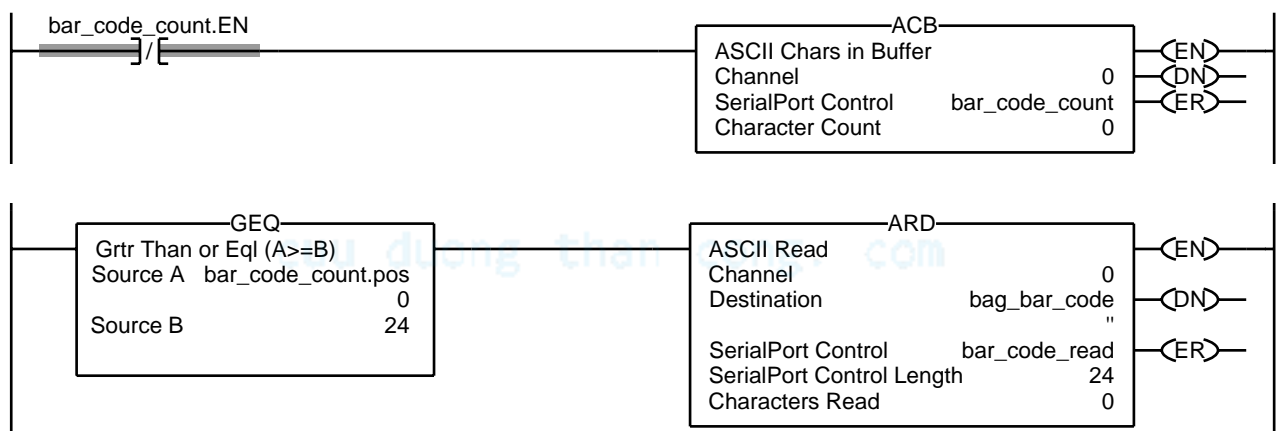
**Arithmetic Status Flags:** not affected

**Fault Conditions:** none

**Execution:**

Condition	Relay Ladder Action	Structured Text Action
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes when rung-condition-in toggles from cleared to set.  The rung-condition-out is set to true.	na
EnableIn is set	na	EnableIn is always set.  The instruction executes.
instruction execution	The instruction removes characters from the buffer and stores them in the destination.  The .EN bit is set.  The remaining status bits, except .UL, are cleared.  The instruction attempts to enter the ASCII queue.	
postscan	The rung-condition-out is set to false.	No action taken.

**Example:** A bar code reader sends bar codes to the serial port (channel 0) of the controller. Each bar code contains 24 characters. To determine when the controller receives a bar code, the ACB instruction continuously counts the characters in the buffer. When the buffer contains at least 24 characters, the controller has received a bar code. The ARD instruction moves the bar code to the DATA member of the *bag\_bar\_code* tag, which is a string.

**Relay Ladder****Structured Text**

```
ACB(0, bar_code_count);

IF bar_code_count.POS >= 24 THEN
```

```
bar_code_read.LEN := 24;  
ARD(0,bag_bar_code,bar_code_read);  
END_IF;
```

cuu duong than cong. com

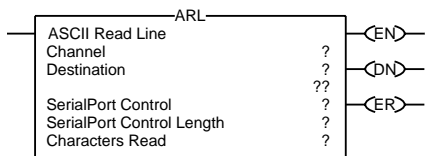
cuu duong than cong. com

## 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	0	
		tag		
Destination	<b>string</b>	tag	tag into which the characters are moved (read):	<ul style="list-style-type: none"> <li>If you want to compare, convert, or manipulate the characters, use a string data type.</li> <li>String data types are: <ul style="list-style-type: none"> <li>default STRING data type</li> <li>any new string data type that you create</li> </ul> </li> </ul>
	SINT			
	INT		<ul style="list-style-type: none"> <li>For a string data type, enter the name of the tag.</li> </ul>	
	DINT		<ul style="list-style-type: none"> <li>For a SINT, INT, or DINT array, enter the first element of the array.</li> </ul>	
Serial Port	SERIAL_PORT_	tag	tag that controls the operation	
Control	CONTROL			
Serial Port Control Length	DINT	immediate	maximum number of characters to read if no termination characters are found	<ul style="list-style-type: none"> <li>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.</li> <li>The Serial Port Control Length must be less than or equal to the size of the Destination.</li> <li>If you want to set the Serial Port Control Length equal to the size of the Destination, enter 0.</li> </ul>
Characters Read	DINT	immediate	0	During execution, displays the number of characters that were read.



#### Structured Text

```
ARL(Channel, Destination,
     SerialPortControl);
```

The operands are the same as those for the relay ladder ARL instruction. However, you specify the Serial Port Control Length and the Characters Read values by accessing the .LEN and .POS

members of the SERIAL\_PORT\_CONTROL structure, rather than by including the values in the operand list.

### SERIAL\_PORT\_CONTROL Structure

Mnemonic	Data Type	Description
.EN	BOOL	The enable bit indicates that the instruction is enabled.
.EU	BOOL	The queue bit indicates that the instruction entered the ASCII queue.
.DN	BOOL	The done bit indicates when the instruction is done, but it is asynchronous to the logic scan.
.RN	BOOL	The run bit indicates that the instruction is executing.
.EM	BOOL	The empty bit indicates that the instruction is done, but it is synchronous to the logic scan.
.ER	BOOL	The error bit indicates when the instruction fails (errors).
.FD	BOOL	The found bit does not apply to this instruction.
.LEN	DINT	The length indicates the maximum number of characters to move to the destination (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 as follows:

- The ARL instruction continues to execute until it removes either the:
  - first set of termination characters
  - specified number of characters (Serial Port Control Length)
- While the ARL instruction is executing, no other ASCII serial port instruction executes.

To program the ARL instruction, follow these guidelines:

1. Configure the serial port of the controller:
  - a. Select User mode.
  - b. Define the characters that serve as the termination characters.
2. Use the results of an ABL instruction to trigger the ARL instruction. This prevents the ARL instruction from holding up the ASCII queue while it waits for the termination characters.
3. This is a transitional instruction:
  - In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.
  - In structured text, condition the instruction so that it only executes on a transition. See .



4. To trigger a subsequent action when the instruction is done, examine the EM bit.

**Arithmetic Status Flags:** not affected

**Fault Conditions:** none

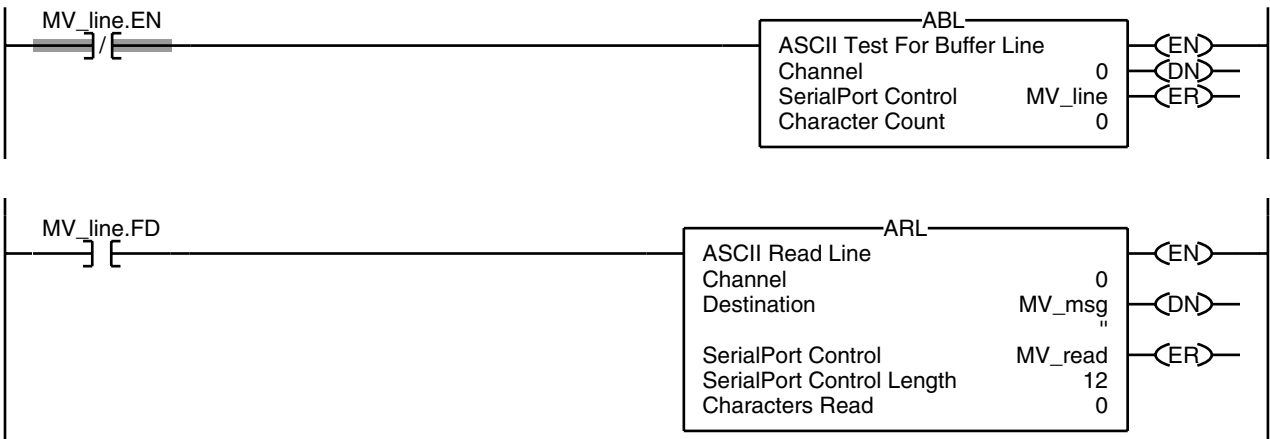
**Execution:**

Condition	Relay Ladder Action	Structured Text Action
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes when rung-condition-in toggles from cleared to set.	na
	The rung-condition-out is set to true.	
EnableIn is set	na	EnableIn is always set.
		The instruction executes.
instruction execution	The instruction removes the specified characters from the buffer and stores them in the destination.  The .EN bit is set.  The remaining status bits, except .UL, are cleared.  The instruction attempts to enter the ASCII queue.	
postscan	The rung-condition-out is set to false.	No action taken.

**Example:** Continuously test the buffer for a message from a MessageView terminal. Since each message ends in a carriage return (\$r), the carriage return is configured as the termination character in the Controller Properties dialog box, User Protocol tab. When the ABL finds a carriage return, it sets the FD bit.

When the ABL instruction finds the carriage return (*MV\_line.FD* is set), the controller has received a complete message. The ARL instruction removes the characters from the buffer, up to and including the carriage return, and places them in the DATA member of the *MV\_msg* tag, which is a string.

### Relay Ladder



### Structured Text

```
ABL(0,MV_line);
```

```
osri_1.InputBit := MVLine.FD;
OSRI(osri_1);
```

```
IF (osri_1.OutputBit) THEN
```

```
    mv_read.LEN := 12;
```

```
    ARL(0,MV_msg,MV_read);
```

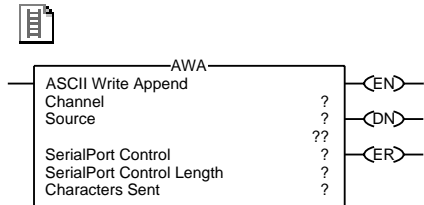
```
END_IF;
```

## ASCII Write Append (AWA)

The AWA instruction sends a specified number of characters of the Source tag to a serial device and appends either one or two predefined characters.

### Operands:

#### Relay Ladder



Operand	Type	Format	Enter	Notes
Channel	DINT	immediate	0	
		tag		
Source	<b>string</b>	tag	tag that contains the characters to send:	<ul style="list-style-type: none"> <li>If you want to compare, convert, or manipulate the characters, use a string data type.</li> <li>String data types are: <ul style="list-style-type: none"> <li>default STRING data type</li> <li>any new string data type that you create</li> </ul> </li> </ul>
	SINT			
	INT			
	DINT			
Serial Port	SERIAL_PORT_	tag	tag that controls the operation	
Control	CONTROL			
Serial Port Control Length	DINT	immediate	number of characters to send	<ul style="list-style-type: none"> <li>The Serial Port Control Length must be less than or equal to the size of the Source.</li> <li>If you want to set the Serial Port Control Length equal to the number of characters in the Source, enter 0.</li> </ul>
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**

<b>Mnemonic</b>	<b>Data Type</b>	<b>Description</b>
.EN	BOOL	The enable bit indicates that the instruction is enabled.
.EU	BOOL	The queue bit indicates that the instruction entered the ASCII queue.
.DN	BOOL	The done bit indicates when the instruction is done, but it is asynchronous to the logic scan.
.RN	BOOL	The run bit indicates that the instruction is executing.
.EM	BOOL	The empty bit indicates that the instruction is done, but it is synchronous to the logic scan.
.ER	BOOL	The error bit indicates when the instruction fails (errors).
.FD	BOOL	The found bit does not apply to this instruction.
.LEN	DINT	The length indicates the number of characters to send.
.POS	DINT	The position displays the number of characters that were sent.
.ERROR	DINT	The error contains a hexadecimal value that identifies the cause of an error.

**Description:** The AWA instruction:

- sends the specified number of characters (Serial Port Control Length) of the Source tag to the device that is connected to the serial port of the controller
- adds to the end of the characters (appends) either one or two characters that are defined in the Controller Properties dialog box, User Protocol tab

To program the AWA instruction, follow these guidelines:

**1.** Configure the serial port of the controller:

- a. Does your application also include ARD or ARL instructions?

<b>If</b>	<b>Then</b>
Yes	Select User mode
No	Select either System or User mode

- b. Define the characters to append to the data.

**2.** This is a transitional instruction:

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

3. Each time the instruction executes, do you always send the same number of characters?

If	Then
Yes	In the Serial Port Control Length, enter the number of characters to send.
No	Before the instruction executes, set the LEN member of the Source tag to the LEN member of the Serial Port Control tag.

**Arithmetic Status Flags:** not affected

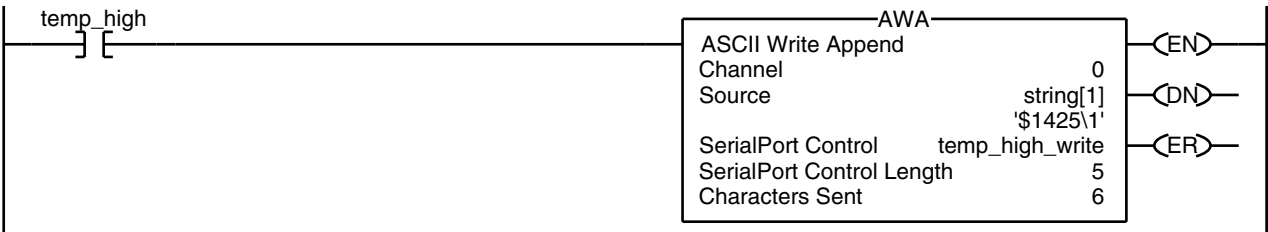
**Fault Conditions:** none

**Execution:**

Condition	Relay Ladder Action	Structured Text Action
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes when rung-condition-in toggles from cleared to set. The rung-condition-out is set to true.	na
EnableIn is set	na	EnableIn is always set. The instruction executes.
instruction execution	The instruction sends a specified number of characters and appends either one or two predefined characters. The .EN bit is set. The remaining status bits, except .UL, are cleared. The instruction attempts to enter the ASCII queue.	
postscan	The rung-condition-out is set to false.	No action taken.

**Example 1:** When the temperature exceeds the high limit (*temp\_high* is set), the AWA instruction sends a message to a MessageView terminal that is connected to the serial port of the controller. The message contains five characters from the DATA member of the *string[1]* tag, which is a string. (The *\$14* counts as one character. It is the hex code for the Ctrl-T character.) The instruction also sends (appends) the characters defined in the controller properties. In this example, the AWA instruction sends a carriage return (\$0D), which marks the end of the message.

### Relay Ladder



### Structured Text

```
IF temp_high THEN

    temp_high_write.LEN := 5;

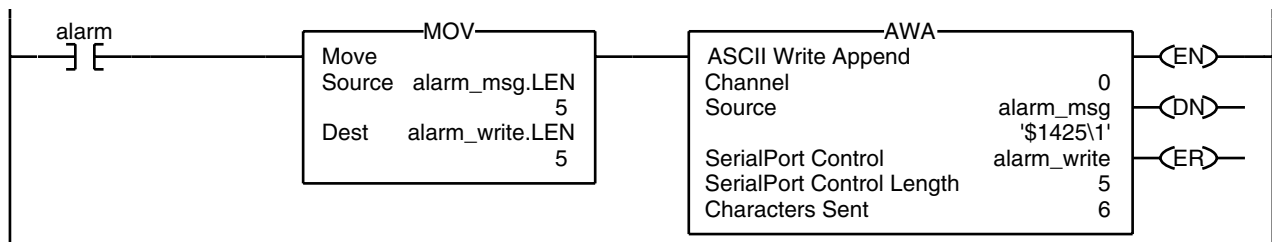
    AWA(0,string[1],temp_high_write);

    temp_high := 0;

END_IF;
```

**Example 2:** When *alarm* is set, the AWA instruction sends the specified number of characters in *alarm\_msg* and appends a termination character (s). Because the number of characters in *alarm\_msg* varies, the rung first moves the length of the string (*alarm\_msg.LEN*) to the Serial Port Control Length of the AWA instruction (*alarm\_write.LEN*). In *alarm\_msg*, the \$14 counts as one character. It is the hex code for the Ctrl-T character.

### Relay Ladder



### Structured Text

```
osri_1.InputBit := alarm;
OSRI(osri_1);

IF (osri_1.OutputBit) THEN

    alarm_write.LEN := alarm_msg.LEN;

    AWA(0,alarm_msg,alarm_write);

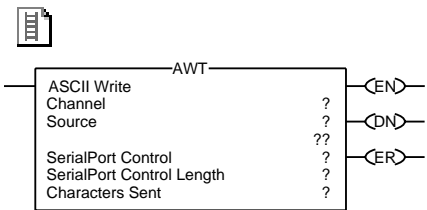
END_IF;
```

# ASCII Write (AWT)

The AWT instruction sends a specified number of characters of the Source tag to a serial device.

## Operands:

### Relay Ladder



Operand	Type	Format	Enter	Notes
Channel	DINT	immediate	0	
		tag		
Source	<b>string</b>	tag	tag that contains the characters to send:	<ul style="list-style-type: none"><li>If you want to compare, convert, or manipulate the characters, use a string data type.</li><li>String data types are:<ul style="list-style-type: none"><li>default STRING data type</li><li>any new string data type that you create</li></ul></li></ul>
	SINT		<ul style="list-style-type: none"><li>For a string data type, enter the name of the tag.</li></ul>	
	INT		<ul style="list-style-type: none"><li>For a SINT, INT, or DINT array, enter the first element of the array.</li></ul>	
	DINT			
Serial Port	SERIAL_PORT_	tag	tag that controls the operation	
Control	CONTROL			
Serial Port Control Length	DINT	immediate	number of characters to send	<ul style="list-style-type: none"><li>The Serial Port Control Length must be less than or equal to the size of the Source.</li><li>If you want to set the Serial Port Control Length equal to the number of characters in the Source, enter 0.</li></ul>
Characters Sent	DINT	immediate	0	During execution, displays the number of characters that were sent.

### Structured Text

```
AWT ( Channel , Source ,  
      SerialPortControl );
```

The operands are the same as those for the relay ladder AWT instruction. However, you specify the Serial Port Control Length and the Characters Sent values by accessing the .LEN and .POS members of the SERIAL\_PORT\_CONTROL structure, rather than by including the values in the operand list



**SERIAL\_PORT\_CONTROL Structure**

Mnemonic	Data Type	Description
.EN	BOOL	The enable bit indicates that the instruction is enabled.
.EU	BOOL	The queue bit indicates that the instruction entered the ASCII queue.
.DN	BOOL	The done bit indicates when the instruction is done, but it is asynchronous to the logic scan.
.RN	BOOL	The run bit indicates that the instruction is executing.
.EM	BOOL	The empty bit indicates that the instruction is done, but it is synchronous to the logic scan.
.ER	BOOL	The error bit indicates when the instruction fails (errors).
.FD	BOOL	The found bit does not apply to this instruction.
.LEN	DINT	The length indicates the number of characters to send.
.POS	DINT	The position displays the number of characters that were sent.
.ERROR	DINT	The error contains a hexadecimal value that identifies the cause of an error.

**Description:** The AWT instruction sends the specified number of characters (Serial Port Control Length) of the Source tag to the device that is connected to the serial port of the controller.

To program the AWT instruction, follow these guidelines:

1. Configure the serial port of the controller:

If Your Application	Then
uses ARD or ARL instructions	Select User mode
<i>does not</i> use ARD or ARL instructions	Select either System or User mode

2. This is a transitional instruction:

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

3. Each time the instruction executes, do you always send the same number of characters?

If	Then
Yes	In the Serial Port Control Length, enter the number of characters to send.
No	Before the instruction executes, move the LEN member of the Source tag to the LEN member of the Serial Port Control tag.

**Arithmetic Status Flags:** not affected

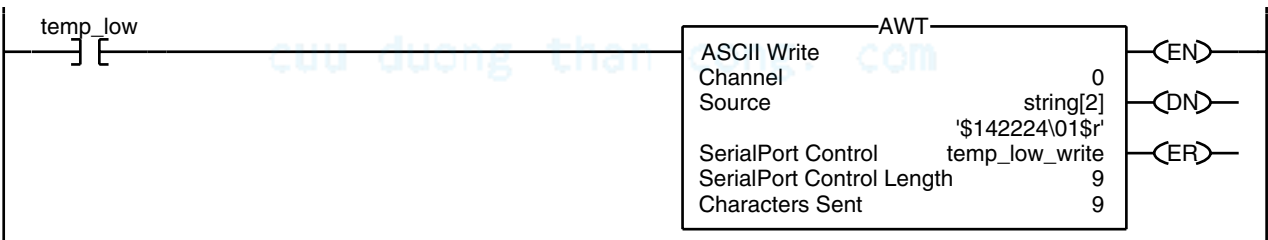
**Fault Conditions:** none

**Execution:**

Condition	Relay Ladder Action	Structured Text Action
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes when rung-condition-in toggles from cleared to set.  The rung-condition-out is set to true.	na
EnableIn is set	na	EnableIn is always set.  The instruction executes.
instruction execution	The instruction sends a specified number of characters.  The .EN bit is set.  The remaining status bits, except .UL, are cleared.  The instruction attempts to enter the ASCII queue.	
postscan	The rung-condition-out is set to false.	No action taken.

**Example 1:** When the temperature reaches the low limit (*temp\_low* is set), the AWT instruction sends a message to the MessageView terminal that is connected to the serial port of the controller. The message contains nine characters from the DATA member of the *string[2]* tag, which is a string. (The *\$14* counts as one character. It is the hex code for the Ctrl-T character.) The last character is a carriage return (\$r), which marks the end of the message.

**Relay Ladder**



### Structured Text

```

osri_1.InputBit := temp_low;

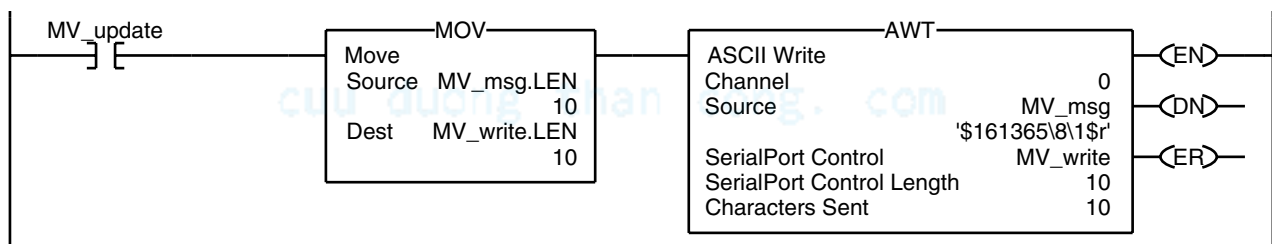
OSRI(osri_1);

IF (osri_1.OutputBit) THEN
    temp_low_write.LEN := 9;
    AWT(0,string[2],temp_low_write);
END_IF;

```

**Example 2:** When *MV\_update* is set, the AWT instruction sends the characters in *MV\_msg*. Because the number of characters in *MV\_msg* varies, the rung first moves the length of the string (*MV\_msg.LEN*) to the Serial Port Control Length of the AWT instruction (*MV\_write.LEN*). In *MV\_msg*, the *\$16* counts as one character. It is the hex code for the Ctrl-V character.

### Relay Ladder



### Structured Text

```

osri_1.InputBit := MV_update;

OSRI(osri_1);

IF (osri_1.OutputBit) THEN

    MV_write.LEN := Mv_msg.LEN;
    AWT(0,MV_msg,MV_write);

END_IF;

```

## Notes:

cuu duong than cong. com

cuu duong than cong. com

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

### Introduction

Use the ASCII string instructions to modify and create strings of ASCII characters.

If you want to	For example	Use this instruction	Available in these languages	See page
add characters to the end of a string	add termination characters or delimiters to a string	CONCAT	relay ladder structured text	608
delete characters from a string	remove header or control characters from a string	DELETE	relay ladder structured text	610
determine the starting character of a sub-string	locate a group of characters within a string	FIND	relay ladder structured text	612
insert characters into a string	create a string that uses variables	INSERT	relay ladder structured text	614
extract characters from a string	extract information from a bar code	MID	relay ladder structured text	616

You can also use the following instructions to compare or convert ASCII characters:

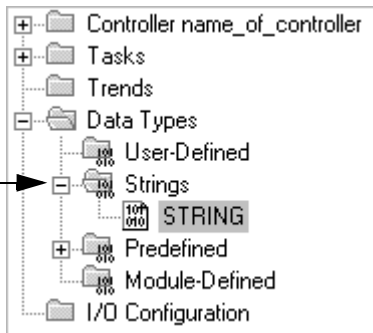
If you want to	Use this instruction	See page
compare a string to another string	CMP	207
see if the characters are equal to specific characters	EQU	212
see if the characters are not equal to specific characters	NEQ	243
see if the characters are equal to or greater than specific characters	GEQ	212
see if the characters are greater than specific characters	GRT	220
see if the characters are equal to or less than specific characters	LEQ	224
see if the characters are less than specific characters	LES	228
rearrange the bytes of a INT, DINT, or REAL tag	SWPB	301
find a string in an array of strings	FSC	349
convert characters to a SINT, INT, DINT, or REAL value	STOD	622

If you want to	Use this instruction	See page
convert characters to a REAL value	STOR	624
convert a SINT, INT, DINT, or REAL value to a string of ASCII characters	DTOS	626
convert REAL value to a string of ASCII characters	RTOS	629

cuu duong than cong. com

cuu duong than cong. com

## String Data Types



You store ASCII characters in tags that use a string data type.

- You can use the default STRING data type. It stores up to 82 characters.
- You can create a new string data type that stores less or more characters.

To create a new string data type, see *Logix5000 Controllers Common Procedures*, publication 1756-PM001.

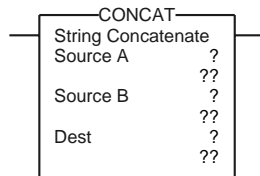
Each string data type contains the following members:

Name	Data Type	Description	Notes
LEN	DINT	number of characters in the string	<p>The LEN automatically updates to the new count of characters whenever you:</p> <ul style="list-style-type: none"> <li>• use the String Browser dialog box to enter characters</li> <li>• use instructions that read, convert, or manipulate a string</li> </ul> <p>The LEN shows the length of the current string. The DATA member may contain additional, old characters, which are not included in the LEN count.</p>
DATA	SINT array	ASCII characters of the string	<ul style="list-style-type: none"> <li>• To access the characters of the string, address the name of the tag. For example, to access the characters of the <i>string_1</i> tag, enter <i>string_1</i>.</li> <li>• Each element of the DATA array contains one character.</li> <li>• You can create new string data types that store less or more characters.</li> </ul>

## String Concatenate (CONCAT)

The CONCAT instruction adds ASCII characters to the end of a string.

### Operands:



### Relay Ladder

Operand	Type	Format	Enter	Notes
Source A	string	tag	tag that contains the initial characters	String data types are: <ul style="list-style-type: none"> <li>• default STRING data type</li> <li>• any new string data type that you create</li> </ul>
Source B	string	tag	tag that contains the end characters	
Destination	string	tag	tag to store the result	



```
CONCAT ( SourceA, SourceB,
        Dest );
```

### Structured Text

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

**Description:** The CONCAT instruction combines the characters in Source A with the characters in Source B and places the result in the Destination.

- The characters from Source A are first, followed by the characters from Source B.
- Unless Source A and the Destination are the same tag, Source A remains unchanged.

**Arithmetic Status Flags:** not affected

### Fault Conditions:

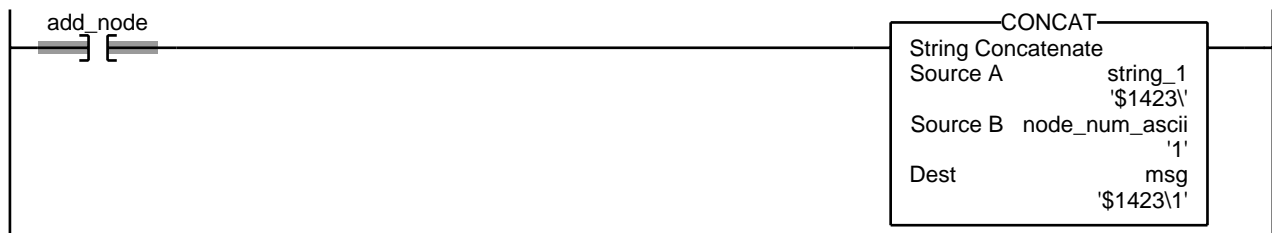
Type	Code	Cause	Recovery Method
4	51	The LEN value of the string tag is greater than the DATA size of the string tag.	1. Check that no instruction is writing to the LEN member of the string tag. 2. In the LEN value, enter the number of characters that the string contains.



**Execution:**

Condition	Relay Ladder Action	Structured Text Action
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes.  The rung-condition-out is set to true.	na
EnableIn is set	na	EnableIn is always set.  The instruction executes.
instruction execution	The instruction concatenates the strings.	
postscan	The rung-condition-out is set to false.	No action taken.

**Example:** To trigger a message in a MessageView terminal, the controller must send an ASCII string that contains a message number and node number. *String\_1* contains the message number. When *add\_node* is set, the CONCAT instruction adds the characters in *node\_num\_ascii* (node number) to the end of the characters in *string\_1* and then stores the result in *msg*.

**Relay Ladder****Structured Text**

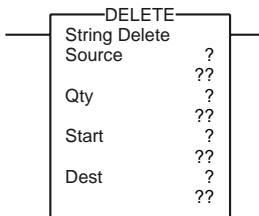
```

IF add_node THEN
    CONCAT(string_1,node_num_ascii,msg);
    add_node := 0;
END_IF;
  
```

# String Delete (DELETE)

The DELETE instruction removes ASCII characters from a string.

## Operands:



## Relay Ladder

Operand	Type	Format	Enter	Notes
Source	string	tag	tag that contains the string from which you want to delete characters	String data types are: <ul style="list-style-type: none"> <li>• default STRING data type</li> <li>• any new string data type that you create</li> </ul>
Quantity	SINT	immediate	number of characters to delete	The Start plus the Quantity must be less than or equal to the DATA size of the Source.
	INT	tag		
	<b>DINT</b>			
Start	SINT	immediate	position of the first character to delete	Enter a number between 1 and the DATA size of the Source.
	INT	tag		
	<b>DINT</b>			
Destination	string	tag	tag to store the result	



DELETE (Source,Qty,Start, Dest);

## Structured Text

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

- Description:** The DELETE instruction deletes (removes) a group of characters from the Source and places the remaining characters in the Destination.
- The Start position and Quantity define the characters to remove.
  - Unless the Source and Destination are the same tag, the Source remains unchanged.

**Arithmetic Status Flags:** not affected

**Fault Conditions:**

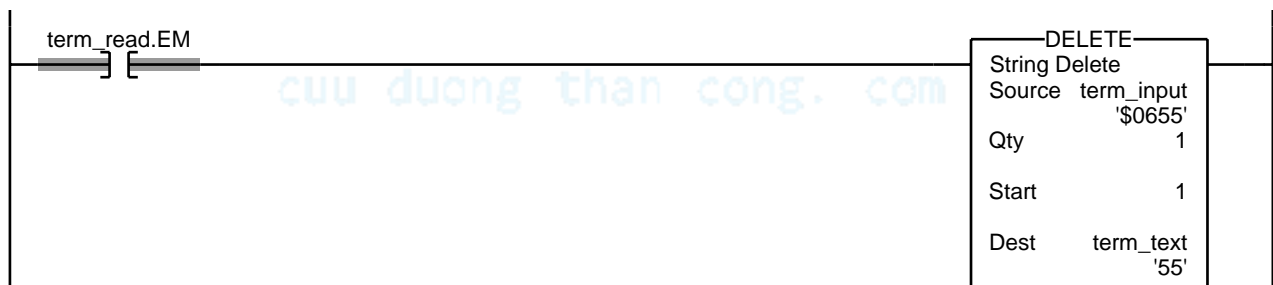
Type	Code	Cause	Recovery Method
4	51	The LEN value of the string tag is greater than the DATA size of the string tag.	1. Check that no instruction is writing to the LEN member of the string tag. 2. In the LEN value, enter the number of characters that the string contains.
4	56	The Start or Quantity value is invalid.	1. Check that the Start value is between 1 and the DATA size of the Source. 2. Check that the Start value plus the Quantity value is less than or equal to the DATA size of the Source.

**Execution:**

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

**Example:** ASCII information from a terminal contains a header character. After the controller reads the data (*term\_read.EM* is set) the DELETE instruction removes the header character.

**Relay Ladder**



Structured Text

```
IF term_read.EM THEN

    DELETE(term_input,1,1,term_text);

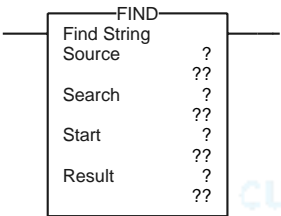
    term_read.EM := 0;

END_IF;
```

Find String (FIND)

The FIND instruction locates the starting position of a specified string within another string

Operands:



Relay Ladder

Operand	Type	Format	Enter	Notes
Source	string	tag	string to search in	String data types are: <ul style="list-style-type: none"><li>• default STRING data type</li><li>• any new string data type that you create</li></ul>
Search	string	tag	string to find	
Start	SINT	immediate	position in Source to start the search	Enter a number between 1 and the DATA size of the Source.
	INT	tag		
	DINT			
Result	SINT	tag	tag that stores the starting position of the string to find	
	INT			
	DINT			



Structured Text

```
FIND(Source,Search,Start,Result);
```

The operands are the same as those for the relay ladder FIND instruction described above.

**Description:** The FIND instruction searches the Source string for the Search string. If the instruction finds the Search string, the Result shows the starting position of the Search string within the Source string.

**Arithmetic Status Flags:** not affected

**Fault Conditions:**

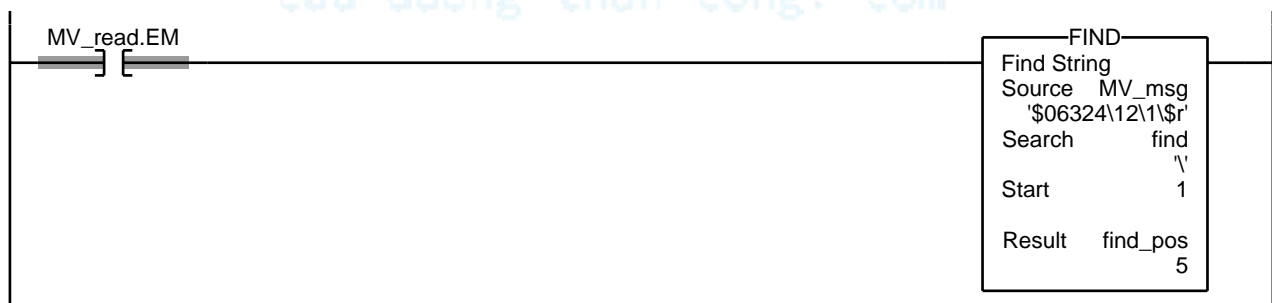
Type	Code	Cause	Recovery Method
4	51	The LEN value of the string tag is greater than the DATA size of the string tag.	1. Check that no instruction is writing to the LEN member of the string tag. 2. In the LEN value, enter the number of characters that the string contains.
4	56	The Start value is invalid.	Check that the Start value is between 1 and the DATA size of the Source.

**Execution:**

Condition	Relay Ladder Action	Structured Text Action
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	na
EnableIn is set	na	EnableIn is always set. The instruction executes.
instruction execution	The instruction searches for the specified characters.	
postscan	The rung-condition-out is set to false.	No action taken.

**Example:** A message from a MessageView terminal contains several pieces of information. The backslash character [ \ ] separates each piece of information. To locate a piece of information, the FIND instruction searches for the backslash character and records its position in *find\_pos*.

**Relay Ladder**



## Structured Text

```
IF MV_read.EM THEN

    FIND(MV_msg,find,1,find_pos);

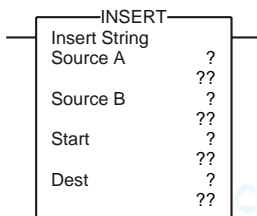
    MV_read.EM := 0;

END_IF;
```

## Insert String (INSERT)

The INSERT instruction adds ASCII characters to a specified location within a string.

### Operands:



### Relay Ladder

Operand	Type	Format	Enter	Notes
Source A	string	tag	string to add the characters to	String data types are: <ul style="list-style-type: none"> <li>• default STRING data type</li> <li>• any new string data type that you create</li> </ul>
Source B	string	tag	string containing the characters to add	
Start	SINT INT  DINT	immediate tag	position in Source A to add the characters	
Result	string	tag	string to store the result	Enter a number between 1 and the DATA size of the Source.



```
INSERT(SourceA,SourceB,
      Start, Dest);
```

### Structured Text

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

**Description:** The INSERT instruction adds the characters in Source B to a designated position within Source A and places the result in the Destination:

- Start defines where in Source A that Source B is added.
- Unless SourceA and the Destination are the same tag, Source A remains unchanged.

**Arithmetic Status Flags:** not affected

### Fault Conditions:

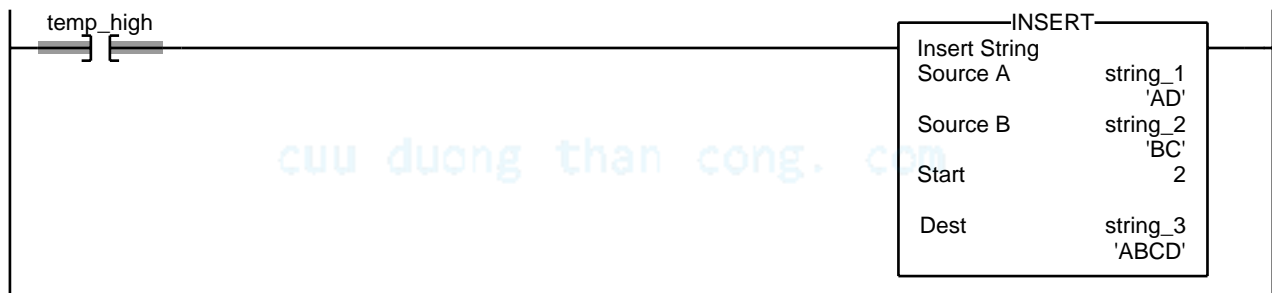
Type	Code	Cause	Recovery Method
4	51	The LEN value of the string tag is greater than the DATA size of the string tag.	1. Check that no instruction is writing to the LEN member of the string tag. 2. In the LEN value, enter the number of characters that the string contains.
4	56	The Start value is invalid.	Check that the Start value is between 1 and the DATA size of the Source.

### Execution:

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

**Example:** When *temp\_high* is set, the INSERT instruction adds the characters in *string\_2* to position 2 within *string\_1* and places the result in *string\_3*:

### Relay Ladder



## Structured Text

```
IF temp_high THEN

    INSERT(string_1,string_2,2,string_3);

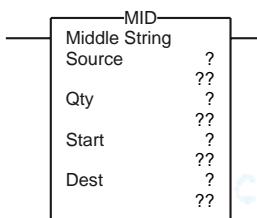
    temp_high := 0;

END_IF;
```

## Middle String (MID)

The MID instruction copies a specified number of ASCII characters from a string and stores them in another string.

### Operands:



### Relay Ladder

Operand	Type	Format	Enter	Notes
Source	string	tag	string to copy characters from	String data types are: <ul style="list-style-type: none"> <li>• default STRING data type</li> <li>• any new string data type that you create</li> </ul>
Quantity	SINT INT <b>DINT</b>	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 <b>DINT</b>	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	



```
MID(Source,Qty,Start,
    Dest);
```

## Structured Text

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



**Description:** The MID instruction copies a group of characters from the Source and places the result in the Destination.

- The Start position and Quantity define the characters to copy.
- Unless the Source and Destination are the same tag, the Source remains unchanged.

**Arithmetic Status Flags:** not affected

**Fault Conditions:**

Type	Code	Cause	Recovery Method
4	51	The LEN value of the string tag is greater than the DATA size of the string tag.	1. Check that no instruction is writing to the LEN member of the string tag. 2. In the LEN value, enter the number of characters that the string contains.
4	56	The Start or Quantity value is invalid.	1. Check that the Start value is between 1 and the DATA size of the Source. 2. Check that the Start value plus the Quantity value is less than or equal to the DATA size of the Source.

**Execution:**

Condition	Relay Ladder Action	Structured Text Action
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	na
EnableIn is set	na	EnableIn is always set. The instruction executes.
instruction execution	The instruction copies the specified characters from a string and stores them in another string.	
postscan	The rung-condition-out is set to false.	No action taken.

**Example:** In a baggage handling conveyor of an airport, each bag gets a bar code. Characters 9 - 17 of the bar code are the flight number and destination airport of the bag. After the bar code is read (*bag\_read.EM* is set) the MID instruction copies the flight number and destination airport to the *bag\_flt\_and\_dest* string.

**Relay Ladder**



**Structured Text**

```
IF bag_read.EM THEN
    MID(bar_barcode,9,9,bag_flt_and_dest);
    bag_read.EM := 0;
END_IF;
```

cuu duong than cong. com

cuu duong than cong. com

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

### Introduction

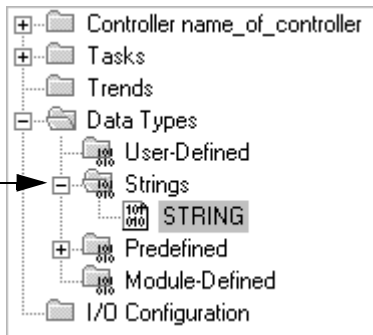
Use the ASCII conversion instructions to alter the format of data.

If You Want To	For Example	Use This Instruction	Available In These Languages	See Page
convert the ASCII representation of an integer value to a SINT, INT, DINT, or REAL value	convert a value from a weight scale or other ASCII device to an integer so you can use it in your logic	STOD	relay ladder structured text	622
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	624
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	626
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	See Page
add characters to the end of a string	CONCAT	608
delete characters from a string	DELETE	610
determine the starting character of a sub-string	FIND	612
insert characters into a string	INSERT	614
extract characters from a string	MID	616
rearrange the bytes of a INT, DINT, or REAL tag	SWPB	301
compare a string to another string	CMP	207
see if the characters are equal to specific characters	EQU	212
see if the characters are not equal to specific characters	NEQ	243
see if the characters are equal to or greater than specific characters	GEQ	216
see if the characters are greater than specific characters	GRT	220
see if the characters are equal to or less than specific characters	LEQ	224
see if the characters are less than specific characters	LES	228
find a string in an array of strings	FSC	349

## String Data Types



You store ASCII characters in tags that use a string data type.

- You can use the default STRING data type. It stores up to 82 characters.
- You can create a new string data type that stores less or more characters.

To create a new string data type, see *Logix5000 Controllers Common Procedures*, publication 1756-PM001.

Each string data type contains the following members:

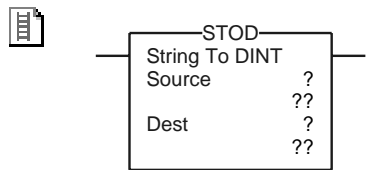
Name:	Data Type:	Description:	Notes:
LEN	DINT	number of characters in the string	<p>The LEN automatically updates to the new count of characters whenever you:</p> <ul style="list-style-type: none"> <li>• use the String Browser dialog box to enter characters</li> <li>• use instructions that read, convert, or manipulate a string</li> </ul> <p>The LEN shows the length of the current string. The DATA member may contain additional, old characters, which are not included in the LEN count.</p>
DATA	SINT array	ASCII characters of the string	<ul style="list-style-type: none"> <li>• To access the characters of the string, address the name of the tag. For example, to access the characters of the <i>string_1</i> tag, enter <i>string_1</i>.</li> <li>• Each element of the DATA array contains one character.</li> <li>• You can create new string data types that store less or more characters.</li> </ul>

# String To DINT (STOD)

The STOD instruction converts the ASCII representation of an integer to an integer or REAL value.

## Operands:

### Relay Ladder



Operand	Type	Format	Enter	Notes
Source	string	tag	tag that contains the value in ASCII	String data types are: <ul style="list-style-type: none"><li>• default STRING data type</li><li>• any new string data type that you create</li></ul>
Destination	SINT INT <b>DINT</b> 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

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.	<ol style="list-style-type: none"> <li>1. Check that no instruction is writing to the LEN member of the string tag.</li> <li>2. In the LEN value, enter the number of characters that the string contains.</li> </ol>
4	53	The output number is beyond the limits of the destination data type.	Either: <ul style="list-style-type: none"> <li>• Reduce the size of the ASCII value.</li> <li>• Use a larger data type for the destination.</li> </ul>

### Execution:

Condition	Relay Ladder Action	Structured Text Action
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes.  The rung-condition-out is set to true.	na
EnableIn is set	na	EnableIn is always set. The instruction executes.
instruction execution	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 ) ;

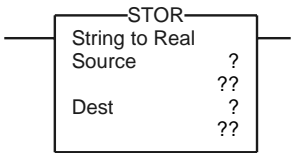
    MV_read.EM := 0 ;

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: <ul style="list-style-type: none"><li>• default STRING data type</li><li>• any new string data type that you create</li></ul>
Destination	REAL	tag	tag to store the REAL value	



STOR ( Source , Dest ) ;

### Structured Text

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

**Description:** The STOR converts the Source to a REAL value and places the result in the Destination.

- The instruction converts positive and negative numbers.
- If the Source string contains non-numeric characters, the STOR converts the first set of contiguous numbers, including the decimal point [ . ]:
  - The instruction skips any initial control or non-numeric characters (except the minus sign in front of a number).
  - If the string contains multiple groups of numbers that are separated by delimiters (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.	<ol style="list-style-type: none"> <li>1. Check that no instruction is writing to the LEN member of the string tag.</li> <li>2. In the LEN value, enter the number of characters that the string contains.</li> </ol>
4	53	The output number is beyond the limits of the destination data type.	Either: <ul style="list-style-type: none"> <li>• Reduce the size of the ASCII value.</li> <li>• Use a larger data type for the destination.</li> </ul>

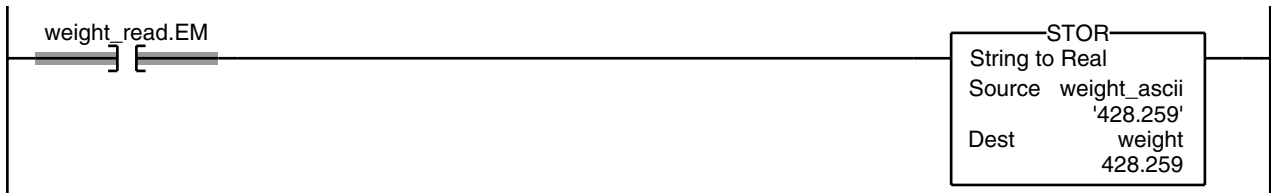
**Execution:**

Condition	Relay Ladder Action	Structured Text Action
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes. The rung-condition-out is set to true.	na
EnableIn is ste	na	EnableIn is always set. The instruction executes.
instruction execution	S:C is set. Destination is cleared. The instruction converts the Source. If the result is zero, then S:Z is set	
postscan	The rung-condition-out is set to false.	No action taken.

**Example:** After reading the weight from a scale (*weight\_read.EM* is set) the STOR instruction converts the numeric characters in *weight\_ascii* to a REAL value.

You *may* see a slight difference between the fractional parts of the Source and Destination.

Relay Ladder



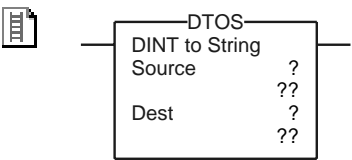
Structured Text

```
IF weight_read.EM THEN
    STOR(weight_ascii,weight);
    weight_read.EM := 0;
END_IF;
```

DINT to String (DTOS) The DTOS instruction produces the ASCII representation of a value.

Operands:

Relay Ladder



Operand	Type	Format	Enter	Notes
Source	SINT	tag	tag that contains the value	If the Source is a REAL, the instruction converts it to a DINT value. Refer to REAL to an integer on page 640.
	INT			
	DINT			
	REAL			
Destination	string	tag	tag to store the ASCII value	String data types are: <ul style="list-style-type: none"><li>• default STRING data type</li><li>• any new string data type that you create</li></ul>



DTOS (Source, Dest) ;

### Structured Text

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

**Description:** The DTOS converts the Source to a string of ASCII characters and places the result in the Destination.

**Arithmetic Status Flags:** not affected

### Fault Conditions:

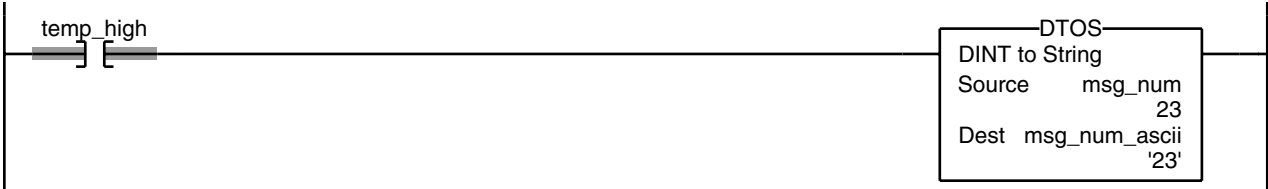
Type	Code	Cause	Recovery Method
4	51	The LEN value of the string tag is greater than the DATA size of the string tag.	1. Check that no instruction is writing to the LEN member of the string tag.  2. In the LEN value, enter the number of characters that the string contains.
4	52	The output string is larger than the destination.	Create a new string data type that is large enough for the output string. Use the new string data type as the data type for the destination.

### Execution:

Condition	Relay Ladder Action	Structured Text Action
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes.  The rung-condition-out is set to true.	na
EnableIn is set	na	EnableIn is always set.  The instruction executes.
instruction execution	The instruction converts the Source.	
postscan	The rung-condition-out is set to false.	No action taken.

**Example:** When *temp\_high* is set, the DTOS instruction converts the value in *msg\_num* to a string of ASCII characters and places the result in *msg\_num\_ascii*. Subsequent rungs insert or concatenate *msg\_num\_ascii* with other strings to produce a complete message for a display terminal.

Relay Ladder



Structured Text

```
IF temp_high THEN
    DTOS(msg_num,msg_num_ascii);
    temp_high := 0;
END_IF;
```

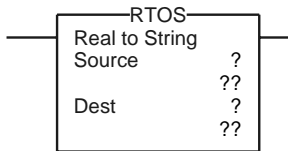
cuu duong than cong. com

cuu duong than cong. com

## REAL to String (RTOS)

The RTOS instruction produces the ASCII representation of a REAL value.

### Operands:



### Relay Ladder

Operand	Type	Format	Enter	Notes
Source	REAL	tag	tag that contains the REAL value	
Destination	string	tag	tag to store the ASCII value	String data types are: <ul style="list-style-type: none"> <li>• default STRING data type</li> <li>• any new string data type that you create</li> </ul>



RTOS (Source, Dest) ;

### Structured Text

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

**Description:** The RTOS converts the Source to a string of ASCII characters and places the result in the Destination.

**Arithmetic Status Flags:** not affected

### Fault Conditions:

Type	Code	Cause	Recovery Method
4	51	The LEN value of the string tag is greater than the DATA size of the string tag.	1. Check that no instruction is writing to the LEN member of the string tag.  2. In the LEN value, enter the number of characters that the string contains.
4	52	The output string is larger than the destination.	Create a new string data type that is large enough for the output string. Use the new string data type as the data type for the destination.

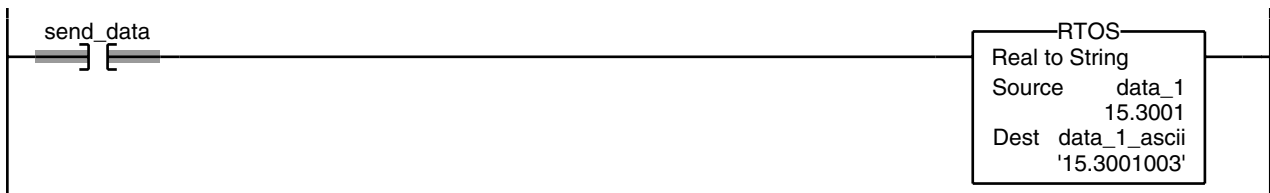
### Execution:

Condition	Relay Ladder Action	Structured Text Action
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes.  The rung-condition-out is set to true.	na
EnableIn is set	na	EnableIn is always set.  The instruction executes.
instruction execution	The instruction converts the Source.	
postscan	The rung-condition-out is set to false.	No action taken.

**Example:** When *send\_data* is set, the RTOS instruction converts the value in *data\_1* to a string of ASCII characters and places the result in *data\_1\_ascii*. Subsequent rungs insert or concatenate *data\_1\_ascii* with other strings to produce a complete message for a display terminal.

You *may* see a slight difference between the fractional parts of the Source and Destination.

### Relay Ladder



### Structured Text

```
IF send_data THEN
```

```
    RTOS(data_1,data_1_ascii);
```

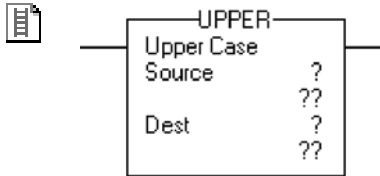
```
    send_data := 0;
```

```
END_IF;
```

## Upper Case (UPPER)

The UPPER instruction converts the alphabetical characters in a string to upper case characters.

### Operands:



### Relay Ladder

Operand	Type	Format	Description
Source	string	tag	tag that contains the characters that you want to convert to upper case
Destination	string	tag	tag to store the characters in upper case



```
UPPER ( Source , Dest ) ;
```

### Structured Text

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

**Description:** The UPPER instruction converts to upper case all the letters in the Source and places the result in the Destination.

- ASCII characters are case sensitive. Upper case “A” (\$41) is *not* equal to lower case “a” (\$61).
- If operators directly enter ASCII characters, convert the characters to all upper case or all lower case before you compare them.

Any characters in the Source string that are not letters remain unchanged.

**Arithmetic Status Flags:** not affected

**Fault Conditions:** none

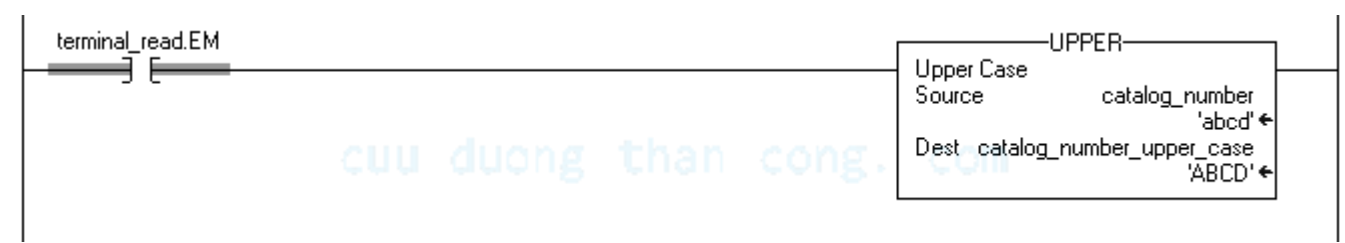
### Execution:

Condition	Relay Ladder Action	Structured Text Action
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes.	na
	The rung-condition-out is set to true.	

Condition	Relay Ladder Action	Structured Text Action
EnableIn is set	na	EnableIn is always set.  The instruction executes.
instruction execution	The instruction converts the Source to upper case.	
postscan	The rung-condition-out is set to false.	No action taken.

**Example:** To find information about a specific item, an operator enters the catalog number of the item into an ASCII terminal. After the controller reads the input from a terminal (*terminal\_read.EM* is set), the UPPER instruction converts the characters in *catalog\_number* to all upper case characters and stores the result in *catalog\_number\_upper\_case*. A subsequent rung then searches an array for characters that match those in *catalog\_number\_upper\_case*.

Relay Ladder



Structured Text

```
IF terminal_read.EM THEN

    UPPER(catalog_number,catalog_number_upper_case);

    terminal_read.EM := 0;

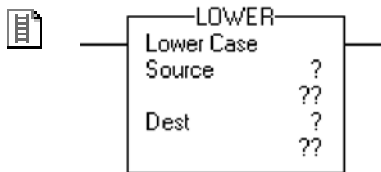
END_IF;
```



## Lower Case (LOWER)

The LOWER instruction converts the alphabetical characters in a string to lower case characters.

### Operands:



### Relay Ladder

Operand	Type	Format	Description
Source	string	tag	tag that contains the characters that you want to convert to lower case
Destination	string	tag	tag to store the characters in lower case

 LOWER (Source, Dest) ;

### Structured Text

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

**Description:** The LOWER instruction converts to lower case all the letters in the Source and places the result in the Destination.

- ASCII characters are case sensitive. Upper case “A” (\$41) is *not* equal to lower case “a” (\$61).
- If operators directly enter ASCII characters, convert the characters to all upper case or all lower case before you compare them.

Any characters in the Source string that are not letters remain unchanged.

**Arithmetic Status Flags:** not affected

**Fault Conditions:** none

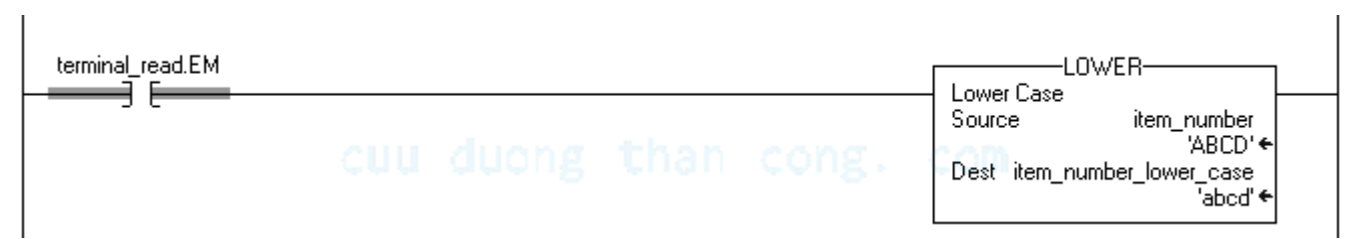
### Execution:

Condition	Relay Ladder Action	Structured Text Action
prescan	The rung-condition-out is set to false.	No action taken.
rung-condition-in is false	The rung-condition-out is set to false.	na
rung-condition-in is true	The instruction executes.	na
	The rung-condition-out is set to true.	

Condition	Relay Ladder Action	Structured Text Action
EnableIn is set	na	EnableIn is always set.  The instruction executes.
instruction execution	The instruction converts the Source to lower case.	
postscan	The rung-condition-out is set to false.	No action taken.

**Example:** To find information about a specific item, an operator enters the item number into an ASCII terminal. After the controller reads the input from a terminal (*terminal\_read.EM* is set), the LOWER instruction converts the characters in *item\_number* to all lower case characters and stores the result in *item\_number\_lower\_case*. A subsequent rung then searches an array for characters that match those in *item\_number\_lower\_case*.

Relay Ladder



Structured Text

```
IF terminal_read.EM THEN  
  
    LOWER(item_number,item_number_lower_case);  
  
    terminal_read.EM := 0;  
  
END_IF;
```

## Common Attributes

### Introduction

This appendix describes attributes that are common to the Logix instructions.

For Information About	See Page
Immediate Values	635
Data Conversions	635

### Immediate Values

Whenever you enter an immediate value (constant) in decimal format (for example, -2, 3) the controller stores the value using 32 bits. If you enter a value in a radix other than decimal, such as binary or hexadecimal, and do not specify all 32 bits, the controller places a zero in the bits that you do not specify (zero-fill).

**EXAMPLE**

Zero-filling of immediate values

If You Enter	The Controller Stores
-1	16#ffff ffff (-1)
16#ffff (-1)	16#0000 ffff (65535)
8#1234 (668)	16#0000 029c (668)
2#1010 (10)	16#0000 000a (10)

### Data Conversions

Data conversions occur when you mix data types in your programming:

When Programming in	Conversions Can Occur When 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 only support one data type (the optimal data type) for its operands.

If you mix data types and use tags that are not the optimal data type, the controller converts the data according to these rules

- Are *any* of the operands a REAL value?

If	Then input operands (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:

- using the same data type throughout the instruction
- minimizing the use of the SINT or INT data types

In other words, use all DINT tags or all REAL tags, along with immediate values, in your instructions.

The following sections explain how the data is converted when you use SINT or INT tags or when you mix data types.

## SINT or INT to DINT

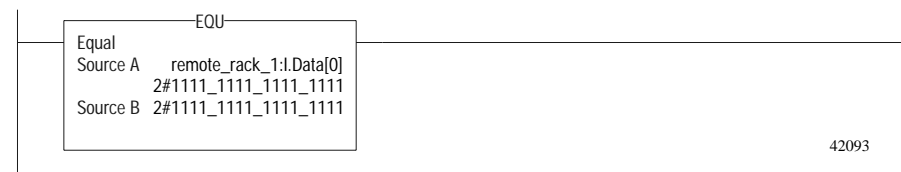
For those instructions that convert SINT or INT values to DINT values, the “Operands” sections in this manual identify the conversion method.

This Conversion Method	Converts Data By Placing
Sign-extension	the value of the left-most bit (the sign of the value) into each bit position to the left of the existing bits until there are 32 bits.
Zero-fill	zeroes to the left of the existing bits until there are 32 bits

The following example shows the results of converting a value using sign-extension and zero-fill.

This value	2#1111_1111_1111_1111	(-1)
Converts to this value by sign-extension	2#1111_1111_1111_1111_1111_1111_1111_1111	(-1)
Converts to this value by zero-fill	2#0000_0000_0000_0000_1111_1111_1111_1111	(65535)

Because immediate values are always zero-filled, the conversion of a SINT or INT value *may* produce unexpected results. In the following example, the comparison is false because Source A, an INT, converts by sign-extension; while Source B, an immediate value, is zero-filled.



If you use a SINT or INT tag and an immediate value in an instruction that converts data by sign-extension, use one of these methods to handle immediate values:

- Specify any immediate value in the decimal radix
- If you are entering the value in a radix other than decimal, specify all 32 bits of the immediate value. To do so, enter the value of the left-most bit into each bit position to its left until there are 32 bits.
- Create a tag for each operand and use the same data type throughout the instruction. To assign a constant value, either:
  - Enter it into one of the tags
  - Add a MOV instruction that moves the value into one of the tags.
- Use a MEQ instruction to check only the required bits

The following examples show two ways to mix an immediate value with an INT tag. Both examples check the bits of a 1771 I/O module to determine if all the bits are on. Since the input data word of a 1771 I/O module is an INT tag, it is easiest to use a 16-bit constant value.

EXAMPLE

Mixing an INT tag with an immediate value

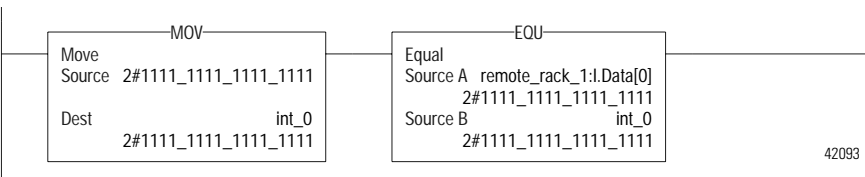
Since *remote\_rack\_1:I.Data[0]* is an INT tag, the value to check it against is also entered as an INT tag.



EXAMPLE

Mixing an INT tag with an immediate value

Since *remote\_rack\_1:I.Data[0]* is an INT tag, the value to check it against first moves into *int\_0*, also an INT tag. The EQU instruction then compares both tags.



## Integer to REAL

The controller stores REAL values in IEEE single-precision, floating-point number format. It uses one bit for the sign of the value, 23 bits for the base value, and eight bits for the exponent (32 bits total). If you mix an integer tag (SINT, INT, or DINT) and a REAL tag as inputs in the same instruction, the controller converts the integer value to a REAL value before the instruction executes.

- A SINT or INT value always converts to the same REAL value.
- A DINT value may not convert to the same REAL value:
  - A REAL value uses up to 24 bits for the base value (23 stored bits plus a “hidden” bit).
  - A DINT value uses up to 32 bits for the value (one for the sign and 31 for the value).
  - If the DINT value requires more than 24 significant bits, it *may not* convert to the same REAL value. If it will not, the controller rounds to the nearest REAL value using 24 significant bits.

cuu duong than cong. com

## DINT to SINT or INT

To convert a DINT value to a SINT or INT value, the controller truncates the upper portion of the DINT and sets the overflow status flag, if necessary. The following example shows the result of a DINT to SINT or INT conversion.

### EXAMPLE

Conversion of a DINT to an INT and a SINT

This DINT Value	Converts To This Smaller Value
16#0001_0081 (65,665)	INT: 16#0081 (129)
	SINT: 16#81 (-127)

## REAL to an integer

To convert a REAL value to an integer value, the controller rounds the fractional part and truncates the upper portion of the non-fractional part. If data is lost, the controller sets the overflow status flag. Numbers round as follows:

- Numbers other than  $x.5$  round to the nearest whole number.
- $X.5$  rounds to the nearest even number.

The following example show the result of converting REAL values to DINT values.

### EXAMPLE

Conversion of REAL values to DINT values

This REAL Value	Converts To This DINT Value
-2.5	-2
-1.6	-2
-1.5	-2
-1.4	-1
1.4	1
1.5	2
1.6	2
2.5	2

### IMPORTANT

The arithmetic status flags are set based on the value being stored. Instructions that normally do not affect arithmetic status keywords might appear to do so if type conversion occurs because of mixed data types for the instruction parameters. The type conversion process sets the arithmetic status keywords.



## Function Block Attributes

### Introduction

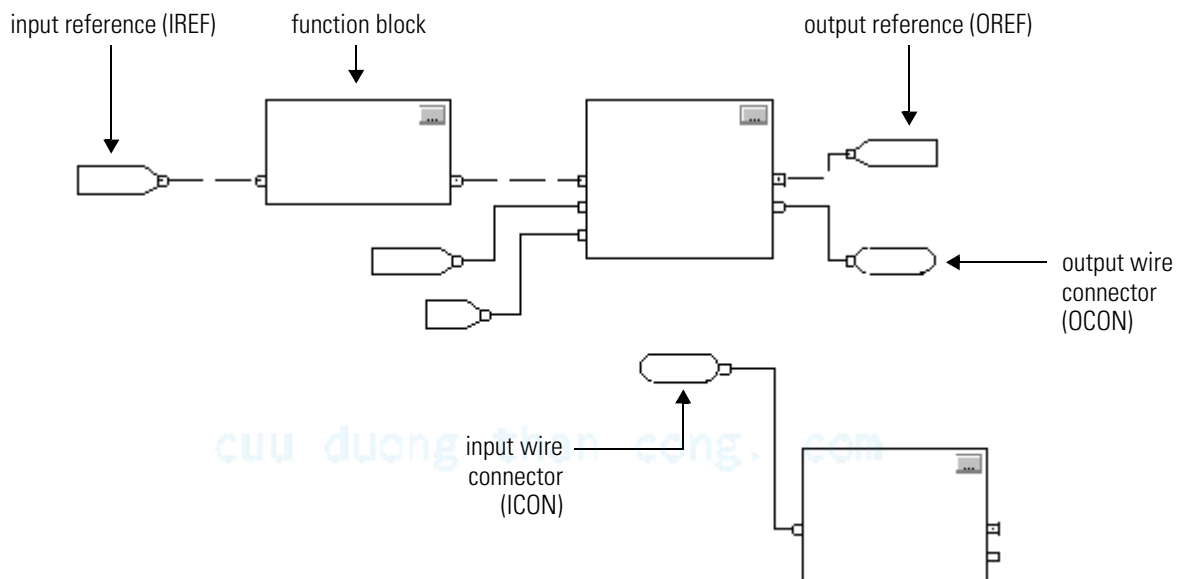
This appendix describes issues that are unique with function block instructions. Review the information in this appendix to make sure you understand how your function block routines will operate.

**IMPORTANT**

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

### Choose the Function Block Elements

To control a device, use the following elements:

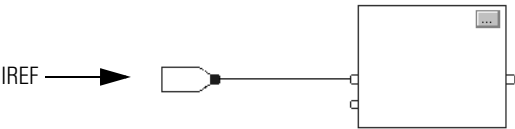


Use the following table to choose your function block elements:

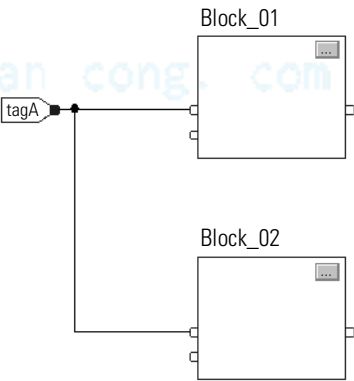
If You Want To	Use a
supply a value from an input device or tag	input reference (IREF)
send a value to an output device or tag	output reference (OREF)
perform an operation on an input value or values and produce an output value or values	function block
transfer data between function blocks when they are: <ul style="list-style-type: none"><li>• far apart on the same sheet</li><li>• on different sheets within the same routine</li></ul>	output wire connector (OCON) and an input wire connector (ICON)
disperse data to several points in the routine	single output wire connector (OCON) and multiple input wire connectors (ICON)

Latching Data

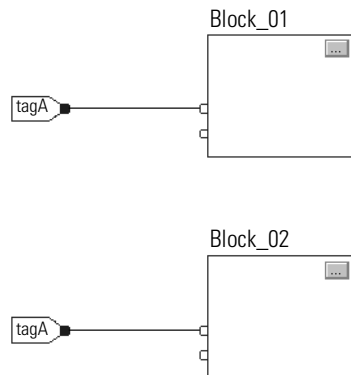
If you use an IREF to specify input data for a function block instruction, the data in that IREF is latched for the scan of the function block routine. The IREF latches data from program-scoped and controller-scoped tags. The controller updates all IREF data at the beginning of each scan.



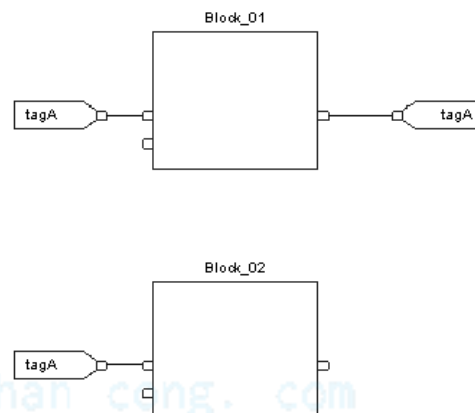
In this example, the value of tagA is stored at the beginning of the routine's execution. The stored value is used when Block\_01 executes. The same stored value is also used when Block\_02 executes. If the value of tagA changes during execution of the routine, the stored value of tagA in the IREF does not change until the next execution of the routine.



This example is the same as the one above. The value of tagA is stored only once at the beginning of the routine's execution. The routine uses this stored value throughout the routine.



Starting with RSLogix 5000 software, version 11, you can use the same tag in multiple IREFs and an OREF in the same routine. Because the values of tags in IREFs are latched every scan through the routine, all IREFs will use the same value, even if an OREF obtains a different tag value during execution of the routine. In this example, if tagA has a value of 25.4 when the routine starts executing this scan, and Block\_01 changes the value of tagA to 50.9, the second IREF wired into Block\_02 will still use a value of 25.4 when Block\_02 executes this scan. The new tagA value of 50.9 will not be used by any IREFs in this routine until the start of the next scan.



## Order of Execution

The RSLogix 5000 programming software automatically determines the order of execution for the function blocks in a routine when you:

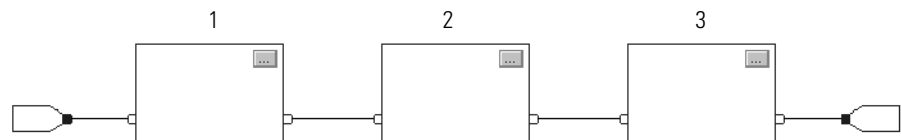
- verify a function block routine
- verify a project that contains a function block routine
- download a project that contains a function block routine

You define execution order by wiring function blocks together and indicating the data flow of any feedback wires, if necessary.

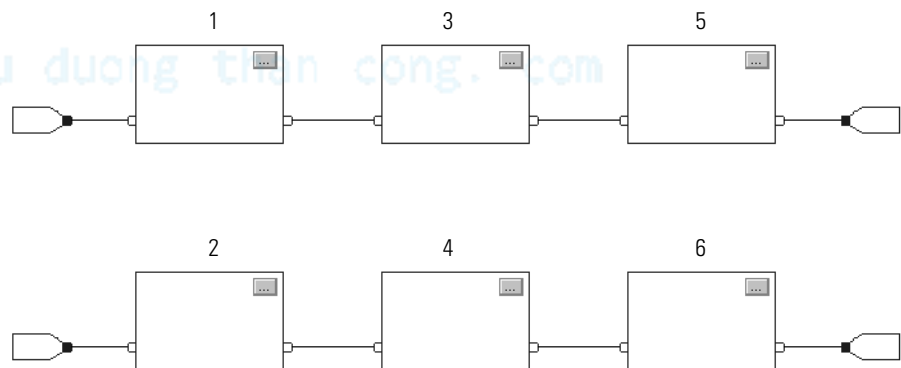
If function blocks are not wired together, it does not matter which block executes first. There is no data flow between the blocks.



If you wire the blocks sequentially, the execution order moves from input to output. The inputs of a block require data to be available before the controller can execute that block. For example, block 2 has to execute before block 3 because the outputs of block 2 feed the inputs of block 3.

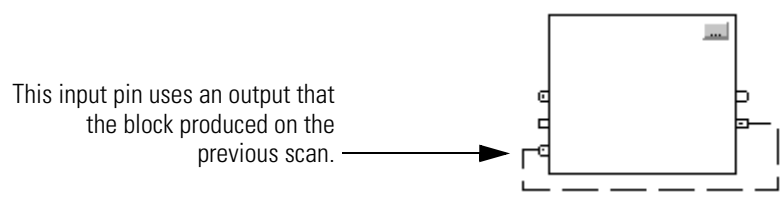


Execution order is only relative to the blocks that are wired together. The following example is fine because the two groups of blocks are not wired together. The blocks within a specific group execute in the appropriate order in relation to the blocks in that group.

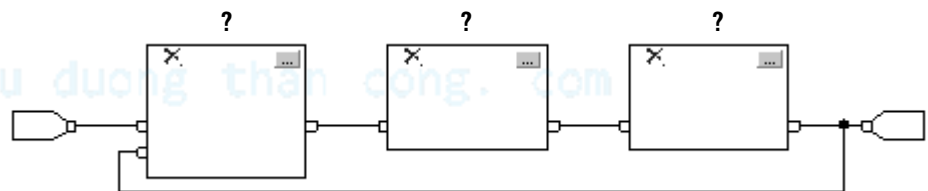


## Resolve a Loop

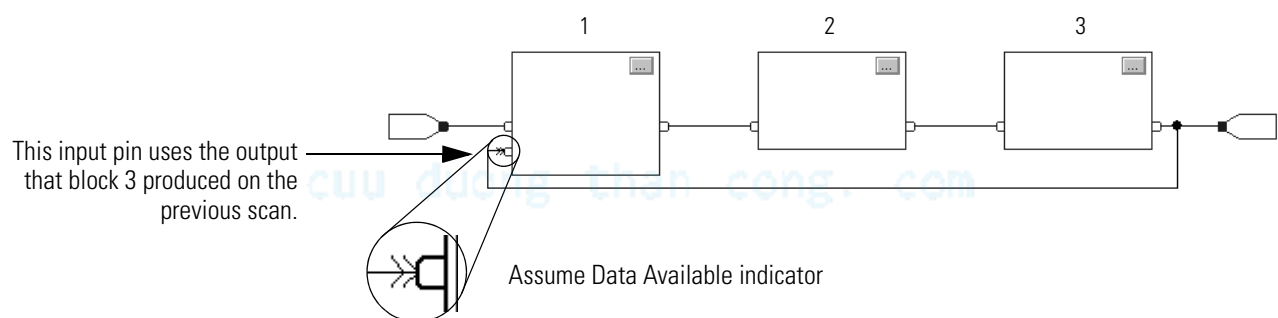
To create a feedback loop around a block, wire an output pin of the block to an input pin of the same block. The following example is OK. The loop contains only a single block, so execution order does not matter.



If a group of blocks are in a loop, the controller cannot determine which block to execute first. In other words, it cannot resolve the loop.

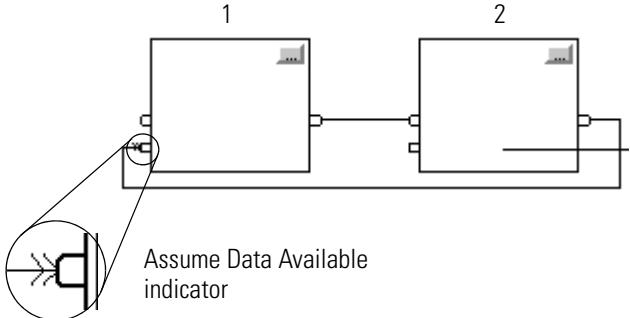
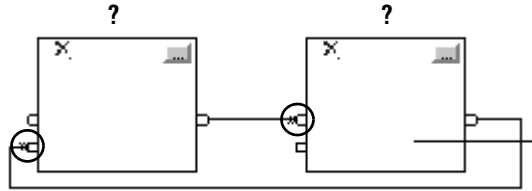


To identify which block to execute first, mark the input wire that creates the loop (the feedback wire) with the *Assume Data Available* indicator. In the following example, block 1 uses the output from block 3 that was produced in the previous execution of the routine.



The *Assume Data Available* indicator defines the data flow within the loop. The arrow indicates that the data serves as input to the first block in the loop.

Do not mark all the wires of a loop with the *Assume Data Available* indicator.

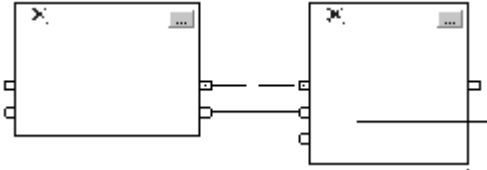
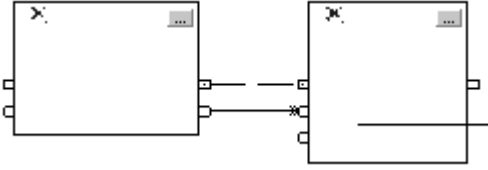
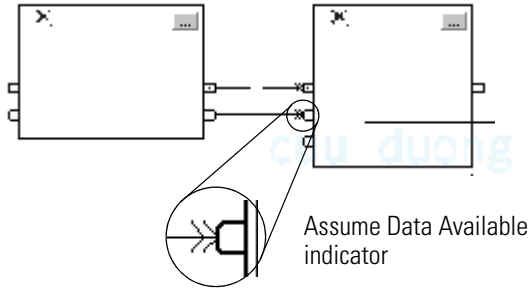
This is OK	This is NOT OK
<div><div><div><div>1</div><div>2</div></div><p>Assume Data Available indicator</p><p>The <i>Assume Data Available</i> indicator defines the data flow within the loop.</p></div></div>	<div><div><div><div>?</div><div>?</div></div><p>The controller cannot resolve the loop because all the wires use the <i>Assume Data Available</i> indicator.</p></div></div>

cuu duong than cong. com

cuu duong than cong. com

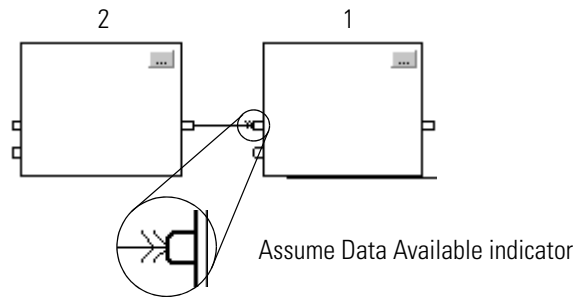
### Resolve Data Flow Between Two Blocks

If you use two or more wires to connect two blocks, use the same data flow indicators for all of the wires between the two blocks.

This is OK	This is NOT OK
	
Neither wire uses the <i>Assume Data Available</i> indicator.	One wire uses the <i>Assume Data Available</i> indicator while the other wire does not.
	
Both wires use the <i>Assume Data Available</i> 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

In summary, a function block routine executes in this order:

1. The controller latches all data values in IREFs.
2. The controller executes the other function blocks in the order determined by how they are wired.
3. The controller writes outputs in OREFs.



## Function Block Responses to Overflow Conditions

In general, the function block instructions that maintain history do not update history with  $\pm\text{NAN}$ , or  $\pm\text{INF}$  values when an overflow occurs. Each instruction has one of these responses to an overflow condition:

Response 1:		Response 2:	Response 3:	
Blocks execute their algorithm and check the result for $\pm\text{NAN}$ or $\pm\text{INF}$ . If $\pm\text{NAN}$ or $\pm\text{INF}$ , the block outputs $\pm\text{NAN}$ or $\pm\text{INF}$ .		Blocks with output limiting execute their algorithm and check the result for $\pm\text{NAN}$ or $\pm\text{INF}$ . The output limits are defined by the HighLimit and LowLimit input parameters. If $\pm\text{INF}$ , the block outputs a limited result. If $\pm\text{NAN}$ , the output limits are not used and the block outputs $\pm\text{NAN}$ .	The overflow condition does not apply. These instructions typically have a boolean output.	
ALM	NTCH	HLL	BAND	OSRI
DEDT	PMUL	INTG	BNOT	RESD
DERV	POSP	PI	BOR	RTOR
ESEL	RLIM	PIDE	BXOR	SETD
FGEN	RMPS	SCL	CUTD	TOFR
HPF	SCRV	SOC	D2SD	TONR
LDL2	SEL		D3SD	
LDLG	SNEG		DFF	
LPF	S RTP		JKFF	
MAVE	SSUM		OSFI	
MAXC	TOT			
MINC	UPDN			
MSTD				
MUX				

## Timing Modes

These process control and drives instructions support different timing modes.

DEDT	LDLG	RLIM
DERV	LPF	SCRV
HPF	NTCH	SOC
INTG	PI	TOT
LDL2	PIDE	

There are three different timing modes:

Timing Mode	Description						
periodic	<p>Periodic mode is the default mode and is suitable for most control applications. We recommend that you place the instructions that use this mode in a routine that executes in a periodic task. The delta time (DeltaT) for the instruction is determined as follows:</p> <table> <tr> <th>If The Instruction Executes In a</th><th>Then DeltaT Equals</th></tr> <tr> <td>periodic task</td><td>period of the task</td></tr> <tr> <td>event or continuous task</td><td> <p>elapsed time since the previous execution</p> <p>The controller truncates the elapsed time to whole milliseconds (ms). For example, if the elapsed time = 10.5 ms, the controller sets DeltaT = 10 ms.</p> </td></tr> </table> <p>The update of the process input needs to be synchronized with the execution of the task or sampled 5-10 times faster than the task executes in order to minimize the sampling error between the input and the instruction.</p>	If The Instruction Executes In a	Then DeltaT Equals	periodic task	period of the task	event or continuous task	<p>elapsed time since the previous execution</p> <p>The controller truncates the elapsed time to whole milliseconds (ms). For example, if the elapsed time = 10.5 ms, the controller sets DeltaT = 10 ms.</p>
If The Instruction Executes In a	Then DeltaT Equals						
periodic task	period of the task						
event or continuous task	<p>elapsed time since the previous execution</p> <p>The controller truncates the elapsed time to whole milliseconds (ms). For example, if the elapsed time = 10.5 ms, the controller sets DeltaT = 10 ms.</p>						
oversample	<p>In oversample mode, the delta time (DeltaT) used by the instruction is the value written into the OversampleDT parameter of the instruction. If the process input has a time stamp value, use the real time sampling mode instead.</p> <p>Add logic to your program to control when the instruction executes. For example, you can use a timer set to the OversampleDeltaT value to control the execution by using the EnableIn input of the instruction.</p> <p>The process input needs to be sampled 5-10 times faster than the instruction is executed in order to minimize the sampling error between the input and the instruction.</p>						

Timing Mode	Description
real time sampling	<p>In the real time sampling mode, the delta time (DeltaT) used by the instruction is the difference between two time stamp values that correspond to the updates of the process input. Use this mode when the process input has a time stamp associated with its updates and you need precise coordination.</p> <p>The time stamp value is read from the tag name entered for the RTTimeStamp parameter of the instruction. Normally this tag name is a parameter on the input module associated with the process input.</p> <p>The instruction compares the configured RTSTime value (expected update period) against the calculated DeltaT to determine if every update of the process input is being read by the instruction. If DeltaT is not within 1 millisecond of the configuration time, the instruction sets the RTSMissed status bit to indicate that a problem exists reading updates for the input on the module.</p>

Time-based instructions require a constant value for DeltaT in order for the control algorithm to properly calculate the process output. If DeltaT varies, a discontinuity occurs in the process output. The severity of the discontinuity depends on the instruction and range over which DeltaT varies. A discontinuity occurs if the:

- instruction is not executed during a scan.
- instruction is executed multiple times during a task.
- task is running and the task scan rate or the sample time of the process input changes.
- user changes the time base mode while the task is running.
- Order parameter is changed on a filter block while the task is running. Changing the Order parameter selects a different control algorithm within the instruction.

## Common instruction parameters for timing modes

The instructions that support time base modes have these input and output parameters:

### *Input parameters*

Input Parameter	Data Type	Description	
TimingMode	DINT	Selects timing execution mode.	
		<b>Value:</b>	<b>Description:</b>
		0	periodic mode
		1	oversample mode
		2	real time sampling mode
		valid = 0 to 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 to 4194.303 seconds	
		default = 0.0	

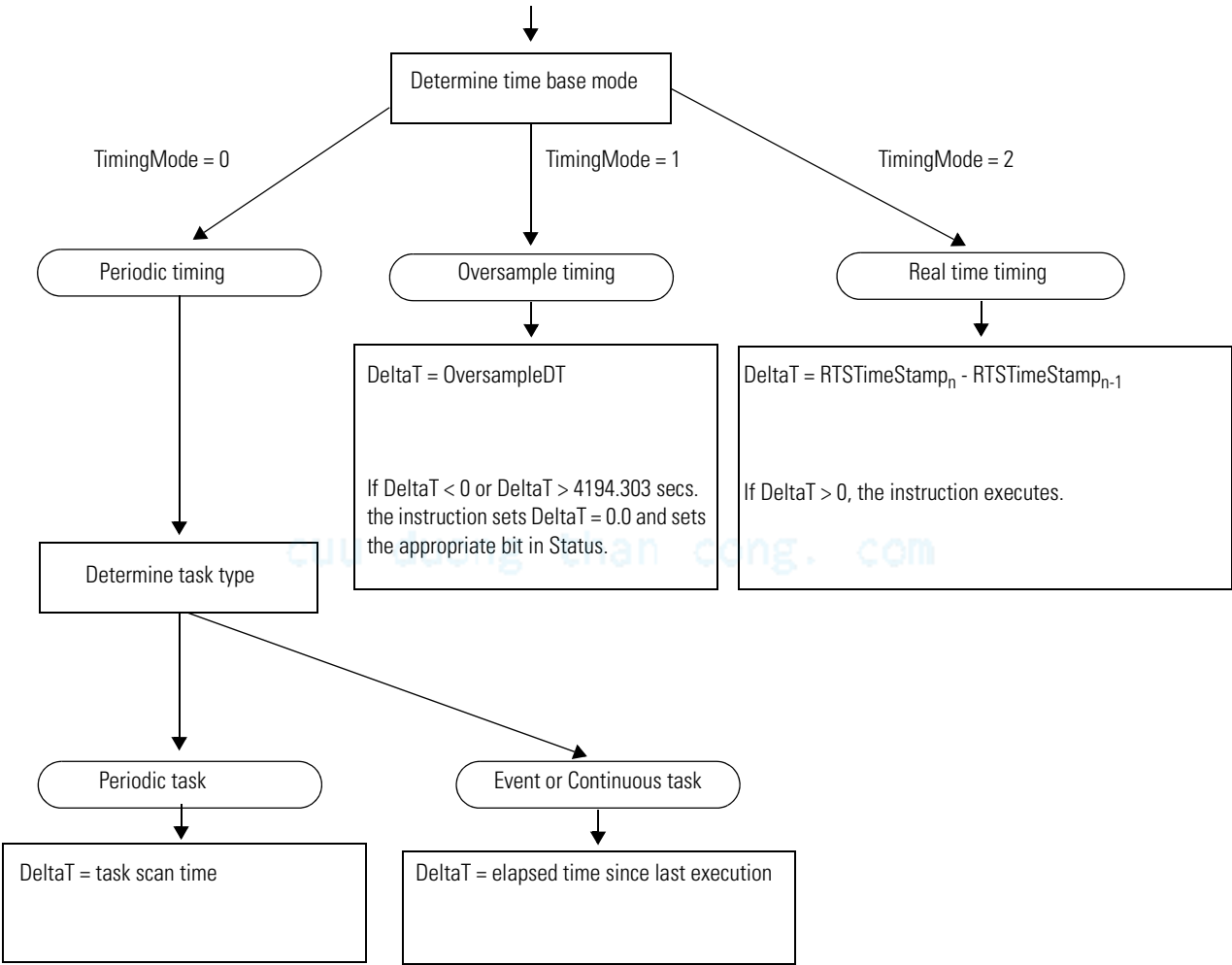
Input Parameter	Data Type	Description
RTSTime	DINT	Module update period for real time sampling timing. The expected DeltaT update period is in milliseconds. The update period is normally the value that was used to configure the module's update time. If invalid, the instruction sets the appropriate bit in Status and disables RTSMissed checking.  valid = 1 to 32,767ms  default = 1
RTTimeStamp	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 to 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: $\Delta T = (RTTimeStamp_n - RTTimeStamp_{n-1})$
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 $ABS   \Delta T - RTSTime   > 1$ (.001 second).
RTTimeInv (Status.29)	BOOL	Invalid RTSTime value.
RTTimeStampInv (Status.30)	BOOL	Invalid RTTimeStamp value.
DeltaTInv (Status.31)	BOOL	Invalid DeltaT value.

### Overview of timing modes

The following diagram shows how an instruction determines the appropriate timing mode.



## Program/Operator Control

Several instructions support the concept of Program/Operator control. These instructions include:

- Enhanced Select (ESEL)
- Totalizer (TOT)
- Enhanced PID (PIDE)
- Ramp/Soak (RMPS)
- Discrete 2-State Device (D2SD)
- Discrete 3-State Device (D3SD)

Program/Operator control lets you control these instructions simultaneously from both your user program and from an operator interface device. When in Program control, the instruction is controlled by the Program inputs to the instruction; when in Operator control, the instruction is controlled by the Operator inputs to the instruction.

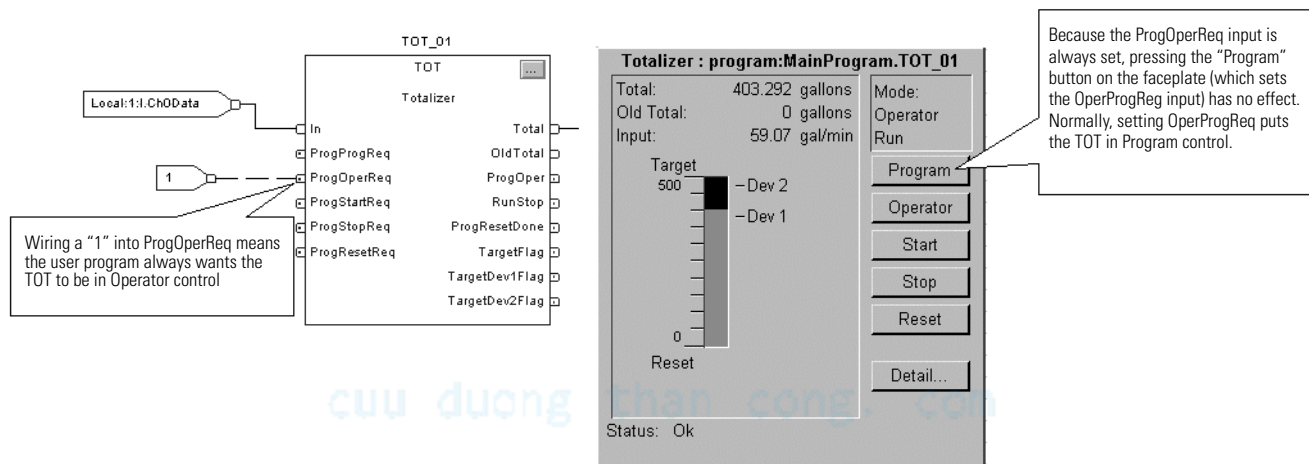
Program or Operator control is determined by using these inputs:

Input	Description
.ProgProgReq	A program request to go to Program control.
.ProgOperReq	A program request to go to Operator control.
.OperProgReq	An operator request to go to Program control.
.OperOperReq	An operator request to go to Operator control.

To determine whether an instruction is in Program or Control control, examine the ProgOper output. If ProgOper is set, the instruction is in Program control; if ProgOper is cleared, the instruction is in Operator control.

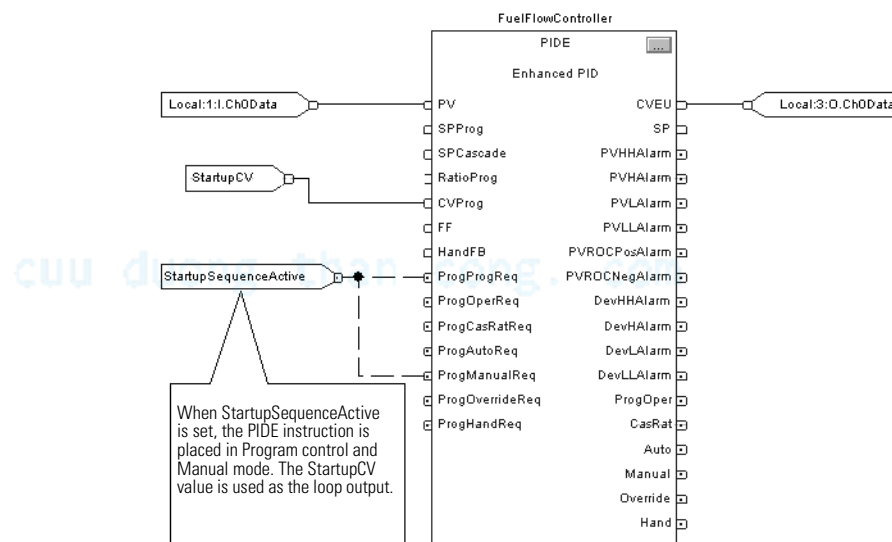
Operator control takes precedence over Program control if both input request bits are set. For example, if ProgProgReq and ProgOperReq are both set, the instruction goes to Operator control.

The Program request inputs take precedence over the Operator request inputs. This provides the capability to use the ProgProgReq and ProgOperReq inputs to “lock” an instruction in a desired control. For example, let’s assume that a Totalizer instruction will always be used in Operator control, and your user program will never control the running or stopping of the Totalizer. In this case, you could wire a literal value of 1 into the ProgOperReq. This would prevent the operator from ever putting the Totalizer into Program control by setting the OperProgReq from an operator interface device.





Likewise, constantly setting the ProgProgReq can “lock” the instruction into Program control. This is useful for automatic startup sequences when you want the program to control the action of the instruction without worrying about an operator inadvertently taking control of the instruction. In this example, you have the program set the ProgProgReq input during the startup, and then clear the ProgProgReq input once the startup was complete. Once the ProgProgReq input is cleared, the instruction remains in Program control until it receives a request to change. For example, the operator could set the OperOperReq input from a faceplate to take over control of that instruction. The following example shows how to lock an instruction into Program control.



Operator request inputs to an instruction are always cleared by the instruction when it executes. This allows operator interfaces to work with these instructions by merely setting the desired mode request bit. You don't have to program the operator interface to reset the request bits. For example, if an operator interface sets the OperAutoReq input to a PIDE instruction, when the PIDE instruction executes, it determines what the appropriate response should be and clears the OperAutoReq.

Program request inputs are not normally cleared by the instruction because these are normally wired as inputs into the instruction. If the instruction clears these inputs, the input would just get set again by the wired input. There might be situations where you want to use other logic to set the Program requests in such a manner that you want the Program requests to be cleared by the instruction. In this case, you can set the ProgValueReset input and the instruction will always clear the Program mode request inputs when it executes.

In this example, a rung of ladder logic in another routine is used to one-shot latch a ProgAutoReq to a PIDE instruction when a pushbutton is pushed. Because the PIDE instruction automatically clears the Program mode requests, you don't have to write any ladder logic to clear the ProgAutoReq after the routine executes, and the PIDE instruction will receive only one request to go to Auto every time the pushbutton is pressed.

When the TIC101AutoReq Pushbutton is pressed, one-shot latch ProgAutoReq for the PIDE instruction TIC101. TIC101 has been configured with the ProgValueReset input set, so when the PIDE instruction executes, it automatically clears ProgAutoReq.



# Structured Text Programming

## Introduction

This appendix describes issues that are unique with structured text programming. Review the information in this appendix to make sure you understand how your structured text programming will execute.

For Information About	See Page
Structured Text Syntax	659
Assignments	661
Expressions	663
Instructions	670
Constructs	671
Comments	687

## 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	Use an assignment statement to assign values to tags.	<code>tag := expression;</code>
(see page 661)	The := operator is the assignment operator. Terminate the assignment with a semi colon “;”.	

Term	Definition	Examples
expression (see page 28-663)	<p>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).</p> <p>An expression contains:</p> <p>tags                    A named area of the memory where data is stored (BOOL, SINT, INT, DINT, REAL, string).</p> <p>immediates            A constant value.</p> <p>operators              A symbol or mnemonic that specifies an operation within an expression.</p> <p>functions              When executed, a function yields one value. Use parentheses to contain the operand of a function.</p> <p>Even though their syntax is similar, functions differ from instructions in that functions can only be used in expressions. Instructions cannot be used in expressions.</p>	<p><i>value1</i></p> <p><i>4</i></p> <p><i>tag1 + tag2</i></p> <p><i>tag1 &gt;= value1</i></p> <p><i>function(tag1)</i></p>
instruction (see page 28-670)	<p>An instruction is a standalone statement.</p> <p>An instruction uses parenthesis to contain its operands.</p> <p>Depending on the instruction, there can be zero, one, or multiple operands.</p> <p>When executed, an instruction yields one or more values that are part of a data structure.</p> <p>Terminate the instruction with a semi colon “;”.</p> <p>Even though their syntax is similar, instructions differ from functions in that instructions cannot be used in expressions. Functions can only be used in expressions.</p>	<p><i>instruction( );</i></p> <p><i>instruction(operand);</i></p> <p><i>instruction(operand1, operand2, operand3);</i></p>

Term	Definition	Examples
construct (see page 28-671)	A conditional statement used to trigger structured text code (i.e, other statements). Terminate the construct with a semi colon ";".	IF . . . THEN  CASE  FOR . . . DO  WHILE . . . DO  REPEAT . . . UNTIL  EXIT
comment (see page 687)	Text that explains or clarifies what a section of structured text does.  <ul style="list-style-type: none"> <li>• Use comments to make it easier to interpret the structured text.</li> <li>• Comments do not affect the execution of the structured text.</li> <li>• Comments can appear anywhere in structured text.</li> </ul>	//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												
<i>tag</i>	represents the tag that is getting the new value the tag must be a BOOL, SINT, INT, DINT, or REAL												
:=	is the assignment symbol												
<i>expression</i>	represents the new value to assign to the tag												
<table> <tr> <th>If <i>tag</i> is this data type:</th><th>Use this type of expression:</th></tr> <tr> <td>BOOL</td><td>BOOL expression</td></tr> <tr> <td>SINT</td><td>numeric expression</td></tr> <tr> <td>INT</td><td></td></tr> <tr> <td>DINT</td><td></td></tr> <tr> <td>REAL</td><td></td></tr> </table>		If <i>tag</i> is this data type:	Use this type of expression:	BOOL	BOOL expression	SINT	numeric expression	INT		DINT		REAL	
If <i>tag</i> is this data type:	Use this type of expression:												
BOOL	BOOL expression												
SINT	numeric expression												
INT													
DINT													
REAL													
;	ends the assignment												

The *tag* retains the assigned value until another assignment changes the value.

The expression can be simple, such as an immediate value or another tag name, or the expression can be complex and include several operators and/or functions. See the next section “Expressions” on page 663 for details.

## Specify a non-retentive assignment

The non-retentive assignment is different from the regular assignment described above in that the tag in a non-retentive assignment is reset to zero each time the controller:

- enters the RUN mode
- leaves the step of an SFC if you configure the SFC for *Automatic reset* (This applies only if you embed the assignment in the action of the step or use the action to call a structured text routine via a JSR instruction.)

A non-retentive assignment has this syntax:

*tag* [:=] *expression* ;

where:

Component	Description												
<i>tag</i>	represents the tag that is getting the new value the tag must be a BOOL, SINT, INT, DINT, or REAL												
[:=]	is the non-retentive assignment symbol												
<i>expression</i>	represents the new value to assign to the tag												
<table> <tr> <th>If <i>tag</i> is this data type:</th><th>Use this type of expression:</th></tr> <tr> <td>BOOL</td><td>BOOL expression</td></tr> <tr> <td>SINT</td><td>numeric expression</td></tr> <tr> <td>INT</td><td></td></tr> <tr> <td>DINT</td><td></td></tr> <tr> <td>REAL</td><td></td></tr> </table>		If <i>tag</i> is this data type:	Use this type of expression:	BOOL	BOOL expression	SINT	numeric expression	INT		DINT		REAL	
If <i>tag</i> is this data type:	Use this type of expression:												
BOOL	BOOL expression												
SINT	numeric expression												
INT													
DINT													
REAL													
;	ends the assignment												

## Assign an ASCII character to a string

Use the assignment operator to assign an ASCII character to an element of the DATA member of a string tag. To assign a character, specify the value of the character or specify the tag name, DATA member, and element of the character. For example:

This is OK	This is <i>not</i> OK.
<code>string1.DATA[0] := 65;</code>	<code>string1.DATA[0] := A;</code>
<code>string1.DATA[0] := string2.DATA[0];</code>	<code>string1 := string2;</code>

To add or insert a string of characters to a string tag, use either of these ASCII string instructions:

To	Use This Instruction
add characters to the end of a string	CONCAT
insert characters into a string	INSERT

## Expressions

An expression is a tag name, equation, or comparison. To write an expression, use any of the following:

- tag name that stores the value (variable)
- number that you enter directly into the expression (immediate value)
- functions, such as: ABS, TRUNC
- operators, such as: +, -, <, >, And, Or

As you write expressions, follow these general rules:

- Use any combination of upper-case and lower-case letter. For example, these three variations of "AND" are acceptable: AND, And, and.
- For more complex requirements, use parentheses to group expressions within expressions. This makes the whole expression easier to read and ensures that the expression executes in the desired sequence. See “Determine the order of execution” on page 669.

In structured text, you use two types of expressions:

**BOOL expression:** An expression that produces either the BOOL value of 1 (true) or 0 (false).

- A bool expression uses bool tags, relational operators, and logical operators to compare values or check if conditions are true or false. For example, `tag1>65`.
- A simple bool expression can be a single BOOL tag.
- Typically, you use bool expressions to condition the execution of other logic.

**Numeric expression:** An expression that calculates an integer or floating-point value.

- A numeric expression uses arithmetic operators, arithmetic functions, and bitwise operators. For example, `tag1+5`.
- Often, you nest a numeric expression within a bool expression. For example, `(tag1+5)>65`.

Use the following table to choose operators for your expressions:

If You Want To	Then
Calculate an arithmetic value	"Use arithmetic operators and functions" on page 665.
Compare two values or strings	"Use relational operators" on page 666.
Check if conditions are true or false	"Use logical operators" on page 668.
Compare the bits within values	"Use bitwise operators" on page 669.



## 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 ( <i>numeric_expression</i> )	DINT, REAL
arc cosine	ACOS ( <i>numeric_expression</i> )	REAL
arc sine	ASIN ( <i>numeric_expression</i> )	REAL
arc tangent	ATAN ( <i>numeric_expression</i> )	REAL
cosine	COS ( <i>numeric_expression</i> )	REAL
radians to degrees	DEG ( <i>numeric_expression</i> )	DINT, REAL
natural log	LN ( <i>numeric_expression</i> )	REAL
log base 10	LOG ( <i>numeric_expression</i> )	REAL
degrees to radians	RAD ( <i>numeric_expression</i> )	DINT, REAL
sine	SIN ( <i>numeric_expression</i> )	REAL
square root	SQRT ( <i>numeric_expression</i> )	DINT, REAL
tangent	TAN ( <i>numeric_expression</i> )	REAL
truncate	TRUNC ( <i>numeric_expression</i> )	DINT, REAL

For example:

Use This Format	Example	
	For This Situation	You'd Write
<i>value1 operator value2</i>	If <i>gain_4</i> and <i>gain_4_adj</i> are DINT tags and your specification says: "Add 15 to <i>gain_4</i> and store the result in <i>gain_4_adj</i> ."	<code>gain_4_adj := gain_4+15;</code>
<i>operator value1</i>	If <i>alarm</i> and <i>high_alarm</i> are DINT tags and your specification says: "Negate <i>high_alarm</i> and store the result in <i>alarm</i> ."	<code>alarm:= -high_alarm;</code>
<i>function(numeric_expression)</i>	If <i>overtravel</i> and <i>overtravel_POS</i> are DINT tags and your specification says: "Calculate the absolute value of <i>overtravel</i> and store the result in <i>overtravel_POS</i> ."	<code>overtravel_POS := ABS(overtravel);</code>
<i>value1 operator (function((value2+value3)/2))</i>	If <i>adjustment</i> and <i>position</i> are DINT tags and <i>sensor1</i> and <i>sensor2</i> are REAL tags and your specification says: "Find the absolute value of the average of <i>sensor1</i> and <i>sensor2</i> , add the <i>adjustment</i> , and store the result in <i>position</i> ."	<code>position := adjustment + ABS((sensor1 + sensor2)/2);</code>

## Use relational operators

Relational operators compare two values or strings to provide a true or false result. The result of a relational operation is a BOOL value:

If The Comparison Is	The Result Is
true	1
false	0

Use the following relational operators:

For This Comparison	Use This Operator	Optimal Data Type
equal	=	DINT, REAL, string
less than	<	DINT, REAL, string
less than or equal	<=	DINT, REAL, string
greater than	>	DINT, REAL, string
greater than or equal	>=	DINT, REAL, string
not equal	<>	DINT, REAL, string

For example:

Use This Format	Example	
	For This Situation	You'd Write
<i>value1 operator value2</i>	If <i>temp</i> is a DINT tag and your specification says: "If <i>temp</i> is less than 100° then..."	IF temp<100 THEN...
<i>stringtag1 operator stringtag2</i>	If <i>bar_code</i> and <i>dest</i> are string tags and your specification says: "If <i>bar_code</i> equals <i>dest</i> then..."	IF bar_code=dest THEN...
<i>char1 operator char2</i>  To enter an ASCII character directly into the expression, enter the decimal value of the character.	If <i>bar_code</i> is a string tag and your specification says: "If <i>bar_code</i> .DATA[0] equals 'A' then..."	IF bar_code.DATA[0]=65 THEN...
<i>bool_tag := bool_expressions</i>	If <i>count</i> and <i>length</i> are DINT tags, <i>done</i> is a BOOL tag, and your specification says "If <i>count</i> is greater than or equal to <i>length</i> , you are done counting."	done := (count >= length);

### How Strings Are Evaluated

The hexadecimal values of the ASCII characters determine if one string is less than or greater than another string.

- When the two strings are sorted as in a telephone directory, the order of the strings determines which one is greater.

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

↑  
 l  
e  
s  
s  
e  
r  
 ↑  
 g  
r  
e  
a  
t  
e  
r  
 ↓

— AB < B  
 — a > B

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

For the decimal value and hex code of a character, see the back cover of this manual.

## Use logical operators

Logical operators let you check if multiple conditions are true or false. The result of a logical operation is a BOOL value:

If The Comparison Is	The Result Is
true	1
false	0

Use the following logical operators:

For	Use This Operator	Data Type
logical AND	&, AND	BOOL
logical OR	OR	BOOL
logical exclusive OR	XOR	BOOL
logical complement	NOT	BOOL

For example:

Use This Format	Example	
	For This Situation	You'd Write
<i>BOOLtag</i>	If <i>photoeye</i> is a BOOL tag and your specification says: "If <i>photoeye</i> is on then..."	IF <i>photoeye</i> THEN...
NOT <i>BOOLtag</i>	If <i>photoeye</i> is a BOOL tag and your specification says: "If <i>photoeye</i> is off then..."	IF NOT <i>photoeye</i> THEN...
<i>expression1</i> & <i>expression2</i>	If <i>photoeye</i> is a BOOL tag, <i>temp</i> is a DINT tag, and your specification says: "If <i>photoeye</i> is on and <i>temp</i> is less than 100° then..."	IF <i>photoeye</i> & ( <i>temp</i> <100) THEN...
<i>expression1</i> OR <i>expression2</i>	If <i>photoeye</i> is a BOOL tag, <i>temp</i> is a DINT tag, and your specification says: "If <i>photoeye</i> is on or <i>temp</i> is less than 100° then..."	IF <i>photoeye</i> OR ( <i>temp</i> <100) THEN...
<i>expression1</i> XOR <i>expression2</i>	If <i>photoeye1</i> and <i>photoeye2</i> are BOOL tags and your specification says: "If: <ul style="list-style-type: none"> <li>• <i>photoeye1</i> is on while <i>photoeye2</i> is off or</li> <li>• <i>photoeye1</i> is off while <i>photoeye2</i> is on</li> </ul> then..."	IF <i>photoeye1</i> XOR <i>photoeye2</i> THEN...
<i>BOOLtag</i> := <i>expression1</i> & <i>expression2</i>	If <i>photoeye1</i> and <i>photoeye2</i> are BOOL tags, <i>open</i> is a BOOL tag, and your specification says: "If <i>photoeye1</i> and <i>photoeye2</i> are both on, set <i>open</i> to true".	<i>open</i> := <i>photoeye1</i> & <i>photoeye2</i> ;

## Use bitwise operators

Bitwise operators manipulate the bits within a value based on two values.

For	Use This Operator	Optimal Data Type
bitwise AND	&, AND	DINT
bitwise OR	OR	DINT
bitwise exclusive OR	XOR	DINT
bitwise complement	NOT	DINT

For example:

Use This Format	Example	
	For This Situation	You'd Write
<i>value1 operator value2</i>	If <i>input1</i> , <i>input2</i> , and <i>result1</i> are DINT tags and your specification says: "Calculate the bitwise result of <i>input1</i> and <i>input2</i> . Store the result in <i>result1</i> ."	<code>result1 := input1 AND input2;</code>

## Determine the order of execution

The operations you write into an expression are performed in a prescribed order, not necessarily from left to right.

- Operations of equal order are performed from left to right.
- If an expression contains multiple operators or functions, group the conditions in parenthesis "( )" . This ensures the correct order of execution and makes it easier to read the expression.

Order	Operation
1.	( )
2.	function (...)
3.	**
4.	– (negate)
5.	NOT
6.	*, /, MOD
7.	+, - (subtract)
8.	<, <=, >, >=
9.	=, <>
10.	&, AND
11.	XOR
12.	OR

## Instructions

Structured text statements can also be instructions. See the Locator Table at the beginning of this manual for a list of the instructions available in structured text. A structured text instruction executes each time it is scanned. A structured text instruction within a construct executes every time the conditions of the construct are true. If the conditions of the construct are false, the statements within the construct are not scanned. There is no rung-condition or state transition that triggers execution.

This differs from function block instructions that use EnableIn to trigger execution. Structured text instructions execute as if EnableIn is always set.

This also differs from relay ladder instructions that use rung-condition-in to trigger execution. Some relay ladder instructions only execute when rung-condition-in toggles from false to true. These are transitional relay ladder instructions. In structured text, instructions will execute each time they are scanned unless you pre-condition the execution of the structured text instruction.

For example, the ABL instruction is a transitional instruction in relay ladder. In this example, the ABL instruction only executes on a scan when *tag\_xic* transitions from cleared to set. The ABL instruction does not execute when *tag\_xic* stays set or when *tag\_xic* is cleared.



In structured text, if you write this example as:

```
IF tag_xic THEN ABL(0,serial_control);

END_IF;
```

the ABL instruction will execute every scan that *tag\_xic* is set, not just when *tag\_xic* transitions from cleared to set.

If you want the ABL instruction to execute only when *tag\_xic* transitions from cleared to set, you have to condition the structured text instruction. Use a one shot to trigger execution.

```
osri_1.InputBit := tag_xic;
OSRI(osri_1);

IF (osri_1.OutputBit) THEN
    ABL(0,serial_control);
END_IF;
```

## Constructs

Constructs can be programmed singly or nested within other constructs.

If You Want To	Use This Construct	Available In These Languages	See Page
do something if or when specific conditions occur	IF...THEN	structured text	672
select what to do based on a numerical value	CASE...OF	structured text	675
do something a specific number of times before doing anything else	FOR...DO	structured text	678
keep doing something as long as certain conditions are true	WHILE...DO	structured text	681
keep doing something until a condition is true	REPEAT...UNTIL	structured text	684

## Some key words are reserved for future use

These constructs are not available:

- GOTO
- REPEAT

RSLogix 5000 software 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:

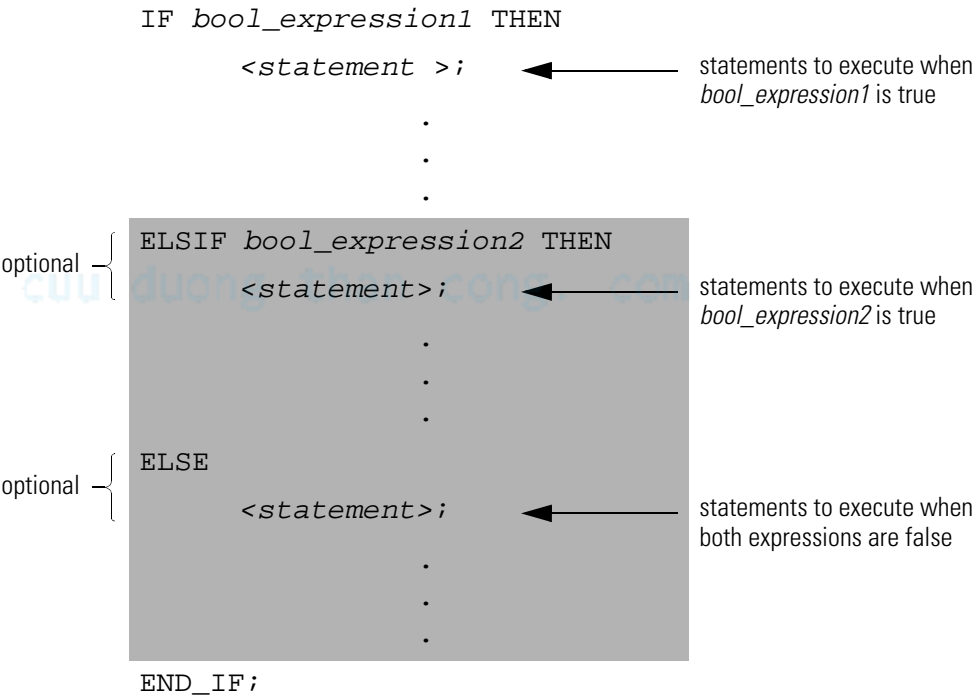


```
IF bool_expression THEN
    <statement>;
END_IF;
```

## Structured Text

Operand	Type	Format	Enter
bool_ expression	BOOL	tag expression	BOOL tag or expression that evaluates to a BOOL value (BOOL expression)

**Description:** The syntax is:



To use ELSIF or ELSE, follow these guidelines:

1. To select from several possible groups of statements, add one or more ELSIF statements.
  - Each ELSIF represents an alternative path.
  - Specify as many ELSIF paths as you need.
  - The controller executes the first true IF or ELSIF and skips the rest of the ELSIFs and the ELSE.
2. To do something when all of the IF or ELSIF conditions are false, add an ELSE statement.



The following table summarizes different combinations of IF, THEN, ELSIF, and ELSE.

If You Want To	And	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 conveyor = off (0) alarm = on (1)	IF rejects > 3 THEN conveyor := 0; alarm := 1; END_IF;

### Example 2: IF...THEN...ELSE

If You Want This	Enter This Structured Text
If conveyor direction contact = forward (1) then light = off Otherwise light = on	IF conveyor_direction THEN light := 0; ELSE light [:=] 1; END_IF;

The [:=] tells the controller to clear *light* whenever the controller:

- enters the RUN mode
- leaves the step of an SFC if you configure the SFC for *Automatic reset* (This applies only if you embed the assignment in the action of the step or use the action to call a structured text routine via a JSR instruction.)

**Example 3: IF...THEN...ELSIF**

If You Want This	Enter This Structured Text
If sugar low limit switch = low (on) and sugar high limit switch = not high (on) then inlet valve = open (on) Until sugar high limit switch = high (off)	IF Sugar.Low & Sugar.High THEN  Sugar.Inlet [:=] 1;  ELSIF NOT(Sugar.High) THEN  Sugar.Inlet := 0;  END_IF;

The [:=] tells the controller to clear *Sugar.Inlet* whenever the controller:

- enters the RUN mode
- leaves the step of an SFC if you configure the SFC for *Automatic reset* (This applies only if you embed the assignment in the action of the step or use the action to call a structured text routine via a JSR instruction.)

**Example 4: IF...THEN...ELSIF...ELSE**

If You Want This	Enter This Structured Text
If tank temperature > 100 then pump = slow If tank temperature > 200 then pump = fast otherwise pump = off	IF tank.temp > 200 THEN pump.fast :=1; pump.slow :=0; pump.off :=0;  ELSIF tank.temp > 100 THEN pump.fast :=0; pump.slow :=1; pump.off :=0;  ELSE pump.fast :=0; pump.slow :=0; pump.off :=1;  END_IF;

## CASE...OF

Use CASE to select what to do based on a numerical value.

### Operands:



```
CASE numeric_expression OF
    selector1: statement;
    selectorN: statement;
ELSE
    statement;
END_CASE;
```

### Structured Text

Operand	Type	Format	Enter
<i>numeric_</i>	SINT	tag	tag or expression that evaluates to a number (numeric expression)
<i>expression</i>	INT	expression	
	DINT		
	REAL		
<i>selector</i>	SINT	immediate	same type as <i>numeric_expression</i>
	INT		
	DINT		
	REAL		

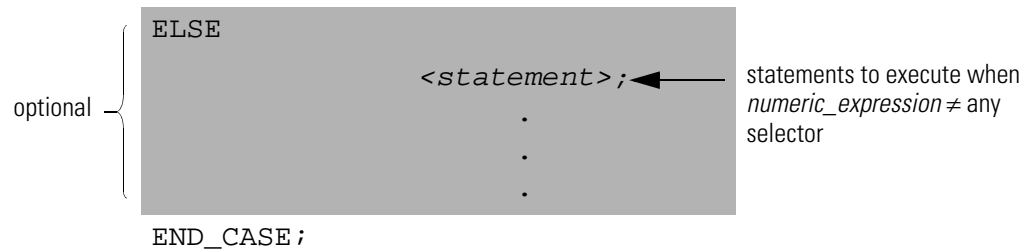
### IMPORTANT

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:

```
CASE numeric_expression OF
    selector1: <statement>; ← statements to execute when
                                numeric_expression = selector1
    .
    .
    .
    selector2: <statement>; ← statements to execute when
                                numeric_expression = selector2
    .
    .
    .
    selector3: <statement>; ← statements to execute when
                                numeric_expression = selector3
    .
    .
    .
```

specify as many alternative selector values (paths) as you need



See the table on the next page for valid selector values.

The syntax for entering the selector values is:

When Selector Is	Enter
one value	<i>value</i> : <i>statement</i>
multiple, distinct values	<i>value1</i> , <i>value2</i> , <i>valueN</i> : <i>&lt;statement&gt;</i>
	Use a comma (,) to separate each value.
a range of values	<i>value1</i> .. <i>valueN</i> : <i>&lt;statement&gt;</i>
	Use two periods (..) to identify the range.
distinct values plus a range of values	<i>valuea</i> , <i>valueb</i> , <i>value1</i> .. <i>valueN</i> : <i>&lt;statement&gt;</i>

The CASE construct is similar to a switch statement in the C or C++ programming languages. However, with the CASE construct the controller executes *only* the statements that are associated with the *first matching* selector value. Execution *always breaks after the statements of that selector* and goes to the END\_CASE statement.

**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:

- enters the RUN mode
- leaves the step of an SFC if you configure the SFC for *Automatic reset* (This applies only if you embed the assignment in the action of the step or use the action to call a structured text routine via a JSR instruction.)

## FOR...DO

Use the FOR...DO loop to do something a specific number of times before doing anything else.

### Operands:



```
FOR count:= initial_value TO
final_value BY increment DO
    <statement>;
END_FOR;
```

### Structured Text

Operand	Type	Format	Description
<i>count</i>	SINT	tag	tag to store count position as the FOR...DO executes
	INT		
	DINT		
<i>initial_value</i>	SINT	tag	must evaluate to a number
	INT	expression	specifies initial value for count
	DINT	immediate	
<i>final_value</i>	SINT	tag	specifies final value for count, which determines when to exit the loop
	INT	expression	
	DINT	immediate	
<i>increment</i>	SINT	tag	<i>(optional)</i> amount to increment count each time through the loop
	INT	expression	
	DINT	immediate	
			If you don't specify an increment, the count increments by 1.

### IMPORTANT

Make sure that you *do not* iterate within the loop too many times in a single scan.

- The controller *does not* execute any other statements in the routine until it completes the loop.
- If the time that it takes to complete the loop is greater than the watchdog timer for the task, a major fault occurs.
- Consider using a different construct, such as IF...THEN.

**Description:** The syntax is:

```

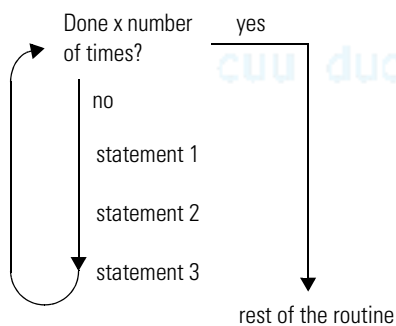
FOR count := initial_value
  TO final_value
optional { BY increment
DO
  <statement>;
optional { IF bool_expression THEN
            EXIT;
            END_IF;
END_FOR;

```

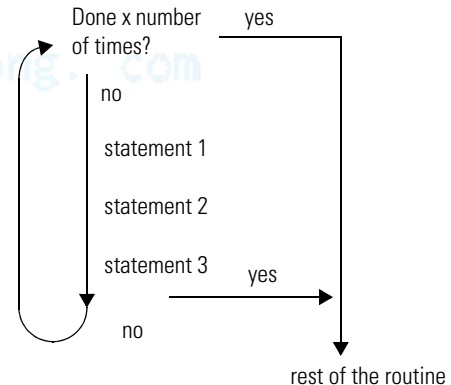
If you don't specify an increment, the loop increments by 1.

If there are conditions when you want to exit the loop early, use other statements, such as an IF...THEN construct, to condition an EXIT statement.

The following diagrams show how a FOR...DO loop executes and how an EXIT statement leaves the loop early.



**The FOR...DO loop executes a specific number of times.**



**To stop the loop before the count reaches the last value, use an EXIT statement.**

**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:	For subscript:=0 to 31 by 1 do
1. Initialize the <i>subscript</i> tag to 0.	array[subscript] := 0;
2. Clear <i>array[ subscript ]</i> . For example, when <i>subscript</i> = 5, clear <i>array[5]</i> .	End_for;
3. Add 1 to <i>subscript</i> .	
4. If <i>subscript</i> is $\leq$ to 31, repeat 2 and 3.	
Otherwise, stop.	

**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:	SIZE(Inventory,0,Inventory_Items);
<ul style="list-style-type: none"> <li>Barcode ID of the item (string data type)</li> <li>Quantity in stock of the item (DINT data type)</li> </ul>	For position:=0 to Inventory_Items - 1 do
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.	If Barcode = Inventory[position].ID then
	Quantity := Inventory[position].Qty;
	Exit;
	End_if;
	End_for;
1. Get the size (number of items) of the <i>Inventory</i> array and store the result in <i>Inventory_Items</i> (DINT tag).	
2. Initialize the <i>position</i> tag to 0.	
3. If <i>Barcode</i> matches the ID of an item in the array, then:	
a. Set the <i>Quantity</i> tag = <i>Inventory[position].Qty</i> . This produces the quantity in stock of the item.	
b. Stop.	
<i>Barcode</i> is a string tag that stores the bar code of the item for which you are searching. For example, when <i>position</i> = 5, compare <i>Barcode</i> to <i>Inventory[5].ID</i> .	
4. Add 1 to <i>position</i> .	
5. If <i>position</i> is $\leq$ to ( <i>Inventory_Items</i> -1), repeat 3 and 4. Since element numbers start at 0, the last element is 1 less than the number of elements in the array.	
Otherwise, stop.	



## WHILE...DO

Use the WHILE...DO loop to keep doing something as long as certain conditions are true.

### Operands:



```
WHILE bool_expression DO
    <statement>;
END_WHILE;
```

### Structured Text

Operand	Type	Format	Enter
bool_ expression	BOOL	tag  expression	BOOL tag or expression that evaluates to a BOOL value

### IMPORTANT

Make sure that you *do not* iterate within the loop too many times in a single scan.

- The controller *does not* execute any other statements in the routine until it completes the loop.
- If the time that it takes to complete the loop is greater than the watchdog timer for the task, a major fault occurs.
- Consider using a different construct, such as IF...THEN.

**Description:** The syntax is:

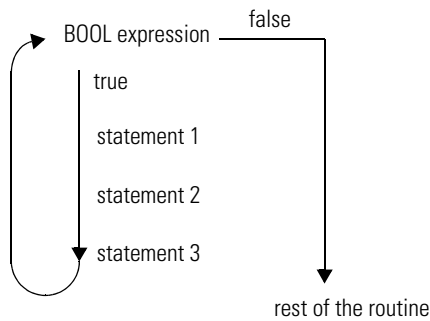
```
WHILE bool_expression1 DO
    <statement>;
    IF bool_expression2 THEN
        EXIT;
    END_IF;
END_WHILE;
```

optional {

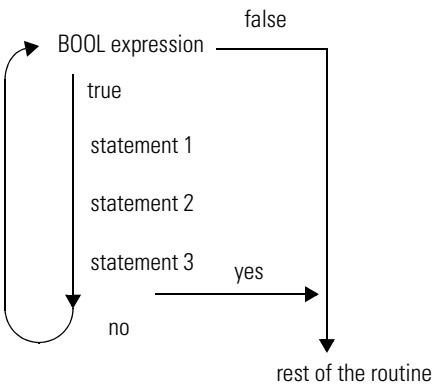
statements to execute while  
*bool\_expression1* is true

If there are conditions when you want to  
exit the loop early, use other statements,  
such as an IF...THEN construct, to  
condition an EXIT statement.

The following diagrams show how a WHILE...DO loop executes and how an EXIT statement leaves the loop early.



While the *bool\_expression* is true, the controller executes only the statements within the WHILE...DO loop.



To stop the loop before the conditions are true, use an EXIT statement.

**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.	<pre>pos := 0; While ((pos &lt;= 100) &amp; structarray[pos].value &lt;&gt; targetvalue)) do     pos := pos + 2;     String_tag.DATA[pos] := SINT_array[pos]; end_while;</pre>
This differs from the REPEAT...UNTIL loop because the REPEAT...UNTIL loop executes the statements in the construct and then determines if the conditions are true before executing the statements again. The statements in a REPEAT...UNTIL loop are always executed at least once. The statements in a WHILE...DO loop might never be executed.	

**Example 2:**

If You Want This	Enter This Structured Text
<p>Move ASCII characters from a SINT array into a string tag. (In a SINT array, each element holds one character.) Stop when you reach the carriage return.</p> <ol style="list-style-type: none"> <li>1. Initialize <i>Element_number</i> to 0.</li> <li>2. Count the number of elements in <i>SINT_array</i> (array that contains the ASCII characters) and store the result in <i>SINT_array_size</i> (DINT tag).</li> <li>3. If the character at <i>SINT_array[element_number]</i> = 13 (decimal value of the carriage return), then stop.</li> <li>4. Set <i>String_tag[element_number]</i> = the character at <i>SINT_array[element_number]</i>.</li> <li>5. Add 1 to <i>element_number</i>. This lets the controller check the next character in <i>SINT_array</i>.</li> <li>6. Set the Length member of <i>String_tag</i> = <i>element_number</i>. (This records the number of characters in <i>String_tag</i> so far.)</li> <li>7. If <i>element_number</i> = <i>SINT_array_size</i>, then stop. (You are at the end of the array and it does not contain a carriage return.)</li> <li>8. Go to 3.</li> </ol>	<pre> element_number := 0; SIZE(SINT_array, 0, SINT_array_size); While SINT_array[element_number] &lt;&gt; 13 do     String_tag.DATA[element_number] :=         SINT_array[element_number];     element_number := element_number + 1; String_tag.LEN := element_number; If element_number = SINT_array_size then     exit; end_if; end_while; </pre>

## REPEAT...UNTIL

Use the REPEAT...UNTIL loop to keep doing something until conditions are true.

### Operands:



REPEAT

`<statement>;`

UNTIL *bool\_expression*

END\_REPEAT;

### Structured Text

Operand	Type	Format	Enter
bool_ expression	BOOL	tag expression	BOOL tag or expression that evaluates to a BOOL value (BOOL expression)

### IMPORTANT

Make sure that you *do not* iterate within the loop too many times in a single scan.

- The controller *does not* execute any other statements in the routine until it completes the loop.
- If the time that it takes to complete the loop is greater than the watchdog timer for the task, a major fault occurs.
- Consider using a different construct, such as IF...THEN.

**Description:** The syntax is:

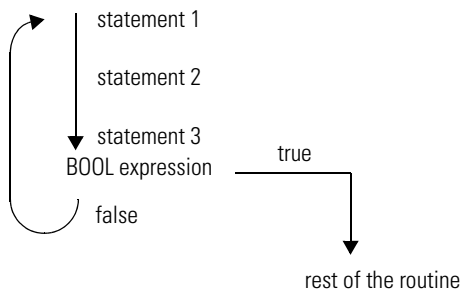
```

REPEAT
    <statement>;
    optional {
        IF bool_expression2 THEN
            EXIT;
        END_IF;
    }
UNTIL bool_expression1
END_REPEAT;
  
```

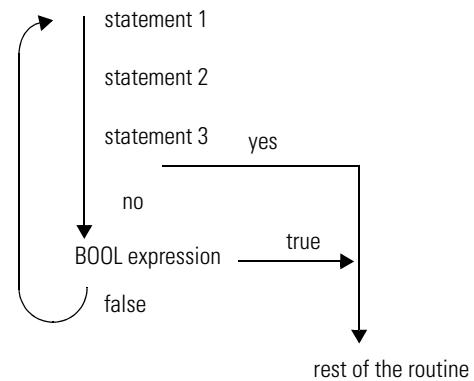
← statements to execute while *bool\_expression1* is false

← If there are conditions when you want to exit the loop early, use other statements, such as an IF...THEN construct, to condition an EXIT statement.

The following diagrams show how a 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
<p>The REPEAT...UNTIL loop executes the statements in the construct and then determines if the conditions are true before executing the statements again.</p> <p>This differs from the WHILE...DO loop because the WHILE...DO The WHILE...DO loop evaluates its conditions first. If the conditions are true, the controller then executes the statements within the loop. The statements in a REPEAT...UNTIL loop are always executed at least once. The statements in a WHILE...DO loop might never be executed.</p>	<pre> pos := -1; REPEAT     pos := pos + 2; UNTIL ((pos = 101) OR (structarray[pos].value = targetvalue)) end_repeat; </pre>

**Example 2:**

If You Want This	Enter This Structured Text
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.	<pre>element_number := 0; SIZE(SINT_array, 0, SINT_array_size); Repeat     String_tag.DATA[element_number] :=         SINT_array[element_number];     element_number := element_number + 1; String_tag.LEN := element_number; If element_number = SINT_array_size then     exit; end_if; Until SINT_array[element_number] = 13 end_repeat;</pre>
1. Initialize <i>Element_number</i> to 0.	
2. Count the number of elements in <i>SINT_array</i> (array that contains the ASCII characters) and store the result in <i>SINT_array_size</i> (DINT tag).	
3. Set <i>String_tag[element_number]</i> = the character at <i>SINT_array[element_number]</i> .	
4. Add 1 to <i>element_number</i> . This lets the controller check the next character in <i>SINT_array</i> .	
5. Set the Length member of <i>String_tag</i> = <i>element_number</i> . (This records the number of characters in <i>String_tag</i> so far.)	
6. If <i>element_number</i> = <i>SINT_array_size</i> , then stop. (You are at the end of the array and it does not contain a carriage return.)	
7. If the character at <i>SINT_array[element_number]</i> = 13 (decimal value of the carriage return), then stop.	
Otherwise, go to 3.	

---

## Comments

To make your structured text easier to interpret, add comments to it.

- Comments let you use plain language to describe how your structured text works.
- Comments do not affect the execution of the structured text.

To add comments to your structured text:

To Add A Comment	Use One Of These Formats
on a single line	<i>//comment</i>
at the end of a line of structured text	<i>(*comment*)</i>  <i>/*comment*/</i>
within a line of structured text	<i>(*comment*)</i>  <i>/*comment*/</i>
that spans more than one line	<i>(*start of comment . . . end of comment*)</i>  <i>/*start of comment . . . end of comment*/</i>

For example:

Format	Example
<code>//comment</code>	<p><b>At the beginning of a line</b></p> <pre>//Check conveyor belt direction IF conveyor_direction THEN...</pre> <p><b>At the end of a line</b></p> <pre>ELSE //If conveyor isn't moving, set alarm light light := 1; END_IF;</pre>
<code>(*comment*)</code>	<pre>Sugar.Inlet[:=]1;(*open the inlet*)  IF Sugar.Low (*low level LS*)&amp; Sugar.High (*high level LS*)THEN...</pre> <p><i>(Controls the speed of the recirculation pump. The speed depends on the temperature in the tank.)</i></p> <pre>IF tank.temp &gt; 200 THEN...</pre>
<code>/*comment*/</code>	<pre>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);</pre>



## ASCII Character Codes

Character	Dec	Hex	Character	Dec	Hex	Character	Dec	Hex	Character	Dec	Hex
[ctrl-@] NUL	0	\$00	SPACE	32	\$20	@	64	\$40	'	96	\$60
[ctrl-A] SOH	1	\$01	!	33	\$21	A	65	\$41	a	97	\$61
[ctrl-B] STX	2	\$02	"	34	\$22	B	66	\$42	b	98	\$62
[ctrl-C] ETX	3	\$03	#	35	\$23	C	67	\$43	c	99	\$63
[ctrl-D] EOT	4	\$04	\$	36	\$24	D	68	\$44	d	100	\$64
[ctrl-E] ENQ	5	\$05	%	37	\$25	E	69	\$45	e	101	\$65
[ctrl-F] ACK	6	\$06	&	38	\$26	F	70	\$46	f	102	\$66
[ctrl-G] BEL	7	\$07	'	39	\$27	G	71	\$47	g	103	\$67
[ctrl-H] BS	8	\$08	(	40	\$28	H	72	\$48	h	104	\$68
[ctrl-I] HT	9	\$09	)	41	\$29	I	73	\$49	i	105	\$69
[ctrl-J] LF	10	\$1 (\$0A)	*	42	\$2A	J	74	\$4A	j	106	\$6A
[ctrl-K] VT	11	\$0B	+	43	\$2B	K	75	\$4B	k	107	\$6B
[ctrl-L] FF	12	\$0C	,	44	\$2C	L	76	\$4C	l	108	\$6C
[ctrl-M] CR	13	\$r (\$0D)	-	45	\$2D	M	77	\$4D	m	109	\$6D
[ctrl-N] SO	14	\$0E	.	46	\$2E	N	78	\$4E	n	110	\$6E
[ctrl-O] SI	15	\$0F	/	47	\$2F	O	79	\$4F	o	111	\$6F
[ctrl-P] DLE	16	\$10	0	48	\$30	P	80	\$50	p	112	\$70
[ctrl-Q] DC1	17	\$11	1	49	\$31	Q	81	\$51	q	113	\$71
[ctrl-R] DC2	18	\$12	2	50	\$32	R	82	\$52	r	114	\$72
[ctrl-S] DC3	19	\$13	3	51	\$33	S	83	\$53	s	115	\$73
[ctrl-T] DC4	20	\$14	4	52	\$34	T	84	\$54	t	116	\$74
[ctrl-U] NAK	21	\$15	5	53	\$35	U	85	\$55	u	117	\$75
[ctrl-V] SYN	22	\$16	6	54	\$36	V	86	\$56	v	118	\$76
[ctrl-W] ETB	23	\$17	7	55	\$37	W	87	\$57	w	119	\$77
[ctrl-X] CAN	24	\$18	8	56	\$38	X	88	\$58	x	120	\$78
[ctrl-Y] EM	25	\$19	9	57	\$39	Y	89	\$59	y	121	\$79
[ctrl-Z] SUB	26	\$1A	:	58	\$3A	Z	90	\$5A	z	122	\$7A
ctrl-[ ESC	27	\$1B	;	59	\$3B	[	91	\$5B	{	123	\$7B
[ctrl-\] FS	28	\$1C	<	60	\$3C	\	92	\$5C		124	\$7C
ctrl-] GS	29	\$1D	=	61	\$3D	]	93	\$5D	}	125	\$7D
[ctrl-^] RS	30	\$1E	>	62	\$3E	^	94	\$5E	~	126	\$7E
[ctrl-_] US	31	\$1F	?	63	\$3F	_	95	\$5F	DEL	127	\$7F

cuu duong than cong. com

cuu duong than cong. com

## **A**

**ABL instruction** 575  
**ABS instruction** 279  
**absolute value** 279  
**ACB instruction** 578  
**ACL instruction** 581  
**ACS instruction** 536  
**ADD instruction** 253  
**addition** 253  
**advanced math instructions**  
     introduction 545  
     LN 546  
     LOG 549  
     XPY 552  
**AFI instruction** 458  
**AHL instruction** 583  
**alarm instruction** 31, 47  
**alarms** 507  
**all mode** 332  
**ALMA instruction** 47  
**ALMD instruction** 31  
**always false instruction** 458  
**analog alarm** 47  
**AND instruction** 305  
**arc cosine** 536  
**arc sine** 532  
**arc tangent** 540  
**ARD instruction** 587  
**arithmetic operators**  
     structured text 665  
**arithmetic status flags**  
     overflow 649  
**ARL instruction** 591  
**array instructions**  
     AVE 368  
     BSL 388  
     BSR 392  
     COP 358  
     CPS 358  
     DDT 488  
     FAL 337  
     FBC 480  
     FFL 396  
     FFU 402  
     file/misc. 331  
     FLL 364  
     FSC 349  
     LFL 408  
     LFU 414  
     mode of operation 332  
     RES 136

sequencer 421  
 shift 387  
 SIZE 384  
 SQI 422  
 SQL 430  
 SQO 426  
 SRT 373  
 STD 378

## **ASCII**

structured text assignment 663

**ASCII chars in buffer** 578  
**ASCII clear buffer** 581  
**ASCII handshake lines** 583  
**ASCII instructions**  
     ABL 575  
     ACB 578  
     ACL 581  
     AHL 583  
     ARD 587  
     ARL 591  
     AWA 595  
     AWT 600  
     CONCAT 608  
     DELETE 610  
     DTOS 626  
     FIND 612  
     INSERT 614  
     LOWER 633  
     MID 616  
     RTOS 629  
     STOD 622  
     STOR 624  
     SWPB 301  
     UPPER 631  
**ASCII read** 587  
**ASCII read line** 591  
**ASCII test for buffer line** 575  
**ASCII write** 600  
**ASCII write append** 595  
**ASN instruction** 532  
**assignment**  
     ASCII character 663  
     non-retentive 662  
     retentive 661  
**assume data available** 645, 647, 648  
**ATN instruction** 540  
**attributes**  
     converting data types 635  
     immediate values 635  
**AVE instruction** 368  
**average** 368  
**AWA instruction** 595

**AWT instruction** 600

## B

**BAND** 319

**bit field distribute** 293

**bit field distribute with target** 296

**bit instructions**

introduction 69

ONS 80

OSF 86

OSFI 92

OSR 83

OSRI 89

OTE 74

OTL 76

OTU 78

XIO 72

**bit shift left** 388

**bit shift right** 392

**bitwise AND** 305

**bitwise exclusive OR** 311

**bitwise NOT** 315

**bitwise operators**

structured text 669

**bitwise OR** 308

**BNOT** 328

**BOOL expression**

structured text 663

**Boolean AND** 319

**Boolean Exclusive OR** 325

**Boolean NOT** 328

**Boolean OR** 322

**BOR** 322

**break** 475

**BRK instruction** 475

**BSL instruction** 388

**BSR instruction** 392

**BTD instruction** 293

**BTDT instruction** 296

**BXOR** 325

## C

**cache**

connection 170

**CASE** 675

**clear** 299

**CLR instruction** 299

**CMP instruction** 207

**comments**

structured text 687

**common attributes** 635

converting data types 635

immediate values 635

**compare** 207

**compare instructions**

CMP 207

EQU 212

expression format 210, 355

GEQ 216

GRT 220

introduction 205

LEQ 224

LES 228

LIM 232

MEQ 238

NEQ 243

order of operation 210, 356

valid operators 209, 355

**COMPARE structure** 481, 489

**compute** 249

**compute instructions**

ABS 279

ADD 253

CPT 249

DIV 263

expression format 251, 347

introduction 247

MOD 268

MUL 260

NEG 276

order of operation 252, 348

SQR 272

SUB 257

valid operators 251, 347

**CONCAT instruction** 608

**configuring** 155

MSG instruction 155

PID instruction 505

**connection**

cache 170

**connector**

function block diagram 641

**construct**

structured text 671

**CONTROL structure** 338, 349, 369, 373,  
378, 388, 392, 397, 403, 408, 409,  
415, 422, 426, 430

**control structure** 450

**CONTROLLER object** 177

**CONTROLLERDEVICE object** 177

**conversion instructions**

- DEG 556
- FRD 565
- introduction 555
- RAD 559
- TOD 562
- TRN 567
- convert to BCD** 562
- convert to integer** 565
- converting data types** 635
- COP instruction** 358
- copy** 358
- COS instruction** 525
- cosine** 525
- count down** 127
- count up** 123
- count up/down** 131
- counter instructions**
  - CTD 127
  - CTU 123
  - CTUD 131
  - introduction 95
  - RES 136
- COUNTER structure** 123, 127
- CPS instruction** 358
- CPT instruction** 249
- CST object** 181
- CTD instruction** 127
- CTU instruction** 123
- CTUD instruction** 131

## D

- data transitional** 496
- DDT instruction**
  - operands 488
  - search mode 490
- deadband** 517
- DEG instruction** 556
- degree** 556
- DELETE instruction** 610
- description**
  - structured text 687
- DF1 object** 182
- diagnostic detect** 488
- digital alarm** 31
- DINT to String** 626
- DIV instruction** 263
- division** 263
- document**
  - structured text 687
- DTOS instruction** 626

- DTR instruction** 496

## E

- elements**
  - SIZE instruction 384
- end of transition instruction** 460
- EOT instruction** 460
- EQU instruction** 212
- equal to** 212
- error codes**
  - ASCII 574
  - MSG instruction 148
- EVENT instruction** 466
- event task**
  - configure 194
  - trigger via consumed tag 200
  - trigger via EVENT instruction 466
- examine if open** 72
- execution order** 644
- exponential** 552
- expression**
  - BOOL expression
    - structured text 663
  - numeric expression
    - structured text 663
  - order of execution
    - structured text 669
  - structured text
    - arithmetic operators 665
    - bitwise operators 669
    - functions 665
    - logical operators 668
    - overview 663
    - relational operators 666
- expressions**
  - format 210, 251, 347, 355
  - order of operation 210, 252, 348, 356
  - valid operators 209, 251, 347, 355

## F

- FAL instruction**
  - mode of operation 332
  - operands 337
- FAULTLOG object** 185
- FBC instruction**
  - operands 480
  - search mode 482
- FBD\_BIT\_FIELD\_DISTRIBUTE structure** 296
- FBD\_BOOLEAN\_AND structure** 319

**FBD\_BOOLEAN\_NOT structure** 328  
**FBD\_BOOLEAN\_OR structure** 322  
**FBD\_BOOLEAN\_XOR structure** 325  
**FBD\_COMPARE structure** 213, 217, 221, 225, 229, 244  
**FBD\_CONVERT structure** 562, 565  
**FBD\_COUNTER structure** 131  
**FBD\_LIMIT structure** 233  
**FBD\_LOGICAL structure** 306, 309, 312, 316  
**FBD\_MASK\_EQUAL structure** 239  
**FBD\_MASKED\_MOVE structure** 290  
**FBD\_MATH structure** 254, 258, 261, 264, 269, 277, 553  
**FBD\_MATH\_ADVANCED structure** 273, 280, 523, 526, 530, 533, 537, 541, 546, 550, 557, 560  
**FBD\_ONESHOT structure** 89, 92  
**FBD\_TIMER structure** 110, 114, 118  
**FBD\_TRUNCATE structure** 567  
**feedback loop**  
     function block diagram 645  
**feedforward** 518  
**FFL instruction** 396  
**FFU instruction** 402  
**FIFO load** 396  
**FIFO unload** 402  
**file arithmetic and logic** 337  
**file bit comparison** 480  
**file fill** 364  
**file instructions. See array instructions**  
**file search and compare** 349  
**FIND instruction** 612  
**Find String** 612  
**FLL instruction** 364  
**FOR instruction** 472  
**for/break instructions**  
     BRK 475  
     FOR 472  
     introduction 471  
     RET 476  
**FOR...DO** 678  
**FRD instruction** 565  
**FSC instruction**  
     mode of operation 332  
     operands 349  
**function block diagram**  
     choose elements 641  
     create a scan delay 648  
     resolve a loop 645  
     resolve data flow between blocks 647

**functions**  
     structured text 665

## G

**GEQ instruction** 216  
**get system value** 173  
**greater than** 220  
**greater than or equal to** 216  
**GRT instruction** 220  
**GSV instruction**  
     objects 176  
     operands 173

## I

**ICON** 641  
**IF...THEN** 672  
**immediate output instruction** 200  
**immediate values** 635  
**incremental mode** 335  
**inhibit**  
     task 194  
**input reference** 641  
**input wire connector** 641  
**input/output instructions**  
     GSV 173  
     introduction 139  
     IOT 200  
     MSG 140  
     SSV 173  
**INSERT instruction** 614  
**Insert String** 614  
**instructions**  
     advanced math 545  
     analog alarm 47  
     array  
         ASCII conversion 619  
         ASCII serial port 571  
         ASCII string manipulation 605  
     bit 69  
     compare 205  
     compute 247  
     conversion 555  
     counter 95  
     digital alarm 31  
     for/break 471  
     input/output 139  
     logical 283  
     math conversion 555  
     move 283  
     program control 435

- sequencer 421
- serial port 571
- shift 387
- special 479
- string conversion 619
- string manipulation 605
- timer 95
- trigonometric 521
- IOT instruction** 200
- IREF** 641

## J

- JMP instruction** 436
- JSR instruction** 438
- jump** 436
- jump to subroutine** 438
- JXR instruction**
  - control structure 450

## L

- label** 436
- latching data** 642
- LBL instruction** 436
- LEQ instruction** 224
- LES instruction** 228
- less than** 228
- less than or equal to** 224
- LFL instruction** 408
- LFU instruction** 414
- LIFO load** 408
- LIFO unload** 414
- LIM instruction** 232
- limit** 232
- LN instruction** 546
- log**
  - base 10 549
  - natural 546
- log base 10** 549
- LOG instruction** 549
- logical instructions**
  - AND 305
  - introduction 283
  - NOT 315
  - OR 308
  - XOR 311
- logical operators**
  - structured text 668
- lower case** 633
- LOWER instruction** 633

## M

- masked equal to** 238
- masked move** 287
- masked move with target** 290
- masks** 497
- master control reset** 454
- math conversion instructions**
  - DEG 556
  - FRD 565
  - introduction 555
  - RAD 559
  - TOD 562
  - TRN 567
- math operators**
  - structured text 665
- MCR instruction** 454
- MEQ instruction** 238
- message** 140
  - cach connections 170
  - programming guidelines 172
- MESSAGE object** 186
- MESSAGE structure** 140
- MID instruction** 616
- Middle String** 616
- mixing data types** 635
- MOD instruction** 268
- mode of operation** 332
- MODULE object** 188
- modulo division** 268
- MOTIONGROUP object** 189
- MOV instruction** 285
- move** 285
- move instructions**
  - BTD 293
  - BTDT 296
  - CLR 299
  - introduction 283
  - MOV 285
  - MVM 287
  - MVMT 290
- move/logical instructions**
  - BAND 319
  - BNOT 328
  - BOR 322
  - BXOR 325
- MSG instruction** 155
  - cache connection 170
  - communication method 169
  - error codes 148
  - operands 140
  - programming guidelines 172

structure 140  
**MUL instruction** 260  
**multiplication** 260  
**MVM instruction** 287  
**MVMT instruction** 290

## N

**natural log** 546  
**NEG instruction** 276  
**negate** 276  
**NEQ instruction** 243  
**no operation** 459  
**NOP instruction** 459  
**not equal to** 243  
**NOT instruction** 315  
**numeric expression** 663  
**numerical mode** 333

## O

### objects

CONTROLLER 177  
 CONTROLLERDEVICE 177  
 CST 181  
 DF1 182  
 FAULTLOG 185  
 GSV/SSV instruction 176  
 MESSAGE 186  
 MODULE 188  
 MOTIONGROUP 189  
 PROGRAM 190  
 ROUTINE 192  
 SERIALPORT 192  
 TASK 194  
 WALLCLOCKTIME 196

**OCON** 641  
**one shot** 80  
**one shot falling** 86  
**one shot falling with input** 92  
**one shot rising** 83  
**one shot rising with input** 89  
**ONS instruction** 80  
**operators** 209, 251, 347, 355  
   order of execution  
     structured text 669  
**OR instruction** 308  
**order of execution** 644  
   structured text expression 669  
**order of operation** 210, 252, 348, 356  
**OREF** 641  
**OSF instruction** 86

**OSFI instruction** 92  
**OSR instruction** 83  
**OSRI instruction** 89  
**OTE instruction** 74  
**OTL instruction** 76  
**OTU instruction** 78

### output

enable or disable end-of-task processing  
 194

update immediately 200

**output biasing** 518  
**output energize** 74  
**output latch** 76  
**output reference** 641  
**output unlatch** 78  
**output wire connector** 641  
**overflow conditions** 649  
**overlap**  
   check for task overlap 194

## P

**pause SFC instruction** 462

### PID instruction

alarms 507  
 configuring 505  
 deadband 517  
 feedforward 518  
 operands 499  
 output biasing 518  
 scaling 508  
 tuning 506

**PID structure** 501

### postscan

structured text 662

**product codes** 177

### program control instructions

AFI 458  
 EOT 460  
 EVENT 466  
 introduction 435  
 JMP 436  
 JSR 438  
 LBL 436  
 MCR 454  
 NOP 459  
 RET 438  
 SBR 438  
 TND 452  
 UID 456  
 UIE 456

**PROGRAM object** 190



**program/operator control**  
     overview 655  
**proportional, integral, and derivative** 499

## R

**RAD instruction** 559  
**radians** 559  
**REAL to String** 629  
**relational operators**  
     structured text 666  
**REPEAT...UNTIL** 684  
**RES instruction** 136  
**reset** 136  
**reset SFC instruction** 464  
**RESULT structure** 481, 489  
**RET instruction** 438, 476  
**retentive timer on** 105  
**retentive timer on with reset** 118  
**return** 438, 476  
**ROUTINE object** 192  
**RTO instruction** 105  
**RTOR instruction** 118  
**RTOS instruction** 629

## S

**SBR instruction** 438  
**scaling** 508  
**scan delay**  
     function block diagram 648  
**search mode** 482, 490  
**search string** 612  
**sequencer input** 422  
**sequencer instructions**  
     introduction 421  
     SQL 422  
     SQL 430  
     SQO 426  
**sequencer load** 430  
**sequencer output** 426  
**serial port instructions**  
     ABL 575  
     ACB 578  
     ACL 581  
     AHL 583  
     ARD 587  
     ARL 591  
     AWA 595  
     AWT 600  
     introduction 571

**SERIAL\_PORT\_CONTROL structure** 572,  
     574, 576, 578, 584, 588, 592, 596,  
     601

**SERIALPORT object** 192

**set system value** 173

**SFP instruction** 462

**SFR instruction** 464

**shift instructions**

BSL 388  
 BSR 392  
 FFL 396  
 FFU 402  
 introduction 387  
 LFL 408  
 LFU 414

**SIN instruction** 522

**sine** 522

**size in elements** 384

**SIZE instruction** 384

**sort** 373

**special instructions**

DDT 488  
 DTR 496  
 FBC 480  
 introduction 479  
 PID 499  
 SFP 462  
 SFR 464

**SQL instruction** 422

**SQL instruction** 430

**SQO instruction** 426

**SQR instruction** 272

**square root** 272

**SRT instruction** 373

**SSV instruction**

objects 176  
 operands 173

**standard deviation** 378

**status**

task 194

**STD instructions** 378

**STOD instruction** 622

**STOR instruction** 624

**string**

evaluation in structured text 667

**String Concatenate** 608

**string conversion instructions**

DTOS 626  
 introduction 619  
 LOWER 633  
 RTOS 629

- STOD 622
  - STOR 624
  - SWPB 301
  - UPPER 631
  - string data type** 573, 607, 621
  - String Delete** 610
  - string manipulation instructions**
    - CONCAT 608
    - DELETE 610
    - FIND 612
    - INSERT 614
    - introduction 605
    - MID 616
  - STRING structure** 573, 607, 621
  - String To DINT** 622
  - String To REAL** 624
  - structured text**
    - arithmetic operators 665
    - assign ASCII character 663
    - assignment 661
    - bitwise operators 669
    - CASE 675
    - comments 687
    - components 659
    - constructs 671
    - evaluation of strings 667
    - expression 663
    - FOR...DO 678
    - functions 665
    - IF...THEN 672
    - logical operators 668
    - non-retentive assignment 662
    - numeric expression 663
    - relational operators 666
    - REPEAT...UNTIL 684
    - WHILE...DO 681
  - structures**
    - COMPARE 481, 489
    - CONTROL 338, 349, 369, 373, 378, 388, 392, 397, 403, 408, 409, 415, 422, 426, 430
    - COUNTER 123, 127
    - FBD\_BIT\_FIELD\_DISTRIBUTE 296
    - FBD\_BOOLEAN\_AND 319
    - FBD\_BOOLEAN\_NOT 328
    - FBD\_BOOLEAN\_OR 322
    - FBD\_BOOLEAN\_XOR 325
    - FBD\_COMPARE 213, 217, 221, 225, 229, 244
    - FBD\_CONVERT 562, 565
    - FBD\_COUNTER 131
    - FBD\_LIMIT 233
    - FBD\_LOGICAL 306, 309, 312, 316
    - FBD\_MASK\_EQUAL 239
    - FBD\_MASKED\_MOVE 290
    - FBD\_MATH 254, 258, 261, 264, 269, 277, 553
    - FBD\_MATH\_ADVANCED 273, 280, 523, 526, 530, 533, 537, 541, 546, 550, 557, 560
    - FBD\_ONESHOT 89, 92
    - FBD\_TIMER 110, 114, 118
    - FBD\_TRUNCATE 567
    - MESSAGE 140
    - PID 501
    - RES instruction 136
    - RESULT 481, 489
    - SERIAL\_PORT\_CONTROL 572, 574, 576, 578, 584, 588, 592, 596, 601
    - STRING 573, 607, 621
    - string 573, 607, 621
    - TIMER 96, 100, 105
  - SUB instruction** 257
  - subroutine** 438
  - subtraction** 257
  - swap byte** 301
  - SWPB instruction** 301
  - synchronous copy** 358
- ## T
- TAN instruction** 529
  - tangent** 529
  - task**
    - configure programmatically 194
    - inhibit 194
    - monitor 194
    - trigger event task 466
    - trigger via consumed tag 200
  - TASK object** 194
  - temporary end** 452
  - timeout**
    - configure for event task 194
  - timer instructions**
    - introduction 95
    - RES 136
    - RTO 105
    - RTOR 118
    - TOF 100
    - TOFR 114
    - TON 96
    - TONR 110
  - timer off delay** 100
  - timer off delay with reset** 114
  - timer on delay** 96

**timer on delay with reset** 110  
**TIMER structure** 96, 100, 105  
**timing modes** 650  
**TND instruction** 452  
**TOD instruction** 562  
**TOF instruction** 100  
**TOFR instruction** 114  
**TON instruction** 96  
**TONR instruction** 110  
**trigger event task** 466  
**trigger event task instruction** 466  
**trigonometric instructions**  
    ACS 536  
    ASN 532  
    ATN 540  
    COS 525  
    introduction 521  
    SIN 522  
    TAN 529  
**TRN instruction** 567  
**truncate** 567  
**tuning** 506

## U

**UID instruction** 456  
**UIE instruction** 456  
**unresolved loop**  
    function block diagram 645  
**update output** 200  
**upper case** 631  
**UPPER instruction** 631  
**user interrupt disable** 456  
**user interrupt enable** 456

## W

**WALLCLOCKTIME object** 196  
**WHILE...DO** 681

## X

**X to the power of Y** 552  
**XIO instruction** 72  
**XOR instruction** 311  
**XPY instruction** 552

cuu duong than cong. com

cuu duong than cong. com

cuu duong than cong. com

cuu duong than cong. com



## How Are We Doing?

Your comments on our technical publications will help us serve you better in the future.  
Thank you for taking the time to provide us feedback.

You can complete this form and mail it back to us, visit us online at [www.ab.com/manuals](http://www.ab.com/manuals), or email us at [RADocumentComments@ra.rockwell.com](mailto:RADocumentComments@ra.rockwell.com)

Pub. Title/Type Logix5000™ Controllers General Instructions

Cat. No. \_\_\_\_\_ Pub. No. 1756-RM003I-EN-P Pub. Date January 2007 Part No. \_\_\_\_\_

Please complete the sections below. Where applicable, rank the feature (1=needs improvement, 2=satisfactory, and 3=outstanding).

<b>Overall Usefulness</b> 1 2 3	How can we make this publication more useful for you?		
<b>Completeness</b> (all necessary information is provided) 1 2 3	procedure/step	illustration	feature
	example	guideline	other
	explanation	definition	
<b>Technical Accuracy</b> (all provided information is correct) 1 2 3	Can we be more accurate?		
	text	illustration	
<b>Clarity</b> (all provided information is easy to understand) 1 2 3	How can we make things clearer?		
<b>Other Comments</b>	You can add additional comments on the back of this form.		

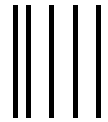
Your Name \_\_\_\_\_ Location/Phone \_\_\_\_\_  
Your Title/Function \_\_\_\_\_ Would you like us to contact you regarding your comments?  
\_\_\_\_ No, there is no need to contact me  
\_\_\_\_ Yes, please call me  
\_\_\_\_ Yes, please email me at \_\_\_\_\_  
\_\_\_\_ Yes, please contact me via \_\_\_\_\_

Return this form to: Allen-Bradley Marketing Communications, 1 Allen-Bradley Dr., Mayfield Hts., OH 44124-9705  
Phone: 440-646-3176 Fax: 440-646-3525 Email: [RADocumentComments@ra.rockwell.com](mailto:RADocumentComments@ra.rockwell.com)

PLEASE FASTEN HERE (DO NOT STAPLE)

Other Comments

PLEASE FOLD HERE



NO POSTAGE  
NECESSARY  
IF MAILED  
IN THE  
UNITED STATES

PLEASE REMOVE

**BUSINESS REPLY MAIL**

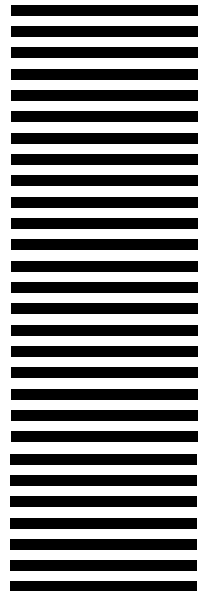
FIRST-CLASS MAIL PERMIT NO. 18235 CLEVELAND OH

POSTAGE WILL BE PAID BY THE ADDRESSEE



**Rockwell  
Automation**

1 ALLEN-BRADLEY DR  
MAYFIELD HEIGHTS OH 44124-9705



cuu duong than cong. com

cuu duong than cong. com

cuu duong than cong. com

cuu duong than cong. com



# Rockwell Automation Support

Rockwell Automation provides technical information on the web to assist you in using its products. At <http://support.rockwellautomation.com>, you can find technical manuals, a knowledge base of FAQs, technical and application notes, sample code and links to software service packs, and a MySupport feature that you can customize to make the best use of these tools.

For an additional level of technical phone support for installation, configuration and troubleshooting, we offer TechConnect Support programs. For more information, contact your local distributor or Rockwell Automation representative, or visit <http://support.rockwellautomation.com>.

## Installation Assistance

If you experience a problem with a hardware module within the first 24 hours of installation, please review the information that's contained in this manual. You can also contact a special Customer Support number for initial help in getting your module up and running:

United States	1.440.646.3223 Monday – Friday, 8am – 5pm EST
Outside United States	Please contact your local Rockwell Automation representative for any technical support issues.

## New Product Satisfaction Return

Rockwell tests all of 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:

United States	Contact your distributor. You must provide a Customer Support case number (see phone number above to obtain one) to your distributor in order to complete the return process.
Outside United States	Please contact your local Rockwell Automation representative for return procedure.

[www.rockwellautomation.com](http://www.rockwellautomation.com)

### Corporate Headquarters

Rockwell Automation, 777 East Wisconsin Avenue, Suite 1400, Milwaukee, WI, 53202-5302 USA, Tel: (1) 414.212.5200, Fax: (1) 414.212.5201

### Headquarters for Allen-Bradley Products, Rockwell Software Products and Global Manufacturing Solutions

Americas: Rockwell Automation, 1201 South Second Street, Milwaukee, WI 53204-2496 USA, Tel: (1) 414.382.2000, Fax: (1) 414.382.4444

Europe/Middle East/Africa: Rockwell Automation SA/NV, Vorstlaan/Boulevard du Souverain 36, 1170 Brussels, Belgium, Tel: (32) 2 663 0600, Fax: (32) 2 663 0640

Asia Pacific: Rockwell Automation, Level 14, Core F, Cyberport 3, 100 Cyberport Road, Hong Kong, Tel: (852) 2887 4788, Fax: (852) 2508 1846

### Headquarters for Dodge and Reliance Electric Products

Americas: Rockwell Automation, 6040 Ponders Court, Greenville, SC 29615-4617 USA, Tel: (1) 864.297.4800, Fax: (1) 864.281.2433

Europe/Middle East/Africa: Rockwell Automation, Herman-Heinrich-Gossen-Strasse 3, 50858 Köln, Germany, Tel: 49 (0) 2234 379410, Fax: 49 (0) 2234 3794164

Asia Pacific: Rockwell Automation, 55 Newton Road, #11-01/02 Revenue House, Singapore 307987, Tel: (65) 6356 9077, Fax: (65) 6356 9011

cuu duong than cong. com

cuu duong than cong. com



***Allen-Bradley***

***Logix5000™ Controllers General Instructions***

***Reference Manual***

cuu duong than cong. com

cuu duong than cong. com