# Logix5000 Controllers General Instructions

A L L E N - B R A D L E Y • R O C K W E L L S O F T W A R E

**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.

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

# Table of Contents

**Chapter 9**

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

**Chapter 10**

**Sequencer Instructions (SQI, SQO, SQL)**

**Chapter 11**

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

**Chapter 12**

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

**Chapter 13**

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

## 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 |
|---|---|
| Chapter 1 — Combined Digital Alarm (ALMD) and Analog Alarm (ALMA) instructions into one chapter. Added configuration, message string, and status information. | 29 |
| Chapter 4 — Added new GSV/SSV Controller Object attributes. | 180 |
| Chapter 20 — Added Debug Instructions (PPT, TPT). | 627 |

**Notes:**

## Where to Find an Instruction

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

| If the locator lists: | The instruction is documented in: |
| --- | --- |
| a page number | this manual |
| motion | *Logix5000 Controllers Motion Instruction Set Reference Manual,* publication 1756-RM007 |
| PhaseManager | *PhaseManager User Manual,* publication LOGIX-UM001 |
| process control | *Logix5000 Controllers Process Control and Drives Instruction Set Reference Manual*, publication 1756-RM006 |

| Instruction: | Location: | Languages: | Instruction: | Location: | Languages: |
|---|---|---|---|---|---|
| ABL<br>ASCII Test For Buffer Line | 609 | relay ladder<br>structured text | BNOT<br>Boolean NOT | 326 | structured text<br>function block |
| ABS<br>Absolute Value | 277 | relay ladder<br>structured text<br>function block | BOR<br>Boolean OR | 320 | structured text<br>function block |
| ACB<br>ASCII Chars in Buffer | 573 | relay ladder<br>structured text | BPT<br>Breakpoints | 627 | relay ladder |
| ACL<br>ASCII Clear Buffer | 575 | relay ladder<br>structured text | BRK<br>Break | 473 | relay ladder |
| ACS<br>Arc Cosine | 532 | relay ladder<br>structured text<br>function block | BSL<br>Bit Shift Left | 386 | relay ladder |
| ADD<br>Add | 252 | relay ladder<br>structured text<br>function block | BSR<br>Bit Shift Right | 390 | relay ladder |
| AFI<br>Always False Instruction | 456 | relay ladder | BTD<br>Bit Field Distribute | 294 | relay ladder |
| AHL<br>ASCII Handshake Lines | 577 | relay ladder<br>structured text | BTDT<br>Bit Field Distribute with Target | 294 | structured text<br>function block |
| ALM<br>Alarm | process control | structured text<br>function block | BTR<br>Message | 144 | relay ladder<br>structured text |
| ALMA<br>Analog Alarm | 42 | relay ladder<br>structured text<br>function block | BTW<br>Message | 144 | relay ladder<br>structured text |
| ALMD<br>Digital Alarm | 30 | relay ladder<br>structured text<br>function block | BXOR<br>Boolean Exclusive OR | 323 | structured text<br>function block |
| AND<br>Bitwise AND | 303 | relay ladder<br>structured text<br>function block | CC<br>Coordinated Control | process control | structured text<br>function block |
| ARD<br>ASCII Read | 581 | relay ladder<br>structured text | CLR<br>Clear | 294 | relay ladder<br>structured text |
| ARL<br>ASCII Read Line | 585 | relay ladder<br>structured text | CMP<br>Compare | 206 | relay ladder |
| ASN<br>Arc Sine | 529 | relay ladder<br>structured text<br>function block | CONCAT<br>String Concatenate | 601 | relay ladder<br>structured text |
| ATN<br>Arc Tangent | 535 | relay ladder<br>structured text<br>function block | COP<br>Copy File | 355 | relay ladder<br>structured text |
| AVE<br>File Average | 365 | relay ladder | COS<br>Cosine | 523 | relay ladder<br>structured text<br>function block |
| AWA<br>ASCII Write Append | 589 | relay ladder<br>structured text | CPS<br>Synchronous Copy File | 355 | relay ladder<br>structured text |
| AWT<br>ASCII Write | 594 | relay ladder<br>structured text | CPT<br>Compute | 248 | relay ladder |
| BAND<br>Boolean AND | 317 | structured text<br>function block | CTD<br>Count Down | 132 | relay ladder |
| | | | CTU<br>Count Up | 128 | relay ladder |

| Instruction: | Location: | Languages: |
|---|---|---|
| CTUD<br>Count Up/Down | 136 | structured text<br>function block |
| D2SD<br>Discrete 2-State Device | process control | structured text<br>function block |
| D3SD<br>Discrete 3-State Device | process control | structured text<br>function block |
| DDT<br>Diagnostic Detect | 486 | relay ladder |
| DEDT<br>Deadtime | process control | structured text<br>function block |
| DEG<br>Degrees | 553 | relay ladder<br>structured text<br>function block |
| DELETE<br>String Delete | 603 | relay ladder<br>structured text |
| DERV<br>Derivative | process control | structured text<br>function block |
| DFF<br>D Flip-Flop | process control | structured text<br>function block |
| DIV<br>Divide | 261 | relay ladder<br>structured text<br>function block |
| DTOS<br>DINT to String | 619 | relay ladder<br>structured text |
| DTR<br>Data Transitional | 494 | relay ladder |
| EOT<br>End of Transition | 458 | relay ladder<br>structured text |
| EQU<br>Equal to | 206 | relay ladder<br>structured text<br>function block |
| ESEL<br>Enhanced Select | process control | structured text<br>function block |
| EVENT<br>Trigger Event Task | 464 | relay ladder<br>structured text |
| FAL<br>File Arithmetic and Logic | 335 | relay ladder |
| FBC<br>File Bit Comparison | 478 | relay ladder |
| FFL<br>FIFO Load | 394 | relay ladder |
| FFU<br>FIFO Unload | 400 | relay ladder |
| FGEN<br>Function Generator | process control | structured text<br>function block |

| Instruction: | Location: | Languages: |
|---|---|---|
| FIND<br>Find String | 605 | relay ladder<br>structured text |
| FLL<br>File Fill | 361 | relay ladder |
| FOR<br>For | 470 | relay ladder |
| FRD<br>Convert to Integer | 559 | relay ladder<br>function block |
| FSC<br>File Search and Compare | 346 | relay ladder |
| GEQ<br>Greater than or Equal to | 215 | relay ladder<br>structured text<br>function block |
| GRT<br>Greater Than | 219 | relay ladder<br>structured text<br>function block |
| GSV<br>Get System Value | 176 | relay ladder<br>structured text |
| HLL<br>High/Low Limit | process control | structured text<br>function block |
| HPF<br>High Pass Filter | process control | structured text<br>function block |
| ICON<br>Input Wire Connector | 641 | function block |
| IMC<br>Internal Model Control | process control | structured text<br>function block |
| INSERT<br>Insert String | 607 | relay ladder<br>structured text |
| INTG<br>Integrator | process control | structured text<br>function block |
| IOT<br>Immediate Output | 201 | relay ladder<br>structured text |
| IREF<br>Input Reference | 641 | function block |
| JKFF<br>JK Flip-Flop | process control | structured text<br>function block |
| JMP<br>Jump to Label | 434 | relay ladder |
| JSR<br>Jump to Subroutine | 436 | relay ladder<br>structured text<br>function block |
| JXR<br>Jump to External Routine | 447 | relay ladder |
| LBL<br>Label | 434 | relay ladder |

| Instruction: | Location: | Languages: | Instruction: | Location: | Languages: |
|---|---|---|---|---|---|
| LDL2<br>Second-Order Lead Lag | process control | structured text<br>function block | MASD<br>Motion Axis Shutdown | motion | relay ladder<br>structured text |
| LDLG<br>Lead-Lag | process control | structured text<br>function block | MAS<br>Motion Axis Stop | motion | relay ladder<br>structured text |
| LEQ<br>Less Than or Equal to | 223 | relay ladder<br>structured text<br>function block | MASR<br>Motion Axis Shutdown Reset | motion | relay ladder<br>structured text |
| LES<br>Less Than | 227 | relay ladder<br>structured text<br>function block | MATC<br>Motion Axis Time Cam | motion | relay ladder<br>structured text |
| LFL<br>LIFO Load | 406 | relay ladder | MAVE<br>Moving Average | process control | structured text<br>function block |
| LFU<br>LIFO Unload | 412 | relay ladder | MAW<br>Motion Arm Watch | motion | relay ladder<br>structured text |
| LIM<br>Limit | 231 | relay ladder<br>function block | MAXC<br>Maximum Capture | process control | structured text<br>function block |
| LN<br>Natural Log | 540 | relay ladder<br>structured text<br>function block | MCCD<br>Motion Coordinated Change Dynamics | motion | relay ladder<br>structured text |
| LOG<br>Log Base 10 | (1) | relay ladder<br>structured text<br>function block | MCCM<br>Motion Coordinated Circular Move | motion | relay ladder<br>structured text |
| LOWER<br>Lower Case | 625 | relay ladder<br>structured text | MCCP<br>Motion Calculate Cam Profile | motion | relay ladder<br>structured text |
| LPF<br>Low Pass Filter | process control | structured text<br>function block | MCD<br>Motion Change Dynamics | motion | relay ladder<br>structured text |
| MAAT<br>Motion Apply Axis Tuning | motion | relay ladder<br>structured text | MCLM<br>Motion Coordinated Linear Move | motion | relay ladder<br>structured text |
| MAFR<br>Motion Axis Fault Reset | motion | relay ladder<br>structured text | MCR<br>Master Control Reset | 452 | relay ladder |
| MAG<br>Motion Axis Gear | motion | relay ladder<br>structured text | MCSD<br>Motion Coordinated Shutdown | motion | relay ladder<br>structured text |
| MAHD<br>Motion Apply Hookup Diagnostics | motion | relay ladder<br>structured text | MCS<br>Motion Coordinated Stop | motion | relay ladder<br>structured text |
| MAH<br>Motion Axis Home | motion | relay ladder<br>structured text | MCSR<br>Motion Coordinated Shutdown Reset | motion | relay ladder<br>structured text |
| MAJ<br>Motion Axis Jog | motion | relay ladder<br>structured text | MCT<br>Motion Coordinated Transform | motion | relay ladder<br>structured text |
| MAM<br>Motion Axis Move | motion | relay ladder<br>structured text | MCTP<br>Motion Calculate Transform Position | motion | relay ladder<br>structured text |
| MAOC<br>Motion Arm Output Cam | motion | relay ladder<br>structured text | MDF<br>Motion Direct Drive Off | motion | relay ladder<br>structured text |
| MAPC<br>Motion Axis Position Cam | motion | relay ladder<br>structured text | | | |
| MAR<br>Motion Arm Registration | motion | relay ladder<br>structured text | | | |

| Instruction: | Location: | Languages: | Instruction: | Location: | Languages: |
|---|---|---|---|---|---|
| MDOC<br>Motion Disarm Output Cam | motion | relay ladder<br>structured text | MSTD<br>Moving Standard Deviation | process control | structured text<br>function block |
| MDO<br>Motion Direct Drive On | motion | relay ladder<br>structured text | MUL<br>Multiply | 258 | relay ladder<br>structured text<br>function block |
| MDR<br>Motion Disarm Registration | motion | relay ladder<br>structured text | MUX<br>Multiplexer | process control | function block |
| MDW<br>Motion Disarm Watch | motion | relay ladder<br>structured text | MVM<br>Masked Move | 285 | relay ladder |
| MEQ<br>Mask Equal to | 237 | relay ladder<br>structured text<br>function block | MVMT<br>Masked Move with Target | 288 | structured text<br>function block |
| MGSD<br>Motion Group Shutdown | motion | relay ladder<br>structured text | NEG<br>Negate | 274 | relay ladder<br>structured text<br>function block |
| MGS<br>Motion Group Stop | motion | relay ladder<br>structured text | NEQ<br>Not Equal to | 242 | relay ladder<br>structured text<br>function block |
| MGSP<br>Motion Group Strobe<br>Position | motion | relay ladder<br>structured text | NOP<br>No Operation | 457 | relay ladder |
| MGSR<br>Motion Group Shutdown<br>Reset | motion | relay ladder<br>structured text | NOT<br>Bitwise NOT | 314 | relay ladder<br>structured text<br>function block |
| MID<br>Middle String | 609 | relay ladder<br>structured text | NTCH<br>Notch Filter | process control | structured text<br>function block |
| MINC<br>Minimum Capture | process control | structured text<br>function block | OCON<br>Output Wire Connector | 641 | function block |
| IMMC<br>Modular Multivariable<br>Control | process control | structured text<br>function block | ONS<br>One Shot | 88 | relay ladder |
| MOD<br>Modulo | 266 | relay ladder<br>structured text<br>function block | OR<br>Bitwise OR | 306 | relay ladder<br>structured text<br>function block |
| MOV<br>Move | 283 | relay ladder | OREF<br>Output Reference | 641 | function block |
| MRAT<br>Motion Run Axis Tuning | motion | relay ladder<br>structured text | OSFI<br>One Shot Falling with Input | 99 | structured text<br>function block |
| MRHD<br>Motion Run Hookup<br>Diagnostics | motion | relay ladder<br>structured text | OSF<br>One Shot Falling | 94 | relay ladder |
| MRP<br>Motion Redefine Position | motion | relay ladder<br>structured text | OSRI<br>One Shot Rising with Input | 91 | structured text<br>function block |
| MSF<br>Motion Servo Off | motion | relay ladder<br>structured text | OSR<br>One Shot Rising | 91 | relay ladder |
| MSG<br>Message | 144 | relay ladder<br>structured text | OTE<br>Output Energize | 82 | relay ladder |
| MSO<br>Motion Servo On | motion | relay ladder<br>structured text | OTL<br>Output Latch | 84 | relay ladder |
| | | | OTU<br>Output Unlatch | 86 | relay ladder |

| Instruction: | Location: | Languages: |
|---|---|---|
| PATT<br>Attach to Equipment Phase | PhaseManager | relay ladder<br>structured text |
| PCLF<br>Equipment Phase Clear Failure | PhaseManager | relay ladder<br>structured text |
| PCMD<br>Equipment Phase Command | PhaseManager | relay ladder<br>structured text |
| PDET<br>Detach from Equipment Phase | PhaseManager | relay ladder<br>structured text |
| PFL<br>Equipment Phase Failure | PhaseManager | relay ladder<br>structured text |
| PIDE<br>Enhanced PID | process control | structured text<br>function block |
| PID<br>Proportional Integral Derivative | 497 | relay ladder<br>structured text |
| PI<br>Proportional + Integral | process control | structured text<br>function block |
| PMUL<br>Pulse Multiplier | process control | structured text<br>function block |
| POSP<br>Position Proportional | process control | structured text<br>function block |
| POVR<br>Equipment Phase Override Command | PhaseManager | relay ladder<br>structured text |
| PPD<br>Equipment Phase Paused | PhaseManager | relay ladder<br>structured text |
| PRNP<br>Equipment Phase New Parameters | PhaseManager | relay ladder<br>structured text |
| PSC<br>Phase State Complete | PhaseManager | relay ladder<br>structured text |
| PXRQ<br>Equipment Phase External Request | PhaseManager | relay ladder<br>structured text |
| RAD<br>Radians | 553 | relay ladder<br>structured text<br>function block |
| RESD<br>Reset Dominant | process control | structured text<br>function block |
| RES<br>Reset | 141 | relay ladder |
| RET<br>Return | 436 and 474 | relay ladder<br>structured text<br>function block |

| Instruction: | Location: | Languages: |
|---|---|---|
| RLIM<br>Rate Limiter | process control | structured text<br>function block |
| RMPS<br>Ramp/Soak | process control | structured text<br>function block |
| RTO<br>Retentive Timer On | 112 | relay ladder |
| RTOR<br>Retentive Timer On with Reset | 124 | structured text<br>function block |
| RTOS<br>REAL to String | 621 | relay ladder<br>structured text |
| SBR<br>Subroutine | 436 | relay ladder<br>structured text<br>function block |
| SCL<br>Scale | process control | structured text<br>function block |
| SCRV<br>S-Curve | process control | structured text<br>function block |
| SEL<br>Select | process control | function block |
| SETD<br>Set Dominant | process control | structured text<br>function block |
| SFP<br>SFC Pause | 460 | relay ladder<br>structured text |
| SFR<br>SFC Reset | 462 | relay ladder<br>structured text |
| SIN<br>Sine | 520 | relay ladder<br>structured text<br>function block |
| SIZE<br>Size In Elements | 381 | relay ladder<br>structured text |
| SNEG<br>Selected Negate | process control | structured text<br>function block |
| SOC<br>Second-Order Controller | process control | structured text<br>function block |
| SQI<br>Sequencer Input | 420 | relay ladder |
| SQL<br>Sequencer Load | 428 | relay ladder |
| SQO<br>Sequencer Output | 424 | relay ladder |
| SQR<br>Square Root | 270 | relay ladder<br>function block |
| SQRT<br>Square Root | 270 | structured text |

| Instruction: | Location: | Languages: | Instruction: | Location: | Languages: |
|---|---|---|---|---|---|
| SRT<br>File Sort | 370 | relay ladder<br>structured text | UIE<br>User Interrupt Enable | 454 | relay ladder<br>structured text |
| SRTP<br>Split Range Time Proportional | process control | structured text<br>function block | UPDN<br>Up/Down Accumulator | process control | structured text<br>function block |
| SSUM<br>Selected Summer | process control | structured text<br>function block | UPPER<br>Upper Case | 623 | relay ladder<br>structured text |
| SSV<br>Set System Value | 176 | relay ladder<br>structured text | XIC<br>Examine If Closed | 78 | relay ladder |
| STD<br>File Standard Deviation | 375 | relay ladder | XIO<br>Examine If Open | 80 | relay ladder |
| STOD<br>String To DINT | 614 | relay ladder<br>structured text | XOR<br>Bitwise Exclusive OR | 310 | relay ladder<br>structured text<br>function block |
| STOR<br>String To REAL | 616 | relay ladder<br>structured text | XPY<br>X to the Power of Y | 546 | relay ladder<br>structured text<br>function block |
| SUB<br>Subtract | 255 | relay ladder<br>structured text<br>function block | | | |
| SWPB<br>Swap Byte | 299 | relay ladder<br>structured text | | | |
| TAN<br>Tangent | 526 | relay ladder<br>structured text<br>function block | | | |
| TND<br>Temporary End | 450 | relay ladder | | | |
| TOD<br>Convert to BCD | 556 | relay ladder<br>function block | | | |
| TOFR<br>Timer Off Delay with Reset | 120 | structured text<br>function block | | | |
| TOF<br>Timer Off Delay | 108 | relay ladder | | | |
| TONR<br>Timer On Delay with Reset | 116 | structured text<br>function block | | | |
| TON<br>Timer On Delay | 104 | relay ladder | | | |
| TOT<br>Totalizer | process control | structured text<br>function block | | | |
| TPT<br>Tracepoints | 631 | relay ladder | | | |
| TRN<br>Truncate | 561 | relay ladder<br>function block | | | |
| TRUNC<br>Truncate | 561 | structured text | | | |
| UID<br>User Interrupt Disable | 454 | relay ladder<br>structured text | | | |

**Notes:**

# Logix5000 Controllers General Instructions

## Introduction

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

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

## Who Should Use This Manual

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

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

# Purpose of This Manual

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

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

The following icons help identify language specific information:

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

## Common Information for All Instructions

The Logix5000 instruction set has some common attributes:

| For this information | See this appendix |
|---|---|
| common attributes | appendix Common Attributes defines:<br><br>• arithmetic status flags<br>• data types<br>• keywords |
| function block attributes | appendix Function Block Attributes defines:<br><br>• program and operator control<br>• timing modes |

## Conventions and Related Terms

### Set and clear

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

| This term | Means |
|---|---|
| set | the bit is set to 1 (ON)<br><br>a value is set to any non-zero number |
| clear | the bit is cleared to 0 (OFF)<br><br>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.
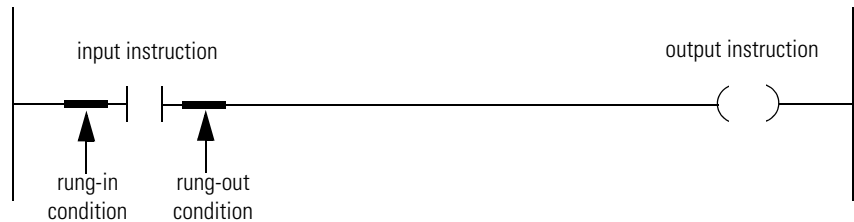


If the rung-in condition to an input instruction is true, the controller evaluates the instruction and sets the rung-out condition based on the results of the instruction. If the instruction evaluates to true, the rung-out condition is true; if the instruction evaluates to false, the rung-out condition 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 $+/-10^{+/-15}$ 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 $(+/-10^{+/-38})$.

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

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

Every function block instruction also includes EnableIn and EnableOut parameters:

- function block instructions execute normally when EnableIn is set.

- when EnableIn is cleared, the function block instruction either executes prescan logic, postscan logic, or just skips normal algorithm execution.

- EnableOut mirrors EnableIn, however, if function block execution detects an overflow condition EnableOut is also cleared.

- function block execution resumes where it left off when EnableIn toggles from cleared to set. However there are some function block instructions that specify special functionality, such as re-initialization, when EnableIn toggles from cleared to set. For function block instructions with time base parameters, whenever the timing mode is Oversample, the instruction always resumes were it left off when EnableIn toggles from cleared to set.

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

**Notes:**

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

## Introduction

These Logix-based alarm instructions are available in relay ladder, structured text, and function block diagram. When used with FactoryTalk View SE software, version 5.0 and later, these instructions create an alarming system with your visualization package. The controller detects alarm conditions and publishes events to FactoryTalk View Alarms and Events servers that propagate alarms to Factory Talk View SE clients that subscribe to receive notifications.

| If You Want To | Use This Instruction | Available In These Languages | See Page |
|---|:---:|:---:|:---:|
| detect alarms based on Boolean (true/false) conditions | ALMD | relay ladder | 30 |
| | | structured text | |
| | | function block | |
| detect alarms based on the level or rate of change of a value | ALMA | relay ladder | 42 |
| | | structured text | |
| | | function block | |

# Digital Alarm (ALMD)

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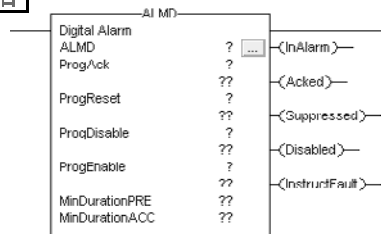
## Operands:

### Relay Ladder



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

| Operand | Type | Format | Description |
|---|---|---|---|
| ALMD tag | ALARM_DIGITAL | Structure | ALMD structure. |
| In | BOOL | Tag Immediate | **Structured text only.** Value is copied to In when instruction executes. The alarm input value is compared to determine whether there is an alarm. |
| ProgAck | BOOL | Tag Immediate | Value is copied to ProgAck when instruction executes. On transition from cleared to set, acknowledges alarm (if acknowledgement is required). |
| ProgReset | BOOL | Tag Immediate | Value is copied to ProgReset when instruction executes. On transition from cleared to set, resets alarm (if required). |
| ProgDisable | BOOL | Tag Immediate | Value is copied to ProgDisable when instruction executes. When set, disables alarm (does not override Enable Commands). |
| ProgEnable | BOOL | Tag Immediate | Value is copied to ProgEnable when instruction executes. When set, enables alarm (takes precedence over Disable Commands). |
| MinDurationPRE | DINT | Immediate | **Relay ladder only.** Specifies how long the alarm condition must be met before it is reported (milliseconds). |
| MinDurationACC | DINT | Immediate | **Relay ladder only.** Indicates the number of milliseconds that have elapsed since the alarm condition was met. |

### Structured Text



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

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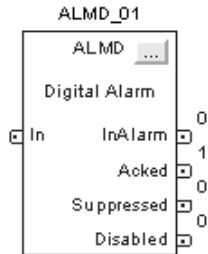
## Function Block



| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| ALMD tag | ALARM_DIGITAL | Structure | ALMD structure |

## ALARM_DIGITAL Structure

| Input Parameter | Data Type | Description |
|-----------------|-----------|-------------|
| EnableIn | BOOL | **Relay Ladder:**<br><br>Corresponds to the rung state. Does not affect processing.<br><br>**Function Block:**<br><br>If cleared, the instruction does not execute and outputs are not updated.<br><br>If set, the instruction executes.<br><br>Default is set.<br><br>**Structured Text:**<br><br>No effect. The instruction always executes. |
| In | BOOL | The digital signal input to the instruction.<br><br>Default is cleared.<br><br>**Relay Ladder:**<br><br>Follows the rung condition. Set if the rung condition is true. Cleared if the rung condition is false.<br><br>**Structured Text:**<br><br>Copied from instruction operand. |
| InFault | BOOL | Bad health indicator for the input. The user application may set InFault to indicate the input signal has an error. When set, the instruction sets InFaulted (Status.1). When cleared, the instruction clears InFaulted (Status.1). In either case, the instruction continues to evaluate In for alarm conditions.<br><br>Default is cleared (good health). |
| Condition | BOOL | Specifies how alarm is activated. When Condition is set, the alarm condition is activated when In is set. When Condition is cleared, the alarm condition is activated when In is cleared.<br><br>Default is set. |

| Input Parameter | Data Type | Description |
|---|---|---|
| AckRequired | BOOL | Specifies whether alarm acknowledgement is required. When set, acknowledgement is required. When cleared, acknowledgement is not required and Acked is always set.<br><br>Default is set. |
| Latched | BOOL | Specifies whether the alarm is latched. Latched alarms remain InAlarm when the alarm condition becomes false, until a Reset command is received. When set, the alarm is latched. When cleared, the alarm is unlatched.<br><br>A latched alarm can only be reset when the alarm condition is false.<br><br>Default is cleared. |
| ProgAck | BOOL | Set by the user program to acknowledge the alarm. Requires a cleared-to-set transition while the alarm is unacknowledged.<br><br>Default is cleared.<br><br>**Relay Ladder:**<br><br>Copied from the instruction operand.<br><br>**Structured Text:**<br><br>Copied from the instruction operand. |
| OperAck | BOOL | Set by the operator interface to acknowledge the alarm. Requires a cleared-to-set transition while the alarm is unacknowledged. The instruction clears this parameter.<br><br>Default is cleared. |
| ProgReset | BOOL | Set by the user program to reset the alarm. Requires a cleared-to-set transition while the alarm is InAlarm and the In condition is not in alarm.<br><br>Default is cleared.<br><br>**Relay Ladder:**<br><br>Copied from the instruction operand.<br><br>**Structured Text:**<br><br>Copied from the instruction operand. |
| OperReset | BOOL | Set by the operator interface to reset the alarm. Requires a cleared-to-set transition while the alarm is InAlarm and the In condition is not in alarm. The alarm instruction clears this parameter.<br><br>Default is cleared. |
| ProgSuppress | BOOL | Set by the user program to suppress the alarm.<br><br>Default is cleared. |
| OperSuppress | BOOL | Set by the operator interface to suppress the alarm. The alarm instruction clears this parameter.<br><br>Default is cleared. |

| Input Parameter | Data Type | Description |
|---|---|---|
| ProgUnsuppress | BOOL | Set by the user program to unsuppress the alarm. Takes precedence over Suppress commands.<br><br>Default is cleared. |
| OperUnsuppress | BOOL | Set by the operator interface to unsuppress the alarm. Takes precedence over Suppress commands. The alarm instruction clears this parameter.<br><br>Default is cleared. |
| ProgDisable | BOOL | Set by the user program to disable the alarm.<br><br>Default is cleared.<br><br>**Relay Ladder:**<br><br>Copied from the instruction operand.<br><br>**Structured Text:**<br><br>Copied from the instruction operand. |
| OperDisable | BOOL | Set by the operator interface to disable the alarm. The alarm instruction clears this parameter.<br><br>Default is cleared. |
| ProgEnable | BOOL | Set by the user program to enable the alarm. Takes precedence over a Disable command.<br><br>Default is cleared.<br><br>**Relay Ladder:**<br><br>Copied from the instruction operand.<br><br>**Structured Text:**<br><br>Copied from the instruction operand. |
| OperEnable | BOOL | Set by the operator interface to enable the alarm. Takes precedence over Disable command. The alarm instruction clears this parameter.<br><br>Default is cleared. |
| AlarmCountReset | BOOL | Set by the user program to reset the alarm count. A cleared-to-set transition resets the alarm count to zero.<br><br>Default is cleared. |
| UseProgTime | BOOL | Specifies whether to use the controller's clock or the ProgTime value to timestamp alarm state change events. When set, the ProgTime value provides timestamp. When cleared, the controller's clock provides timestamp.<br><br>Default is cleared. |

| Input Parameter | Data Type | Description |
|---|---|---|
| ProgTime | LINT | If UseProgTime is set, this value is used to provide the timestamp value for all events. This lets the application apply timestamps obtained from the alarm source, such as a sequence-of-events input module. |
| Severity | DINT | Severity of the alarm. This does not affect processing of alarms by the controller, but can be used for sorting and filtering functions at the alarm subscriber.<br><br>Valid = 1...1000 (1000 = most severe; 1 = least severe).<br><br>Default = 500. |
| MinDurationPRE | DINT | Minimum duration preset (milliseconds) for the alarm condition to remain true before the alarm is marked as InAlarm and alarm notification is sent to clients. The controller collects alarm data as soon as the alarm condition is detected, so no data is lost while waiting to meet the minimum duration.<br><br>Valid = 0...2,147,483,647.<br><br>Default = 0. |

| Output Parameter | Data Type | Description |
|---|---|---|
| EnableOut | BOOL | Enable output. |
| InAlarm | BOOL | Alarm active status. Set when the alarm is active. Cleared when the alarm is not active (normal status). |
| Acked | BOOL | Alarm acknowledged status. Set when the alarm is acknowledged. Cleared when the alarm is not acknowledged.<br><br>Acked is always set when AckRequired is cleared. |
| InAlarmUnack | BOOL | Combined alarm active and acknowledged status. Set when the alarm is active (InAlarm is set) and unacknowledged (Acked is cleared). Cleared when the alarm is normal (inactive), acknowledged, or both. |
| Suppressed | BOOL | Suppressed status of the alarm. Set when the alarm is suppressed. Cleared when the alarm is not suppressed. |
| Disabled | BOOL | Disabled status of the alarm. Set when the alarm is disabled. Cleared when the alarm is enabled. |
| MinDurationACC | DINT | Elapsed time since the alarm was detected. When this value reaches MinDurationPRE, the alarm becomes active (InAlarm is set), and a notification is sent to clients. |
| AlarmCount | DINT | Number of times the alarm has been activated (InAlarm is set). If the maximum value is reached, the counter leaves the value at the maximum count value. |
| InAlarmTime | LINT | Timestamp of alarm detection. |
| AckTime | LINT | Timestamp of alarm acknowledgement. If the alarm does not require acknowledgement, this timestamp is equal to alarm time. |
| RetToNormalTime | LINT | Timestamp of alarm returning to a normal state. |
| AlarmCountResetTime | LINT | Timestamp indicating when the alarm count was reset. |
| DeliveryER | BOOL | Delivery error of the alarm notification message. Set when there is a delivery error: either no alarm subscriber was subscribed or at least one subscriber did not receive the latest alarm change state message. Cleared when delivery is successful or is in progress. |

| Output Parameter | Data Type | Description |
|---|---|---|
| DeliveryDN | BOOL | Delivery completion of the alarm notification message. Set when delivery is successful: at least one subscriber was subscribed and all subscribers received the latest alarm change state message successfully. Cleared when delivery does not complete successfully or is in progress. |
| DeliveryEN | BOOL | Delivery status of the alarm notification message. Set when delivery is in progress. Cleared when delivery is not in progress. |
| NoSubscriber | BOOL | Alarm had no subscribers when attempting to deliver the most recent message. Set when there are no subscribers. Cleared when there is at least one subscriber. |
| NoConnection | BOOL | Alarm's subscribers were not connected when attempting to deliver the most recent message. Set when all subscribers are disconnected. Cleared when at least one subscriber is connected or there are no subscribers. |
| CommError | BOOL | Communication error when delivering an alarm message. Set when there are communication errors and all retries are used. This means that a subscriber was subscribed and it had a connection, but the controller did not receive confirmation of message delivery. Cleared when all connected subscribers confirm receipt of the alarm message. |
| AlarmBuffered | BOOL | Alarm message buffered due to a communication error (CommError is set) or a lost connection (NoConnection is set). Set when the alarm message is buffered for at least one subscriber. Cleared when the alarm message is not buffered. |
| Subscribers | DINT | Number of subscribers for this alarm. |
| SubscNotified | DINT | Number of subscribers successfully notified about the most recent alarm state change. |
| Status | DINT | Combined status indicators: <br><br> Status.0 = InstructFault. <br><br> Status.1= InFaulted. <br><br> Status.2 = SeverityInv. |
| InstructFault (Status.0) | BOOL | Instruction error conditions exist. This is not a minor or major controller error. Check the remaining status bits to determine what occurred. |
| InFaulted (Status.1) | BOOL | User program has set InFault to indicate bad quality input data. Alarm continues to evaluate In for alarm condition. |
| SeverityInv (Status.2) | BOOL | Alarm severity configuration is invalid. <br><br> If severity <1, the instruction uses Severity = 1. <br><br> If severity >1000, the instruction uses Severity = 1000. |

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

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

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

## State Diagrams when Acknowledgement Required

**Latched = False**
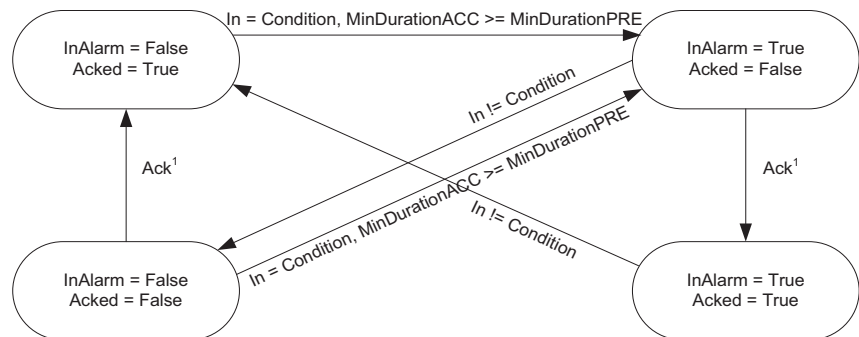


**Latched = True**
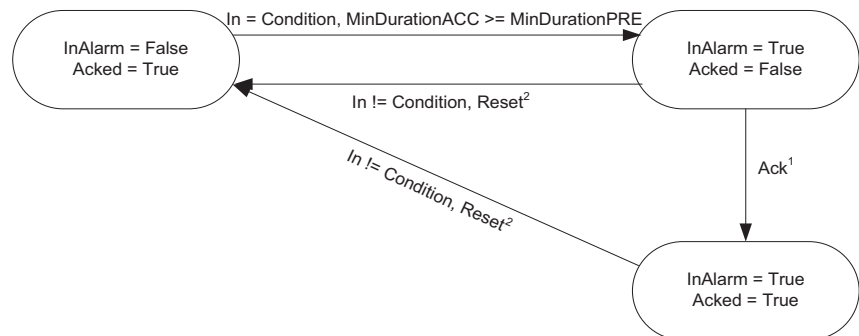


[1] Alarm can be acked by several different ways: ProgAck, OperAck, clients (RSLogix 5000 software, RSView software).

[2] Alarm can be reset by several different ways: ProgReset, OperReset, clients (RSLogix 5000 software, RSView software).

## State Diagrams when Acknowledgment Not Required

**Latched = False**



**Latched = True**
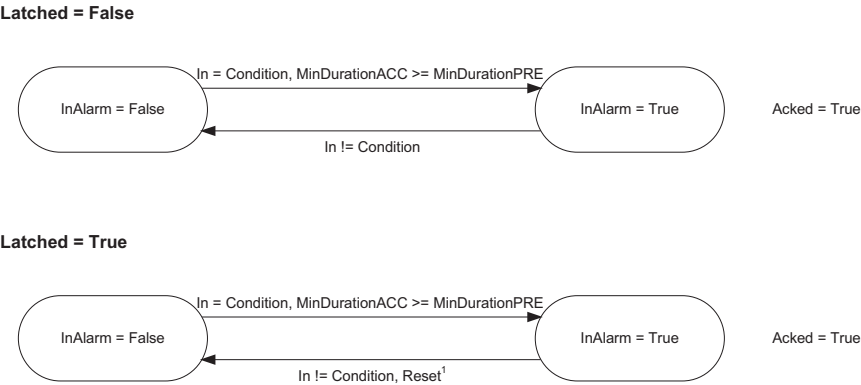


[1] Alarm can be reset by several different ways: ProgReset, OperReset, clients (RSLogix 5000 software, RSView software)

**Arithmetic Status Flags:**  none

**Fault Conditions:**  none

**Execution:**

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

| Condition | Function Block Action | Structured Text Action |
|---|---|---|
| prescan | All operator requests, timestamps, and delivery flags are cleared.<br><br>InAlarm is cleared and Acked is set. | All operator requests, timestamps, and delivery flags are cleared.<br><br>InAlarm is cleared and Acked is set. |
| instruction first scan | No action taken. | No action taken. |
| instruction first run | No action taken. | No action taken. |
| EnableIn is cleared | The instruction does not execute.<br><br>EnableOut is cleared. | The instruction executes.<br><br>EnableOut is always set. |
| EnableIn is set | The instruction executes.<br><br>EnableOut is set. | The instruction executes.<br><br>EnableOut is always set. |
| postscan | No action taken. | No action taken. |

## ALMD Alarm Acknowledge Required and Latched
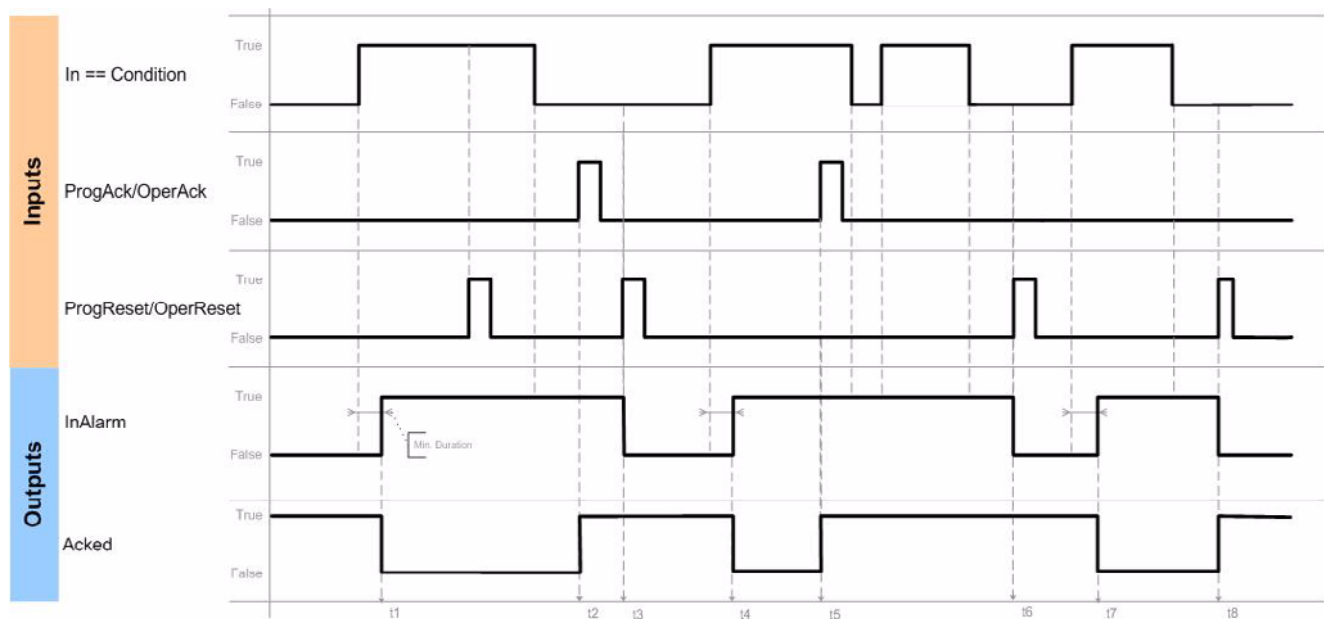
# ALMD Alarm Acknowledge Required and Not Latched



# ALMD Alarm Acknowledge Not Required and Latched



Under this configuration, this signal does not have any influence on alarm state change.
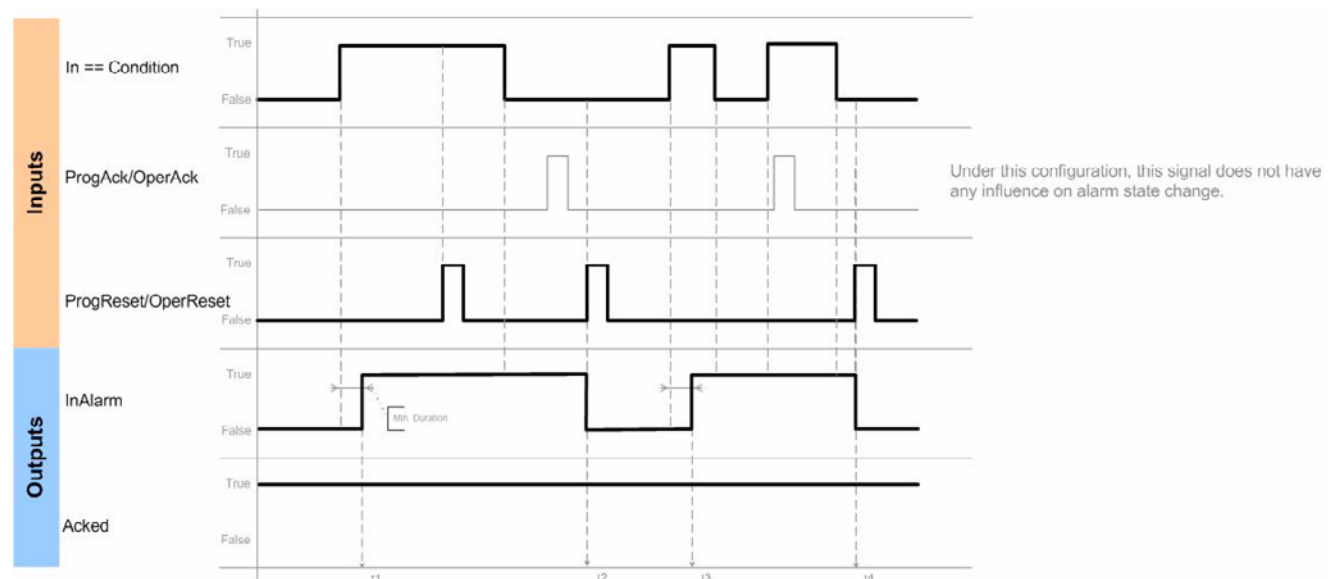
# ALMD Alarm Acknowledge Not Required and Not Latched
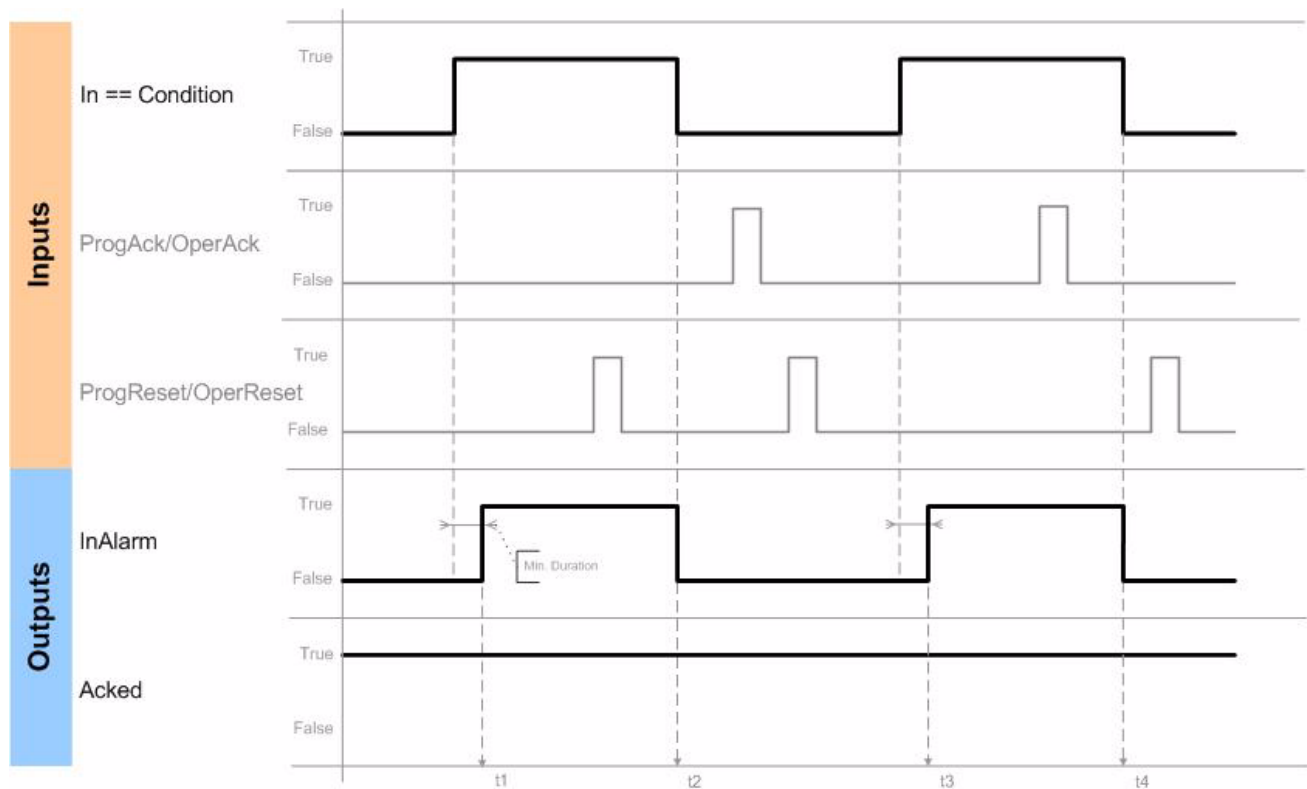


**Example:**   Two motor failure signals are combined such that if either one occurs, a motor fault alarm is activated. Programmatically acknowledge the alarm with a cleared-to-set transition of the Motor101Ack tag value. The application logic must clear Motor101Ack.

### Relay Ladder



Publication 1756-RM003K-EN-P - July 2008
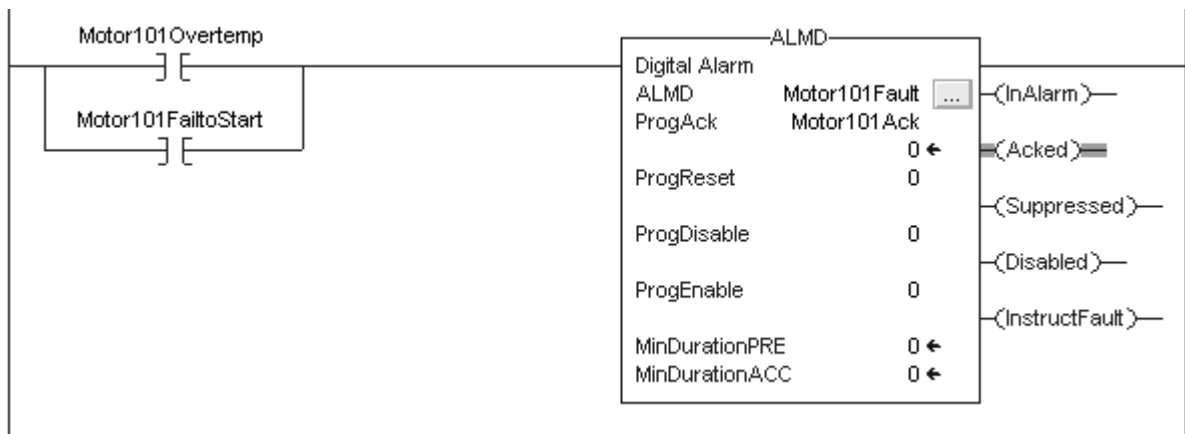
### Structured Text

```
Motor101FaultConditions := Motor101Overtemp OR
Motor101FailToStart;

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

### Function Block
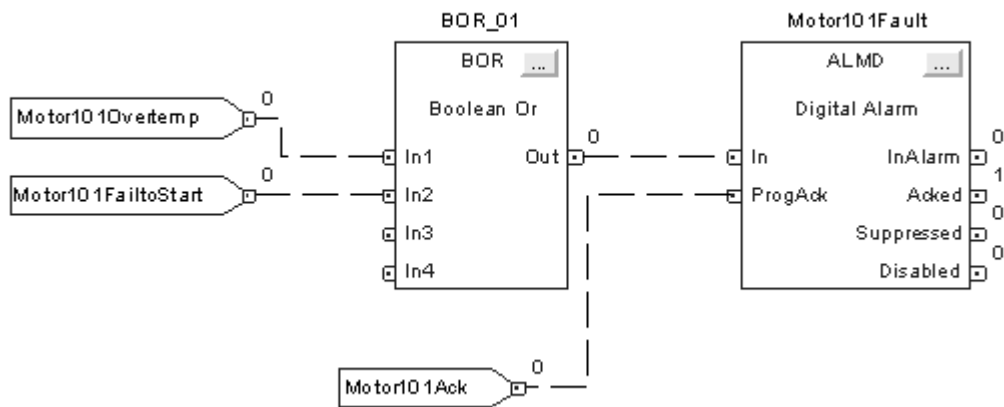
# Analog Alarm (ALMA)

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

## Operands:

### Relay Ladder



| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| ALMA tag | ALARM_ANALOG | Structure | ALMA structure. |
| In | REAL<br>DINT<br>INT<br>SINT | Tag<br>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<br>Immediate | Value is copied to ProgAckAll when instruction executes. On transition from cleared to set, acknowledges all alarm conditions that require acknowledgement. |
| ProgDisable | BOOL | Tag<br>Immediate | Value is copied to ProgDisable when instruction executes. When set, disables alarm (does not override Enable Commands). |
| ProgEnable | BOOL | Tag<br>Immediate | Value is copied to ProgEnable when instruction executes. When set, enables alarm (takes precedence over Disable commands). |
| HHlimit | REAL | Immediate | **Relay ladder only.**<br><br>High High alarm limit. |
| HLimit | REAL | Immediate | **Relay ladder only.**<br><br>High alarm limit. |
| LLimit | REAL | Immediate | **Relay ladder only.**<br><br>Low alarm limit. |
| LLLimit | REAL | Immediate | **Relay ladder only.**<br><br>Low Low alarm limit. |

## Structured Text

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

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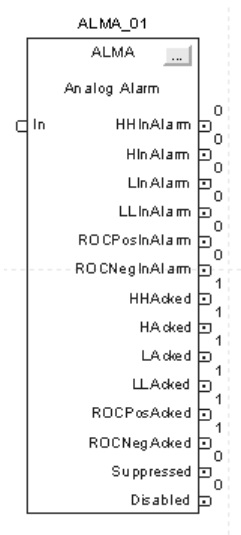
## Function Block



| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| ALMA tag | ALARM_ANALOG | Structure | ALMA structure |

## ALARM_ANALOG Structure

| Input Parameter | Data Type | Description |
|---|---|---|
| EnableIn | BOOL | **Relay Ladder:**<br><br>Corresponds to the rung state. If cleared, the instruction does not execute and outputs are not updated.<br><br>**Structured Text:**<br><br>No effect. The instruction always executes.<br><br>**Function Block:**<br><br>If cleared, the instruction does not execute and outputs are not updated.<br><br>Default is set. |
| In | REAL | The alarm input value, which is compared with alarm limits to detect alarm conditions.<br><br>Default = 0.0.<br><br>**Relay Ladder:**<br><br>Copied from the instruction operand.<br><br>**Structured Text:**<br><br>Copied from instruction operand. |
| InFault | BOOL | Bad health indicator for the input. The user application may set InFault to indicate the input signal has an error. When set, the instruction sets InFaulted (Status.1). When cleared, the instruction clears InFaulted (Status.1). In either case, the instruction continues to evaluate In for alarm conditions.<br><br>Default is cleared (good health). |
| HHEnabled | BOOL | High High alarm condition detection. Set to enable detection of the High High alarm condition. Clear to disable detection of the High High alarm condition.<br><br>Default is set. |
| HEnabled | BOOL | High alarm condition detection. Set to enable detection of the High alarm condition. Clear to disable detection of the High alarm condition.<br><br>Default is set. |
| LEnabled | BOOL | Low alarm condition detection. Set to enable detection of the Low alarm condition. Clear to disable detection of the Low alarm condition.<br><br>Default is set. |
| LLEnabled | BOOL | Low Low alarm condition detection. Set to enable detection of the Low Low alarm condition. Clear to disable detection of the Low Low alarm condition.<br><br>Default is set. |

| Input Parameter | Data Type | Description |
|---|---|---|
| AckRequired | BOOL | Specifies whether alarm acknowledgement is required. When set, acknowledgement is required. When cleared, acknowledgement is not required and HHAcked, HAcked, LAcked, LLAcked, ROCPosAcked, and ROCNegAcked are always set.<br><br>Default is set. |
| ProgAckAll | BOOL | Set by the user program to acknowledge all conditions of this alarm. Requires a cleared-to-set transition while the alarm conditions are unacknowledged.<br><br>Default is cleared.<br><br>**Relay Ladder:**<br><br>Copied from the instruction operand.<br><br>**Structured Text:**<br><br>Copied from the instruction operand. |
| OperAckAll | BOOL | Set by the operator interface to acknowledge all conditions of this alarm. Requires a cleared-to-set transition while the alarm conditions are unacknowledged. The alarm instruction clears this parameter.<br><br>Default is cleared. |
| HHProgAck | BOOL | High High program acknowledge. Set by the user program to acknowledge a High High condition. Requires a cleared-to-set transition while the alarm condition is unacknowledged.<br><br>Default is cleared. |
| HHOperAck | BOOL | High High operator acknowledge. Set by the operator interface to acknowledge a High High condition. Requires a cleared-to-set transition while the alarm condition is unacknowledged. The alarm instruction clears this parameter.<br><br>Default is cleared. |
| HProgAck | BOOL | High program acknowledge. Set by the user program to acknowledge a High condition. Requires a cleared-to-set transition while the alarm condition is unacknowledged.<br><br>Default is cleared. |
| HOperAck | BOOL | High operator acknowledge. Set by the operator interface to acknowledge a High condition. Requires a cleared-to-set transition while the alarm condition is Unacknowledged. The alarm instruction clears this parameter.<br><br>Default is cleared. |
| LProgAck | BOOL | Low program acknowledge. Set by the user program to acknowledge a Low condition. Requires a cleared-to-set transition while the alarm condition is unacknowledged.<br><br>Default is cleared. |
| LOperAck | BOOL | Low operator acknowledge. Set by the operator interface to acknowledge a Low condition. Requires a cleared-to-set transition while the alarm condition is unacknowledged. The alarm instruction clears this parameter.<br><br>Default is cleared. |

| Input Parameter | Data Type | Description |
|---|---|---|
| LLProgAck | BOOL | Low Low program acknowledge. Set by the user program to acknowledge a Low Low condition. Requires a cleared-to-set transition while the alarm condition is unacknowledged.<br><br>Default is cleared. |
| LLOperAck | BOOL | Low Low operator acknowledge. Set by the operator interface to acknowledge a Low Low condition. Requires a cleared-to-set transition while the alarm condition is unacknowledged. The alarm instruction clears this parameter.<br><br>Default is cleared. |
| ROCPosProgAck | BOOL | Positive rate of change program acknowledge. Set by the user program to acknowledge a positive rate-of-change condition. Requires a cleared-to-set transition while the alarm condition is unacknowledged.<br><br>Default is cleared. |
| ROCPosOperAck | BOOL | Positive rate of change operator acknowledge. Set by the operator interface to acknowledge a positive rate-of-change condition. Requires a cleared-to-set transition while the alarm condition is unacknowledged. The alarm instruction clears this parameter.<br><br>Default is cleared. |
| ROCNegProgAck | BOOL | Negative rate of change program acknowledge. Set by the user program to acknowledge a negative rate-of-change condition. Requires a cleared-to-set transition while the alarm condition is unacknowledged.<br><br>Default is cleared. |
| ROCNegOperAck | BOOL | Negative rate of change operator acknowledge. Set by the operator interface to acknowledge a negative rate-of-change condition. Requires a cleared-to-set transition while the alarm condition is unacknowledged. The alarm instruction clears this parameter.<br><br>Default is cleared. |
| ProgSuppress | BOOL | Set by the user program to suppress the alarm.<br><br>Default is cleared. |
| OperSuppress | BOOL | Set by the operator interface to suppress the alarm. The alarm instruction clears this parameter.<br><br>Default is cleared. |
| ProgUnsuppress | BOOL | Set by the user program to unsuppress the alarm. Takes precedence over Suppress commands.<br><br>Default is cleared. |
| OperUnsuppress | BOOL | Set by the operator interface to unsuppress the alarm. Takes precedence over Suppress commands. The alarm instruction clears this parameter.<br><br>Default is cleared. |

| Input Parameter | Data Type | Description |
|---|---|---|
| ProgDisable | BOOL | Set by the user program to disable the alarm.<br><br>Default is cleared.<br><br>**Relay Ladder:**<br><br>Copied from the instruction operand.<br><br>**Structured Text:**<br><br>Copied from the instruction operand. |
| OperDisable | BOOL | Set by the operator interface to disable the alarm. The alarm instruction clears this parameter.<br><br>Default is cleared. |
| ProgEnable | BOOL | Set by the user program to enable the alarm. Takes precedence over a Disable command.<br><br>Default is cleared.<br><br>**Relay Ladder:**<br><br>Copied from the instruction operand.<br><br>**Structured Text:**<br><br>Copied from the instruction operand. |
| OperEnable | BOOL | Set by the operator interface to enable the alarm. Takes precedence over Disable command. The alarm instruction clears this parameter.<br><br>Default is cleared. |
| AlarmCountReset | BOOL | Set by the user program to reset the alarm counts for all conditions. A cleared-to-set transition resets the alarm counts to zero.<br><br>Default is cleared. |
| HHLimit | REAL | High High alarm limit.<br><br>Valid = HLimit < HHLimit < maximum positive float.<br><br>Default = 0.0. |
| HHSeverity | DINT | Severity of the High High alarm condition. This does not affect processing of alarms by the controller, but can be used for sorting and filtering functions at the alarm subscriber.<br><br>Valid = 1...1000 (1000 = most severe; 1 = least severe).<br><br>Default = 500. |
| HLimit | REAL | High alarm limit.<br><br>Valid = LLimit < HLimit < HHLimit.<br><br>Default = 0.0. |

| Input Parameter | Data Type | Description |
|---|---|---|
| HSeverity | DINT | Severity of the High alarm condition. This does not affect processing of alarms by the controller, but can be used for sorting and filtering functions at the alarm subscriber.<br><br>Valid = 1...1000 (1000 = most severe; 1 = least severe).<br><br>Default = 500. |
| LLimit | REAL | Low alarm limit.<br><br>Valid = LLLimit < LLimit < HLimit.<br><br>Default = 0.0. |
| LSeverity | DINT | Severity of the Low alarm condition. This does not affect processing of alarms by the controller, but can be used for sorting and filtering functions at the alarm subscriber.<br><br>Valid = 1...1000 (1000 = most severe; 1 = least severe).<br><br>Default = 500. |
| LLLimit | REAL | Low Low alarm limit.<br><br>Valid = maximum negative float < LLLimit < LLimit.<br><br>Default = 0.0. |
| LLSeverity | DINT | Severity of the Low Low alarm condition. This does not affect processing of alarms by the controller, but can be used for sorting and filtering functions at the alarm subscriber.<br><br>Valid = 1...1000 (1000 = most severe; 1 = least severe).<br><br>Default = 500. |
| MinDurationPRE | DINT | Minimum duration preset (milliseconds) for an alarm level condition to remain true before the condition is marked as InAlarm and alarm notification is sent to clients. The controller collects alarm data as soon as the alarm condition is detected, so no data is lost while waiting to meet the minimum duration. Does not apply to rate-of-change conditions.<br><br>MinDurationPRE only applies to the first excursion from normal in either direction. For example, once the High condition times out, the High High condition will become active immediately, while a low condition will wait for the timeout period.<br><br>Valid = 0...2,147,483,647.<br><br>Default = 0. |

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

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

| Output Parameter | Data Type | Description |
|---|---|---|
| LInAlarmUnack | BOOL | Combined Low condition active and unacknowledged status. Set when the Low condition is active (LInAlarm is set) and unacknowledged. Cleared when the Low condition is normal (inactive), acknowledged, or both. |
| LLInAlarmUnack | BOOL | Combined Low Low condition active and unacknowledged status. Set when the Low Low condition is active (LLInAlarm is set) and unacknowledged. Cleared when the Low Low condition is normal (inactive), acknowledged, or both. |
| ROCPosInAlarmUnack | BOOL | Combined positive rate-of-change condition active and unacknowledged status. Set when the positive rate-of-change condition is active (ROCPosInAlarm is set) and unacknowledged. Cleared when the positive rate-of-change condition is normal (inactive), acknowledged, or both. |
| ROCNegInAlarmUnack | BOOL | Combined negative rate-of-change condition active and unacknowledged status. Set when the negative rate-of-change condition is active (ROCNegInAlarm is set) and unacknowledged. Cleared when the negative rate-of-change condition is normal (inactive), acknowledged, or both. |
| Suppressed | BOOL | Suppressed status of the alarm. Set when the alarm is suppressed. Cleared when the alarm is not suppressed. |
| Disabled | BOOL | Disabled status of the alarm. Set when the alarm is disabled. Cleared when the alarm is enabled. |
| MinDurationACC | DINT | Elapsed time since an alarm condition was detected. When this value reaches MinDurationPRE, all detected alarm level conditions become active (xInAlarm is set), and a notification is sent to clients. |
| HHInAlarmTime | LINT | Timestamp when the ALMA instruction detected that the In value exceeded the High High condition limit for the most recent transition to the active state. |
| HHAlarmCount | DINT | The number of times the High High condition has been activated. If the maximum value is reached, the counter leaves the value at the maximum count value. |
| HInAlarmTime | LINT | Timestamp when the ALMA instruction detected that the In value exceeded the High condition limit for the most recent transition to the active state. |
| HAlarmCount | DINT | The number of times the High condition has been activated. If the maximum value is reached, the counter leaves the value at the maximum count value. |
| LInAlarmTime | LINT | Timestamp when the ALMA instruction detected that the In value exceeded the Low condition limit for the most recent transition to the active state. |
| LAlarmCount | DINT | The number of times the Low condition has been activated. If the maximum value is reached, the counter leaves the value at the maximum count value. |
| LLInAlarmTime | LINT | Timestamp when the ALMA instruction detected that the In value exceeded the Low Low condition limit for the most recent transition to the active state. |
| LLAlarmCount | DINT | The number of times the Low Low condition has been activated. If the maximum value is reached, the counter leaves the value at the maximum count value. |
| ROCPosInAlarmTime | LINT | Timestamp when the ALMA instruction detected that the In value exceeded the positive rate-of-change condition limit for the most recent transition to the active state. |
| ROCPosInAlarmCount | DINT | The number of times the positive rate-of-change condition has been activated. If the maximum value is reached, the counter leaves the value at the maximum count value. |
| ROCNegInAlarmTime | LINT | Timestamp when the ALMA instruction detected that the In value exceeded the negative rate-of-change condition limit for the most recent transition to the active state. |
| ROCNegAlarmCount | DINT | The number of times the negative rate-of-change condition has been activated. If the maximum value is reached, the counter leaves the value at the maximum count value. |

| 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 | Delivery error of the alarm notification message. Set when there is a delivery error: either no alarm subscriber was subscribed or at least one subscriber did not receive the latest alarm change state message. Cleared when delivery is successful or is in progress. |
| DeliveryDN | BOOL | Delivery completion of the alarm notification message. Set when delivery is successful: at least one subscriber was subscribed and all subscribers received the latest alarm change state message successfully. Cleared when delivery does not complete successfully or is in progress. |
| DeliveryEN | BOOL | Delivery status of the alarm notification message. Set when delivery is in progress. Cleared when delivery is not in progress. |
| NoSubscriber | BOOL | Alarm had no subscribers when attempting to deliver the most recent message. Set when there are no subscribers. Cleared when there is at least one subscriber. |
| NoConnection | BOOL | Alarm's subscribers were not connected when attempting to deliver the most recent message. Set when all subscribers are disconnected. Cleared when at least one subscriber is connected or there are no subscribers. |
| CommError | BOOL | Communication error when delivering an alarm message. Set when there are communication errors and all retries are used. This means that a subscriber was subscribed and it had a connection, but the controller did not receive confirmation of message delivery. Cleared when all connected subscribers confirm receipt of the alarm message. |
| AlarmBuffered | BOOL | Alarm message buffered due to a communication error (CommError is set) or a lost connection (NoConnection is set). Set when the alarm message is buffered for at least one subscriber. Cleared when the alarm message is not buffered. |
| Subscribers | DINT | Number of subscribers for this alarm. |
| SubscNotified | DINT | Number of subscribers successfully notified about the most recent alarm state change. |
| Status | DINT | Combined status indicators: <br><br> Status.0 = InstructFault. <br><br> Status.1 = InFaulted. <br><br> Status.2 = SeverityInv. <br><br> Status.3 = AlarmLimitsInv. <br><br> Status.4 = DeadbandInv. <br><br> Status.5 = ROCPosLimitInv. <br><br> Status.6 = ROCNegLimitInv. <br><br> Status.7 = ROCPeriodInv. |
| InstructFault (Status.0) | BOOL | Instruction error conditions exist. This is not a minor or major controller error. Check the remaining status bits to determine what occurred. |

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

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

The ALMA instruction provides additional functionality when used with 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. Use the output parameters to 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.

# State Diagrams when Acknowledgement Required
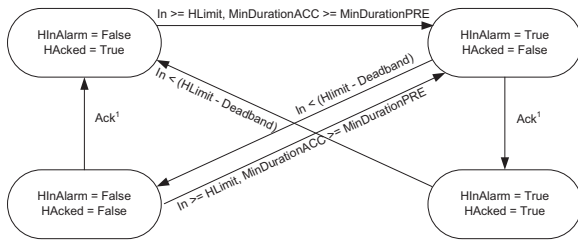


[1] H alarm condition can be acked by several different ways: HProgAck, HOperAck, ProgAckAll, OperAckAll, clients (RSLogix 5000 software, RSView software).
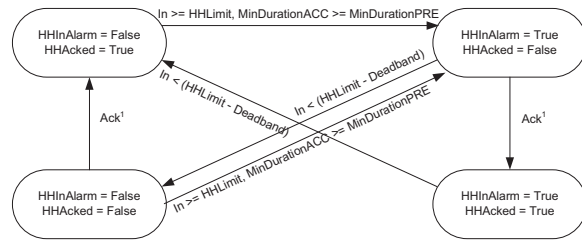


[1] HH alarm condition can be acked by several different ways: HHProgAck, HHOperAck, ProgAckAll, OperAckAll, clients (RSLogix 5000 software, RSView software).
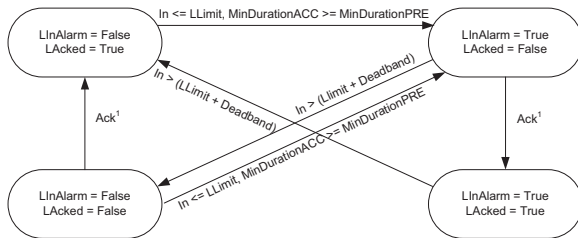


[1] L alarm condition can be acked by several different ways: LProgAck, LOperAck, ProgAckAll, OperAckAll, clients (RSLogix 5000 software, RSView software).
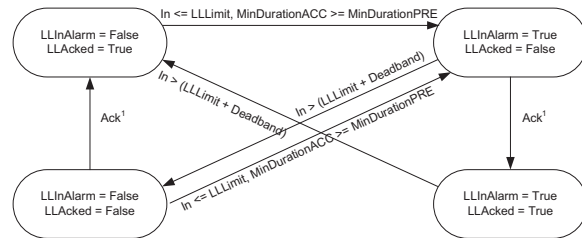


[1] LL alarm condition can be acked by several different ways: LLProgAck, LLOperAck, ProgAckAll, OperAckAll, clients (RSLogix 5000 software, RSView software).

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

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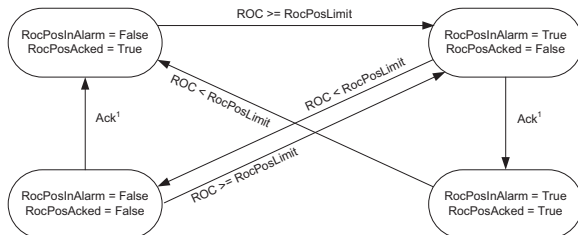


[1] ROCPos alarm condition can be acked by several different ways: RocPosProgAck, RocPosOperAck, ProgAckAll, OperAckAll, clients (RSLogix 5000 software, RSView software).
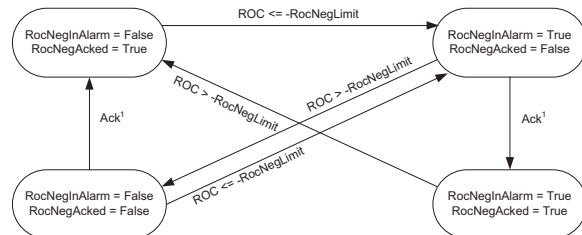


[1] ROCNeg alarm condition can be acked by several different ways: RocNegProgAck, RocNegOperAck, ProgAckAll, OperAckAll, clients (RSLogix 5000 software, RSView software).

## State Diagrams when Acknowledgement Not Required



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

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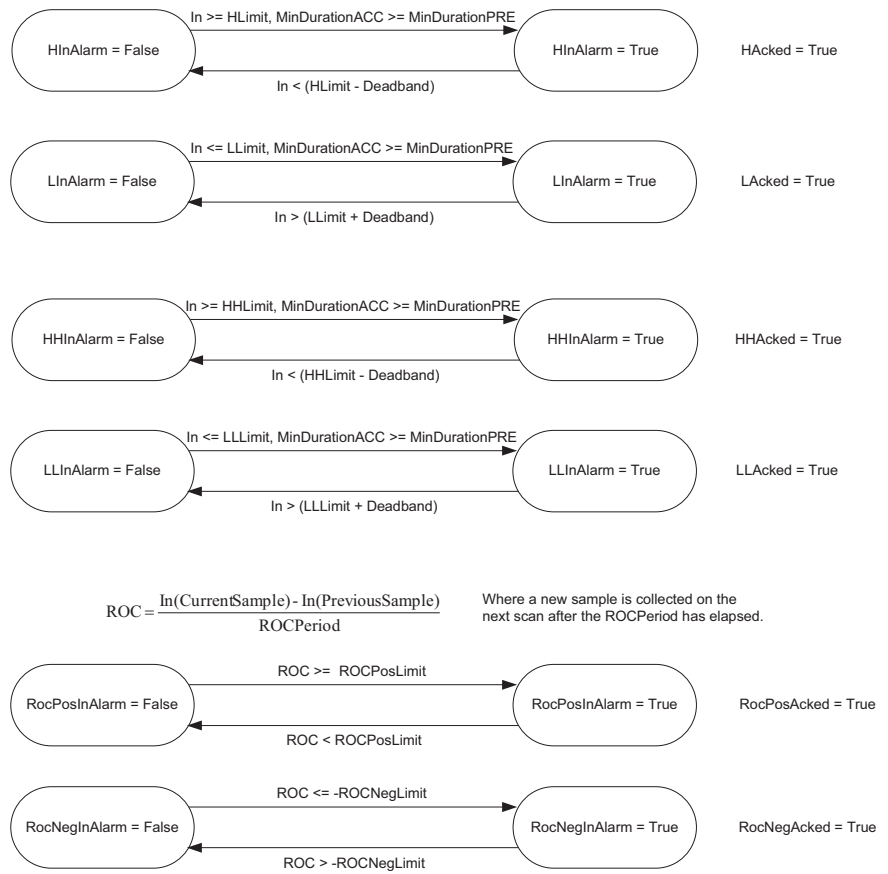


**Arithmetic Status Flags:**     Arithmetic status flags are set for the ROC output.

**Fault Conditions:**

| Minor Fault | Fault Type | Fault Code |
|---|---|---|
| ROC overflow | 4 | 4 |

**Execution:**

| Condition | Relay Ladder Action |
|---|---|
| prescan | The rung-condition-out is set to false. |
|  | All the *x*InAlarm parameters are cleared and all alarm conditions are acknowledged. |
|  | All operator requests, timestamps, and delivery flags are cleared. |
| rung-condition-in is false | The instruction does not execute. |
|  | EnableOut is cleared. |

| Condition | Relay Ladder Action |
|---|---|
| rung-condition-in is true | The instruction executes.<br><br>EnableOut is set. |
| postscan | The rung-condition-out is set to false. |

| Condition | Function Block Action | Structured Text Action |
|---|---|---|
| prescan | All operator requests, timestamps, and delivery flags are cleared.<br><br>All the *x*InAlarm parameters are cleared and all alarm conditions are acknowledged. | All operator requests, timestamps, and delivery flags are cleared.<br><br>All the *x*InAlarm parameters are cleared and all alarm conditions are acknowledged. |
| instruction first scan | No action taken. | No action taken. |
| instruction first run | No action taken. | No action taken. |
| EnableIn is cleared | The instruction does not execute.<br><br>EnableOut is cleared. | The instruction executes.<br><br>EnableOut is always set. |
| EnableIn is set | The instruction executes.<br><br>EnableOut is set. | The instruction executes.<br><br>EnableOut is always set. |
| postscan | No action taken. | No action taken. |

# ALMA Level Condition Acknowledge Required
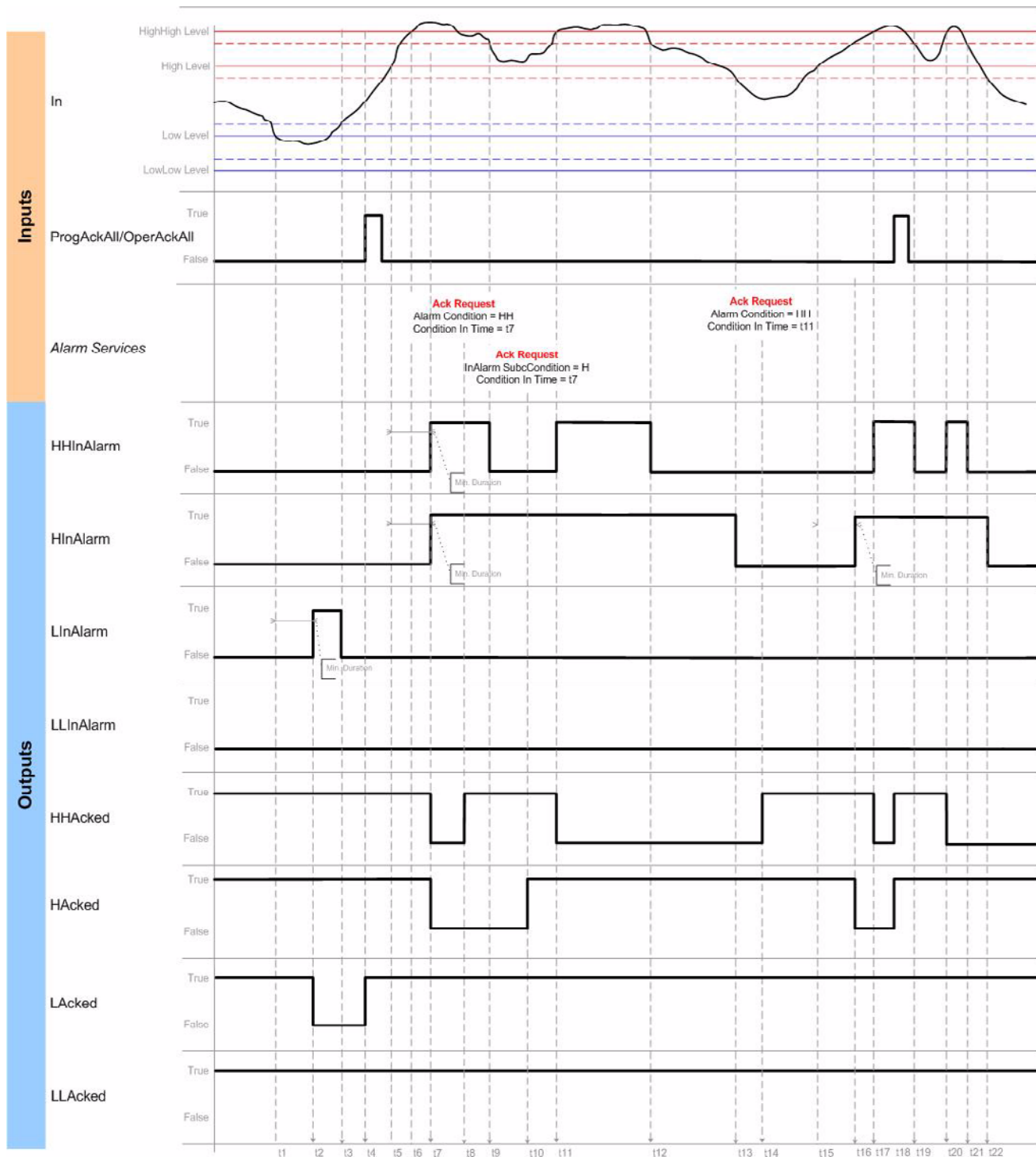
# ALMA Level Condition Acknowledge Not Required



Under this configuration, this signal does not have any influence on alarm state change.

# ALMA Rate of Change Acknowledge Required



Publication 1756-RM003K-EN-P - July 2008
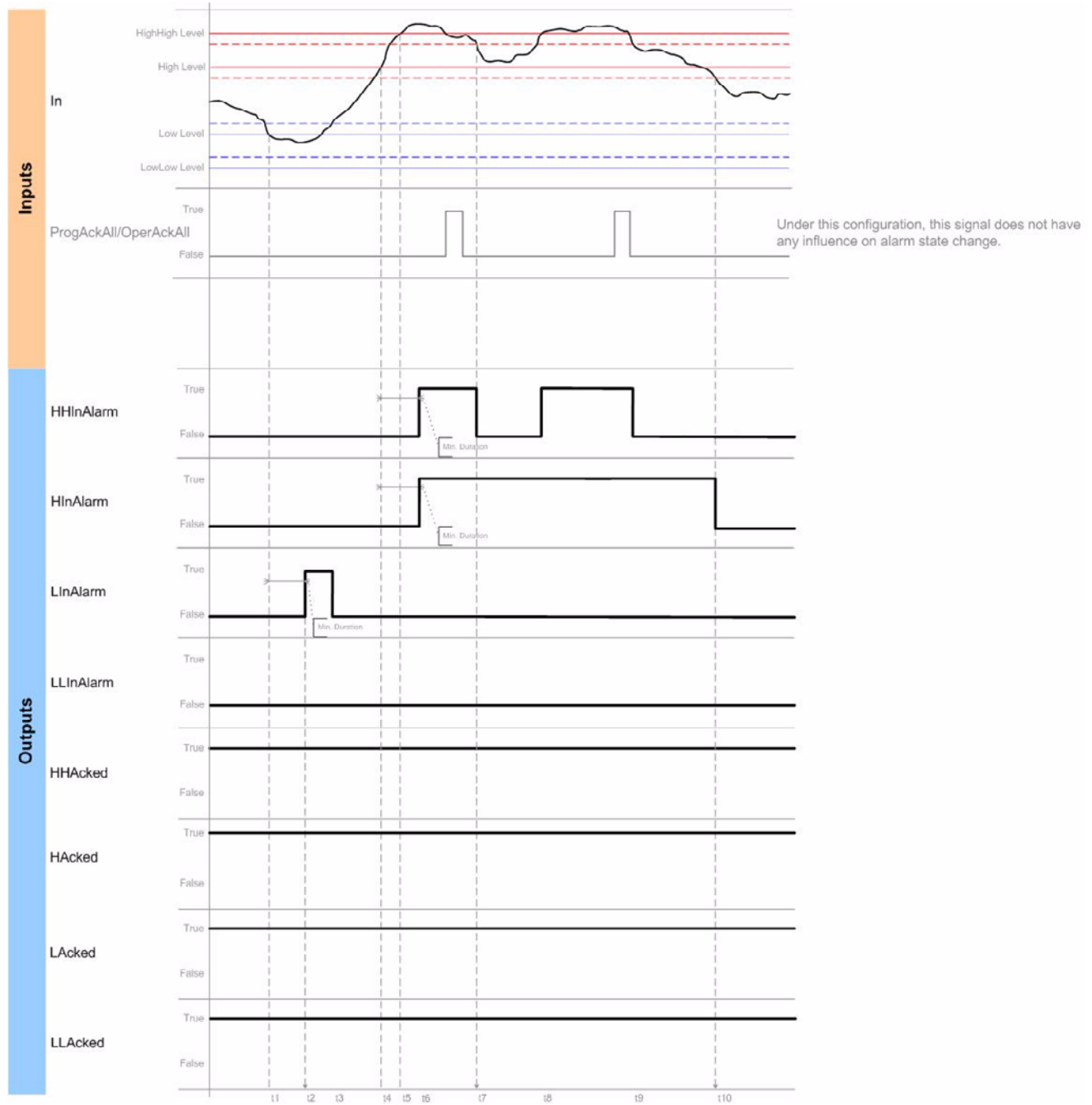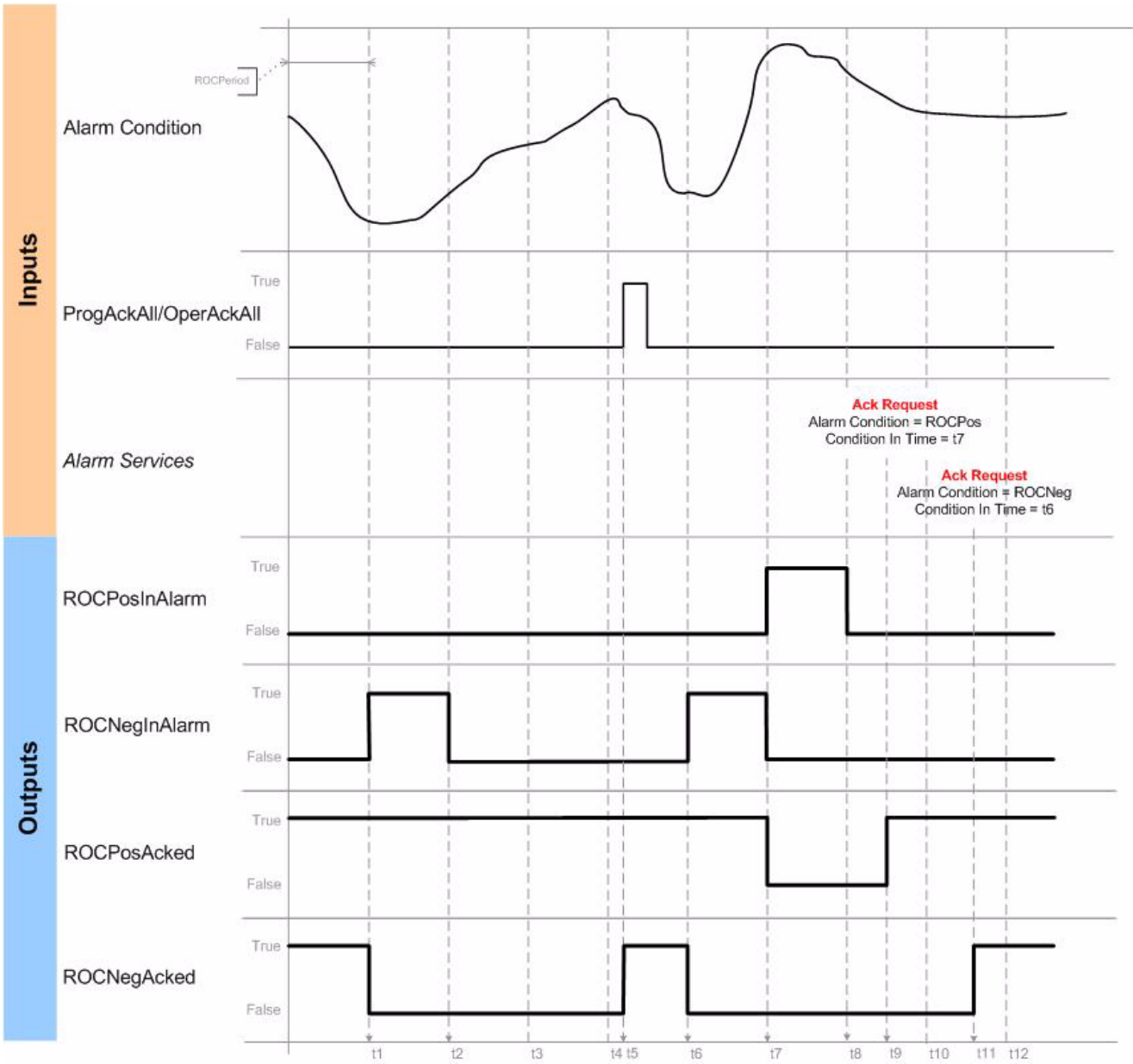
## ALMA Rate of Change Acknowledge Not Required
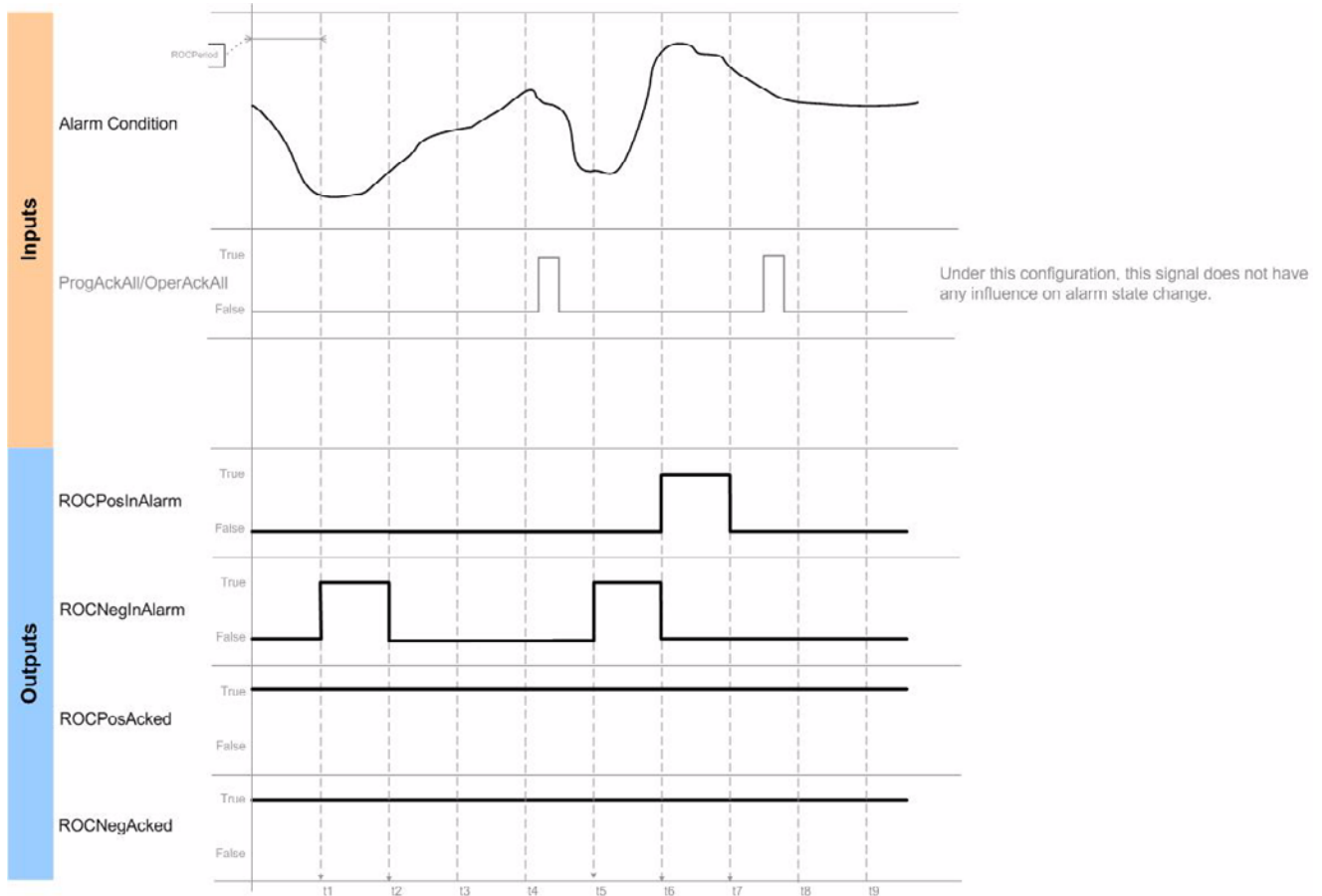


**Example:**    A tank alarm is activated if the tank level surpasses a High or High High limit. Programmatically acknowledge all the alarm conditions with a cleared-to-set

transition of the Tank32LevelAck tag value. The application logic must clear Tank32LevelAck.

## Relay Ladder
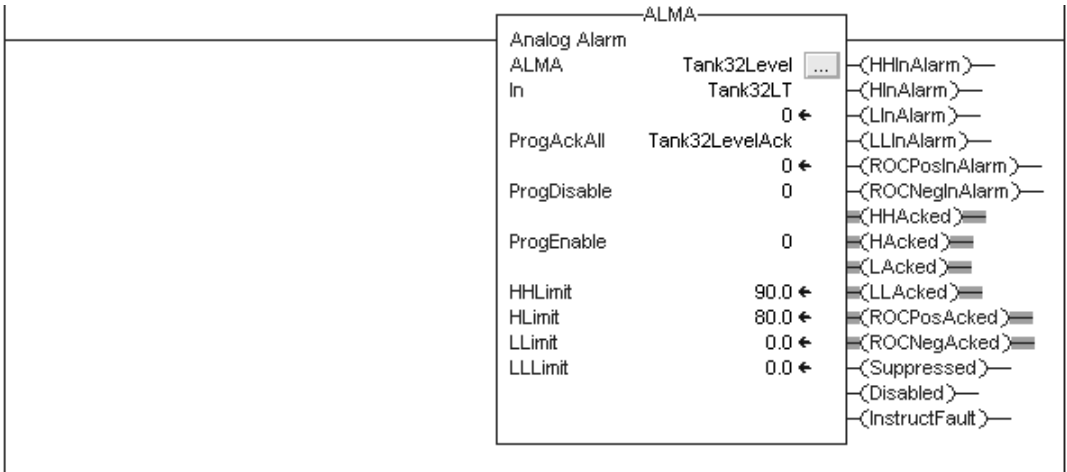


## Structured Text

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

## Function Block
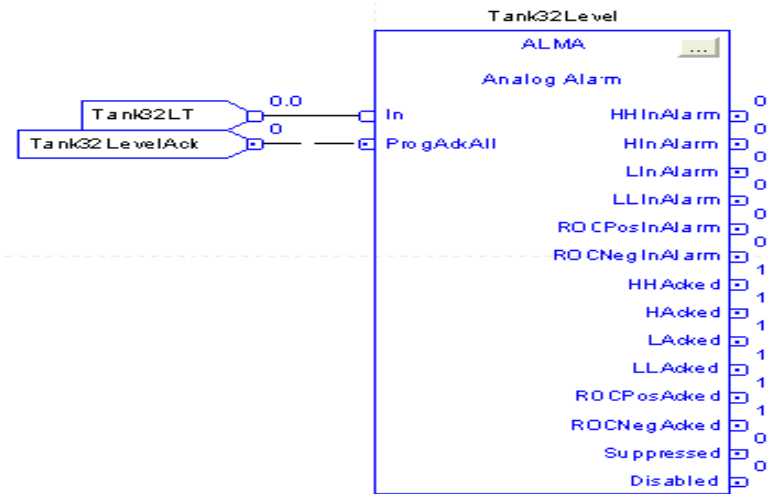
## Configure an Alarm Instruction

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



Click here to configure the instruction.
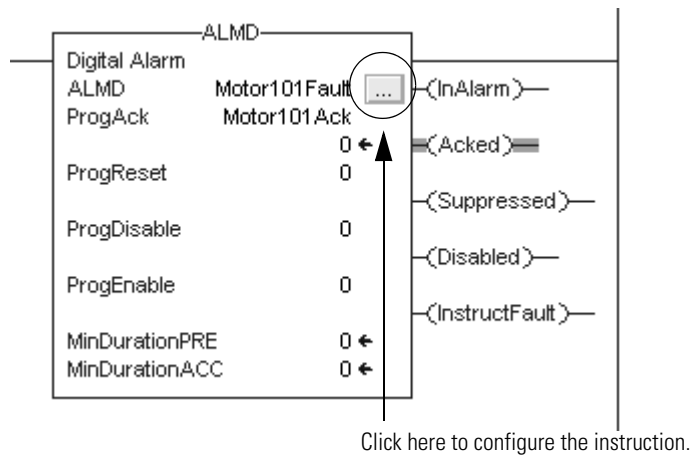
The Properties dialog for the alarm instruction includes a Configuration tab.



For each alarm instruction, configure this information.

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

| Option | Description |
|---|---|
| Severity | Select a severity range from 1...1000 to rank the importance of an alarm condition. A severity of 1 is for low priority alarms; a severity of 1000 is for an emergency condition. |
| | By default, in the FactoryTalk Alarms and Events system, severity ranges are mapped to priorities as follows: |
| | <ul><li>1...250 are low priority.</li><li>251...500 are medium priority.</li><li>501...750 are high priority.</li><li>751...1000 are urgent priority.</li></ul> |
| | You can configure the severity-to-priority mapping in the FactoryTalk Alarms and Events system. See the FactoryTalk help for details. |
| Minimum Duration | Enter the amount of time in ms an alarm condition must be active before reporting the alarm. |
| Latched - ALMD instruction | Select Latched if you want the alarm to stay active (InAlarm) after the alarm condition returns to inactive (normal). Latched alarms require a reset command to transition to normal. The reset command must be received after the condition returns to normal. |
| | Acknowledge commands will not reset a latched alarm. |
| Deadband - ALMA instruction | Specify a Deadband value to reduce alarm condition chattering caused by small fluctuations in the In value. |
| | The deadband value does not affect the alarm limit for the transition into the active state, and is also not used during the Minimum Duration interval. |
| | Once a level condition becomes active (InAlarm), it will remain active until the In value crosses back over the limit by the specified deadband. For example, if the High limit is 80, the Low limit is 20, and the Deadband is 5, the High condition will be active at $\geq$ 80 and return to normal at $\leq$75; the Low condition will be active at $\leq$20 and return to normal at $\geq$ $\leq$ 25. |
| | The Deadband has no effect on Rate of Change alarm conditions. |
| Acknowledgement Required | Alarms are configured to require acknowledgement by default. Acknowledgement indicates that an operator is aware of the alarm condition, whether or not conditions have returned to normal. |
| | Clear the Acknowledgement Required setting when you want the alarm to appear and disappear from the Alarm Summary on the HMI with no operator interaction. |
| | Alarms that do not require acknowledgement always have the Acked status set. |
| | If a digital alarm is configured as latched, the reset command also acknowledges the alarm. |

| Option | Description |
|---|---|
| Alarm class | Use the alarm class to group related alarms. Specify the alarm class exactly the same for each alarm you want in the same class. The alarm class is case sensitive. |
| | For example, specify class Tank Farm A to group all the tank alarms for a specific area. Or specify class Control Loop to group all alarms for PID loops. |
| | You can then display and filter alarms at the HMI based on the class. For example, an operator can display all tank alarms or all PID loop alarms. |
| | The alarm class does not limit the alarms that an Alarm Summary object subscribes to. Use the alarm class to filter the alarms that display to an operator once they have been received by the Alarm Summary object. FactoryTalk View software can filter the alarm class substituting wild cards for characters. |
| View command | Execute a command on the operator station when requested by an operator for a specific alarm. This lets an operator execute any standard FactoryTalk View command, such as call specific faceplates and displays, execute macros, access help files, and launch external applications. When the alarm condition occurs and is displayed to the operator, a button on the summary and banner displays lets the operator run an associated view command. |
| | Be careful to enter the correct command syntax and test the command at runtime as there is no error checking performed when the command is entered. |

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

# Enter Alarm Message Text

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

the Configuration tab. For an ALMA instruction, you enter the message information on the Message tab.
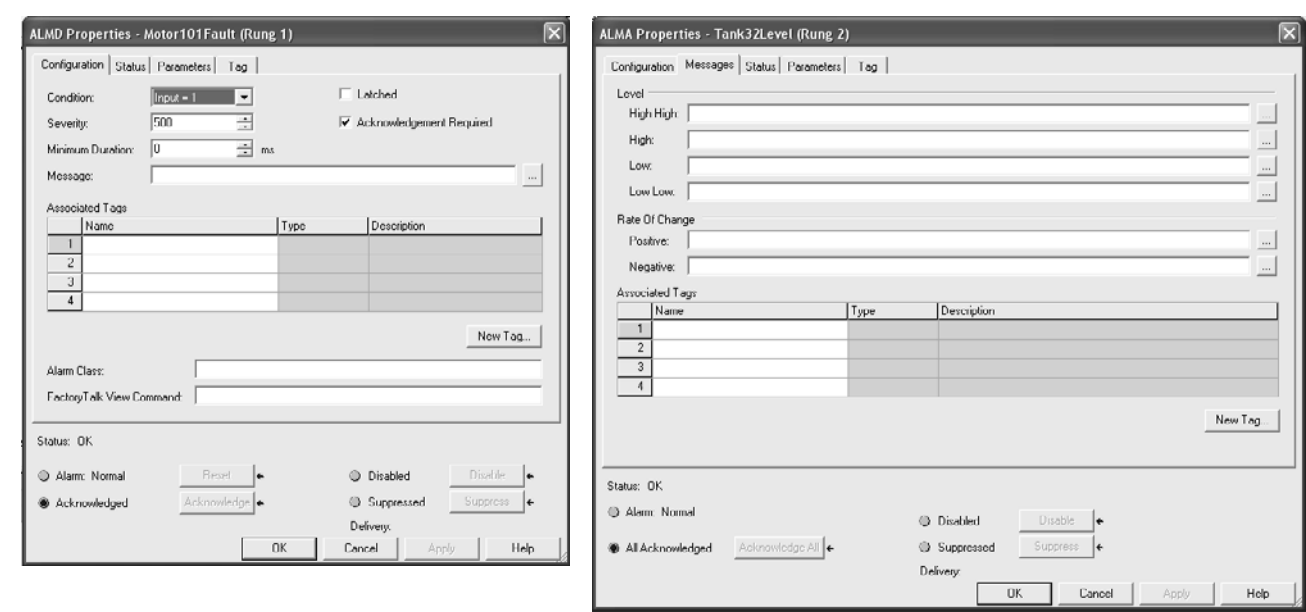


To define an alarm message, specify this information.

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

## Message String Variables

You can embed this variable information in a message string.

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

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

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

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

## Multiple Language Versions of Alarm Messages

You can maintain alarm messages in multiple languages. Either enter the different languages in the associated language versions of RSLogix 5000 programming software or in an import/export (.CSV or .TXT) file.

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

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

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

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

## Monitor Alarm Status

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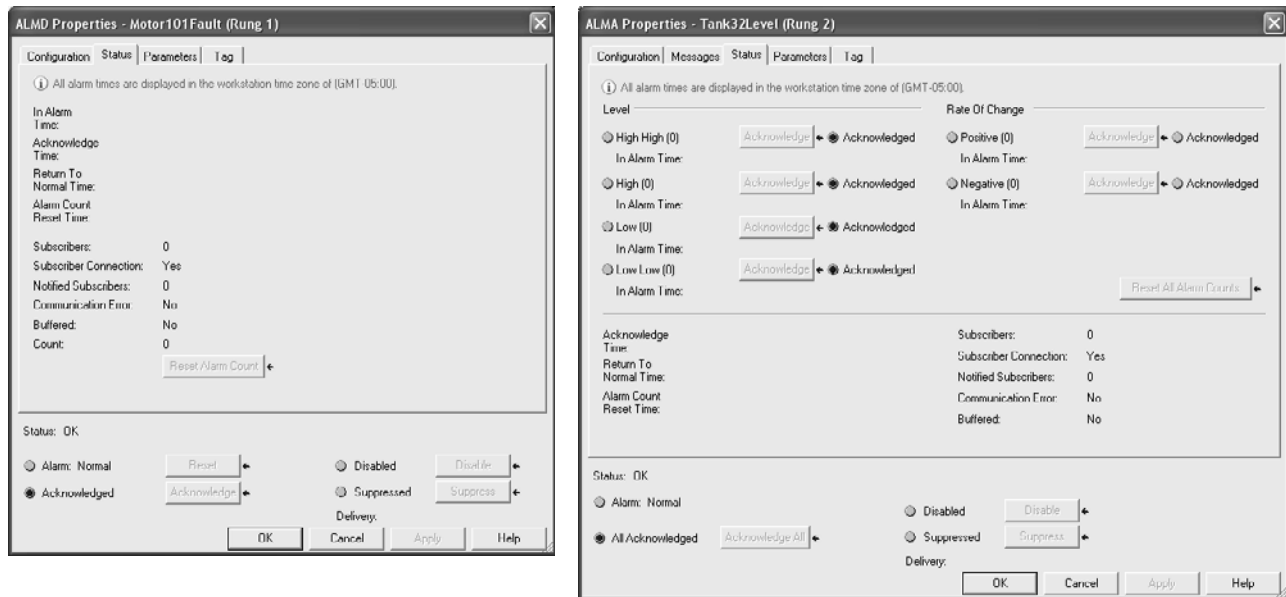
## Buffering Alarms

In order to receive controller-based alarm messages, alarm clients (such as an RSLinx Enterprise server) must establish a subscription to the alarms in the Logix controller. The controller maintains a connection to each subscriber and monitors the status of that connection.

As alarm state changes occur, the alarm instructions in the controller cache the necessary information (such as timestamps and associated tag values) and request the transmission of the alarm message to all of the subscribers. The publisher mechanism delivers the alarm messages to each subscriber as quickly as possible.

If any subscriber fails to confirm receipt of the alarm message, or if the connection to a known subscriber is not good, the publisher mechanism stores the undelivered alarm messages in a 100 KB buffer. Each subscriber has its own buffer so communication problems with one subscriber do not interfere with alarm delivery to other subscribers. When the buffer is full, newer alarm messages are discarded. The buffer is created when the subscriber establishes its initial connection and is maintained for a configurable length of time (0...120 minutes, default is 20 minutes) after a subscriber loses its connection.

When the subscriber re-establishes a connection within the buffer timeout interval, it obtains the current state of all alarms, begins to receive current alarm messages, and also uploads any buffered messages that may have accumulated. Even if the buffer was full, and messages were discarded, the

subscribers accurately synchronize to the current state of the alarms (including the most recent InAlarmTime, RetToNormalTime, and AckTime timestamps).

The buffer continues until is filled. Once filled, the buffer stops adding alarm transitions until space is made available in the buffer by the subscriber.

# Programmatically Access Alarm Information

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

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

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

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

- ProgAck
- ProgReset
- ProgDisable
- ProgEnable

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

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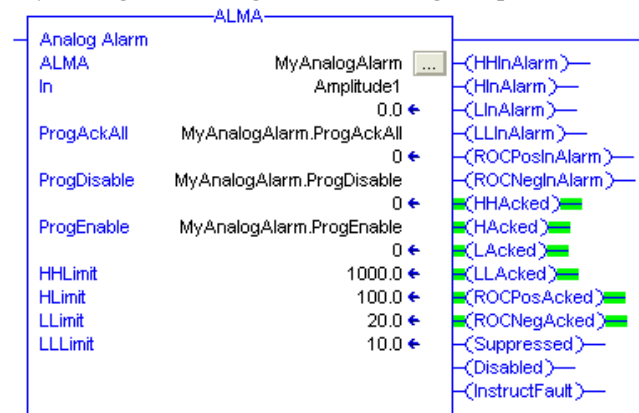


## Suppress or Disable Alarms

Suppress alarms to remove alarms you know exist from the HMI but still keep the alarms alive. This lets you clear the alarm summary while you are resolving a known alarm without continuing to view alarm information. A suppressed alarm does not appear on the operator summary or banner screens, but a suppressed alarm is still sent to subscribers, logged in the historical database, able to transition alarm status, time stamped, and responsive to other programmatic or operator interactions.

- When an alarm is Suppressed, it continues to function normally, monitor the In parameter for alarm conditions, and respond to Acknowledge requests. All subscribers are notified of this event, and any alarm messages generated while the alarm is in the Suppressed state include the Suppressed status. Alarm clients can respond differently to Suppressed alarms. For example, suppressed alarms can be logged to the historical database but not annunciated to the operator.

- When an alarm is Unsuppressed, all subscribers are notified and alarm messages to subscribers no longer include the Suppressed status.

Disable an alarm to treat the alarm as if it does not exist in the control program. A disabled alarm does not transition alarm status or get logged in the historical database. A disabled alarm is still tracked, and can be re-enabled, in the Alarm Status Explorer in FactoryTalk View SE software.

- When an alarm is Disabled, all of its conditions are set to the initial state (InAlarm is cleared and Acked is set). The In parameter is not monitored for alarm conditions. All subscribers are notified of this event.

- When an alarm is Enabled, it begins to monitor the In parameter for alarm conditions. All subscribers are notified of this event.

## Controller-based Alarm Execution

Controller-based alarms process inputs from two sources.

| Source | Description |
|---|---|
| Alarm tag members | Alarm tag members are, for the most part, processed when the user application scans the alarm instruction. This includes:<br><br>• processing changes to configuration parameters.<br>• evaluating the alarm condition.<br>• measuring elapsed time for MinDuration.<br>• capturing InAlarmTime and RetToNormalTime timestamps.<br>• capturing associated tag values.<br>• processing Prog and Oper commands.<br><br>In addition, these alarm tag status members are updated as alarm messages are delivered to each subscriber, asynchronously to the program scan:<br><br>• DeliveryEN, DeliveryER, DeliveryDN<br>• NoSubscriber, NoConnection, CommError, AlarmBuffered, SubscNotified |
| Client messages | Client messages are processed as they are received, asynchronously to the program scan.<br><br>• Reset, Acknowledge, Disable/Enable, and Suppress/Unsuppress commands from an RSLogix 5000 terminal<br>• Reset, Acknowledge, Disable/Enable, and Suppress/Unsuppress commands from a FactoryTalk View SE alarm subscriber |

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

## Controller Memory Use

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

- Typically 1 KB per digital alarm with no associated tags

| Digital Alarm Example | Approximate Size |
|---|---|
| Digital alarm with no associated tags and this configuration:<br><br>• Alarm message: Contactor Fault<br>• Alarm Class: Tank Farm A | 1012 bytes |
| Digital alarm with two associated tags and this configuration:<br><br>• Alarm message: Contactor Fault<br>• Alarm Class: Tank Farm A<br>• Associated Tag 1 = DINT data type<br>• Associated Tag 2 = DINT data type | 1100 bytes |
| Digital alarm with two associated tags and this configuration:<br><br>• Alarm message: Contactor Fault<br>• Alarm Class: Tank Farm A<br>• Associated Tag 1 = DINT data type<br>• Associated Tag 2 = STRING data type | 1522 bytes |

• Typically 2.2 KB per analog alarm with no associated tags

| Analog Alarm Example | Approximate Size |
|---|---|
| Analog alarm with no associated tags and this configuration:<br><br>• HH Alarm message: Level Alarm<br>• H Alarm Message: Level Alarm<br>• L Alarm Message: Level Alarm<br>• LL Alarm Message: Level Alarm<br>• Rate of Change Positive Message: Fill Too Fast<br>• Rate of Change Negative Message: Empty Too Fast<br>• Alarm Class: Tank Farm A | 2228 bytes |
| Analog alarm with two associated tags and this configuration:<br><br>• HH Alarm message: Level Alarm<br>• H Alarm Message: Level Alarm<br>• L Alarm Message: Level Alarm<br>• LL Alarm Message: Level Alarm<br>• Rate of Change Positive Message: Fill Too Fast<br>• Rate of Change Negative Message: Empty Too Fast<br>• Alarm Class: Tank Farm A<br>• Associated Tag 1 = DINT data type<br>• Associated Tag 2 = DINT data type | 2604 bytes |
| Analog alarm with two associated tags and this configuration:<br><br>• HH Alarm message: Level Alarm<br>• H Alarm Message: Level Alarm<br>• L Alarm Message: Level Alarm<br>• LL Alarm Message: Level Alarm<br>• Rate of Change Positive Message: Fill Too Fast<br>• Rate of Change Negative Message: Empty Too Fast<br>• Alarm Class: Tank Farm A<br>• Associated Tag 1 = DINT data type<br>• Associated Tag 2 = STRING data type | 4536 bytes |

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

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

## Scan Time

These execution times show how ALMD instructions and ALMA instructions affect total scan time.

| Rung State | | Execution Times | |
|---|---|---|---|
| | | **Digital Alarm (ALMD)** | **Analog Alarm (ALMA)** |
| No Alarm State Change | Rung False | 8 µs | 17 µs |
| | Rung True | 8 µs | 60 µs |
| Alarm State Change | Rung False | 35 µs | 17 µs |
| | Rung True | 35 µs | 126 µs |

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

**Notes:**

# Bit Instructions

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

## 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<br>structured text[1] | 78 |
| enable outputs when a bit is cleared | XIO | relay ladder<br>structured text[1] | 80 |
| set a bit | OTE | relay ladder<br>structured text[1] | 82 |
| set a bit (retentive) | OTL | relay ladder<br>structured text[1] | 84 |
| clear bit (retentive) | OTU | relay ladder<br>structured text[1] | 86 |
| enable outputs for one scan each time a rung goes true | ONS | relay ladder<br>structured text[1] | 88 |
| set a bit for one scan each time a rung goes true | OSR | relay ladder | 91 |
| set a bit for one scan each time the rung goes false | OSF | relay ladder | 94 |
| set a bit for one scan each time the input bit is set in function block | OSRI | structured text<br>function block | 96 |
| set a bit for one scan each time the input bit is cleared in function block | OSFI | structured text<br>function block | 99 |

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

# Examine If Closed (XIC)

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

### Operands:

### Relay Ladder

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

### Structured Text

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

```
IF data_bit THEN
    <statement>;
END_IF;
```
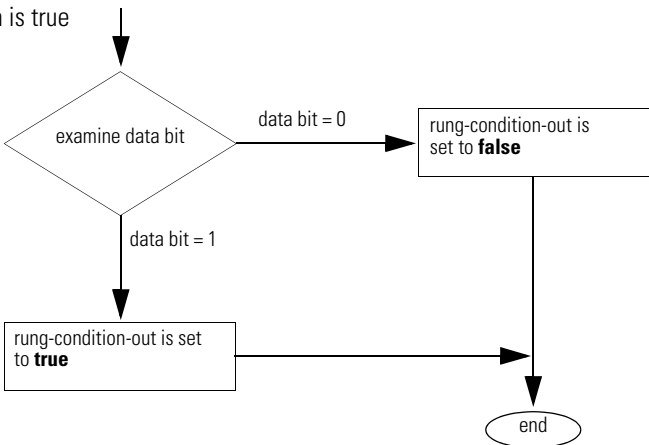
See Appendix 641B, Function Block Attributes for information on the syntax of constructs within structured text.

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

**Arithmetic Status Flags:** not affected

**Fault Conditions:** none

### Execution:

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



| | |
|---|---|
| postscan | The rung-condition-out is set to false. |

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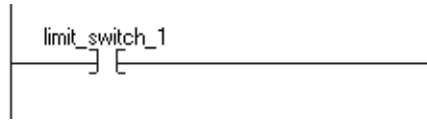
### Relay Ladder



### Structured Text

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

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

### Relay Ladder



### Structured Text

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

# Examine If Open (XIO)

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

### Operands:

#### Relay Ladder

—]/[—
?

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

#### Structured Text

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

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

See Appendix 641B, Function Block Attributes for information on the syntax of constructs within structured text.

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

**Arithmetic Status Flags:** not affected

**Fault Conditions:** none

### Execution:

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



| | |
|---|---|
| postscan | The rung-condition-out is set to false. |

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

### Relay Ladder

```
      limit_switch_2
────────]/[─────────────────────
```
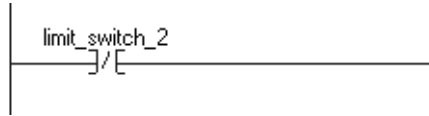
### Structured Text

```
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

```
      S:V
────────]/[─────────────────────
```
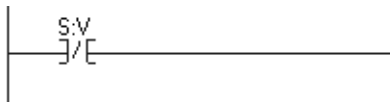
### Structured Text

```
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 641B, Function Block Attributes for information on the syntax of assignments and expressions within structured text.

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

**Arithmetic Status Flags:**    not affected

**Fault Conditions:**    none

### Execution:

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

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

**Relay Ladder**

```
        switch                                                   light_1
      ──┤ ├──────────────────────────────────────────────────────( )──
```
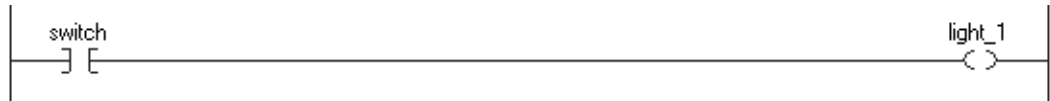
**Structured Text**

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

# Output Latch (OTL)

The OTL instruction sets (latches) the data bit.

**Operands:**

## Relay Ladder

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

## Structured Text

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

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

See Appendix 641B, Function Block Attributes for information on the syntax of constructs, expressions, and assignments within structured text.

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

**Arithmetic Status Flags:** not affected

**Fault Conditions:** none

**Execution:**

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

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

### Relay Ladder
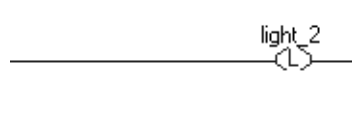


### Structured Text

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

# Output Unlatch (OTU)

The OTU instruction clears (unlatches) the data bit.

**Operands:**

📄

—(U)—

### 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 641B, Function Block Attributes for information on the syntax of constructs, expressions, and assignments within structured text.

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

**Arithmetic Status Flags:**    not affected

**Fault Conditions:**    none

**Execution:**

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

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

**Relay Ladder**



**Structured Text**

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

# One Shot (ONS)

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

## Operands:
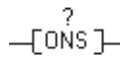
### Relay Ladder

```
       ?
—[ ONS ]—
```

| 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;
```
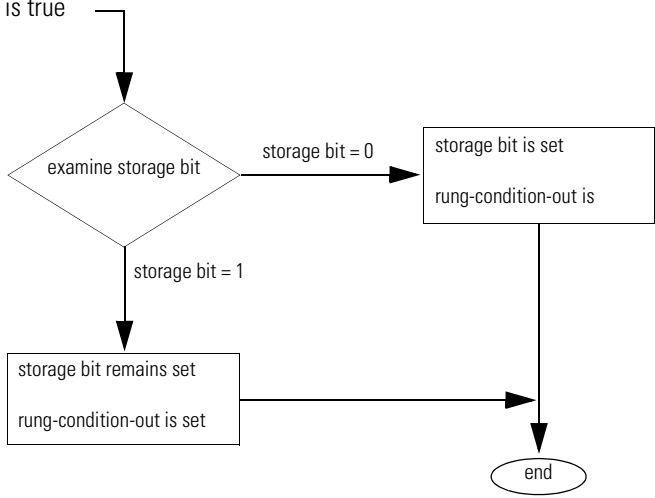
See Appendix 641B, Function Block Attributes for information on the syntax of constructs, expressions, and expressions within structured text.

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

**Arithmetic Status Flags:** not affected

**Fault Conditions:** none

**Execution:**

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

## Operands:

### Relay Ladder

```
----OSR----------
One Shot Rising         <OB>
Storage Bit        ?    <SB>
Output Bit         ?
```

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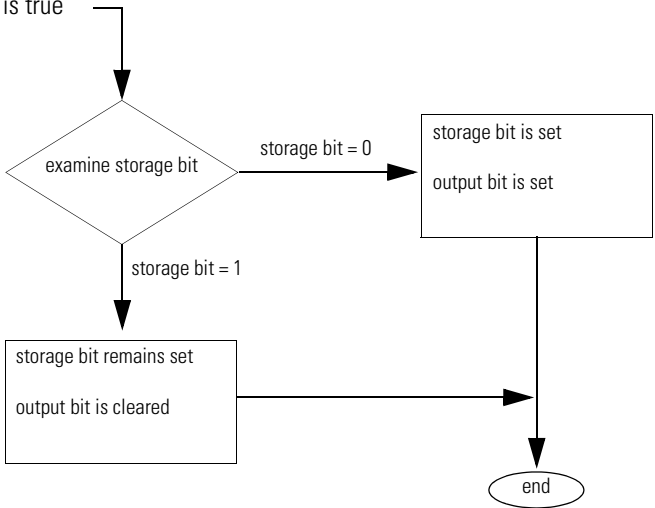
**Description:** When enabled and the storage bit is cleared, the OSR instruction sets the output bit. When enabled and the storage bit is set or when disabled, the OSR instruction clears the output bit



instruction is executed

instruction resets during next scan execution

**Arithmetic Status Flags:** not affected

**Fault Conditions:** none

**Execution:**

| Condition | Relay Ladder Action |
|---|---|
| prescan | The storage bit is set to prevent an invalid trigger during the first scan. |
| | The output bit is cleared. |
| | The rung-condition-out is set to false. |
| rung-condition-in is false | The storage bit is cleared. |
| | The output bit is not modified. |
| | The rung-condition-out is set to false. |

rung-condition-in is true

```
              ┌──────────┐
              ▼
         ◇ examine storage bit ◇ ── storage bit = 0 ──▶ ┌─────────────────┐
              │                                          │ storage bit is set │
              │ storage bit = 1                          │ output bit is set  │
              ▼                                          └─────────────────┘
   ┌──────────────────────┐                                       │
   │ storage bit remains set │ ──────────────────────▶            ▼
   │ output bit is cleared   │                              ( end )
   └──────────────────────┘
```

| | |
|---|---|
| postscan | The storage bit is cleared. |
| | The output bit is not modified. |
| | The rung-condition-out is set to false. |

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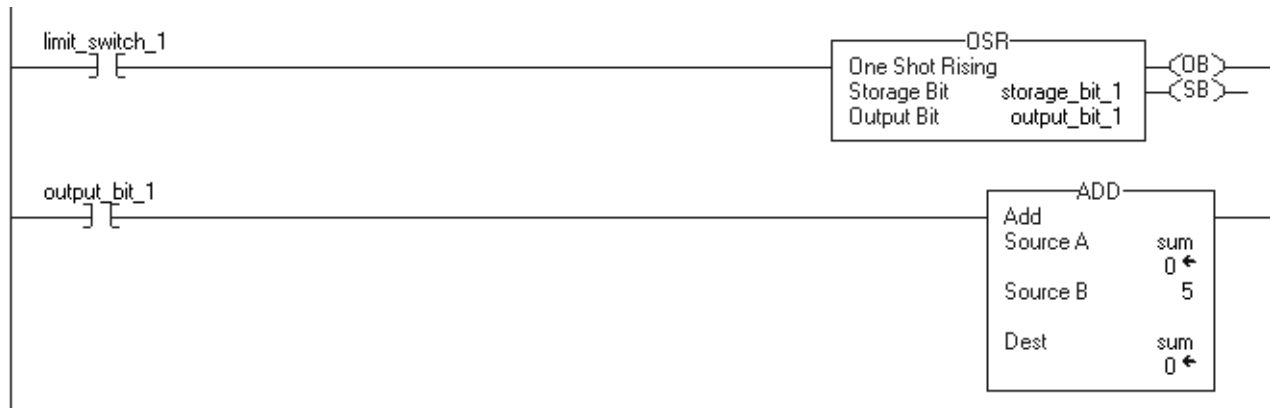
# One Shot Falling (OSF)

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

This instruction is available in structured text and function block as OSFI, see .
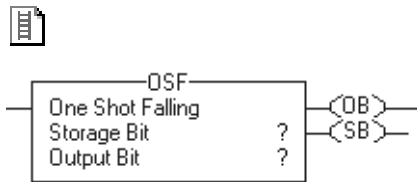
## Operands:

### Relay Ladder Operands



| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| storage bit | BOOL | tag | internal storage bit<br><br>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

Publication 1756-RM003K-EN-P - July 2008

### 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 | |



| | |
|---|---|
| rung-condition-in is true | The storage bit is set. |
| | The output bit is cleared. |
| | The rung-condition-out is set to true. |
| postscan | See rung-condition-in is false above. |

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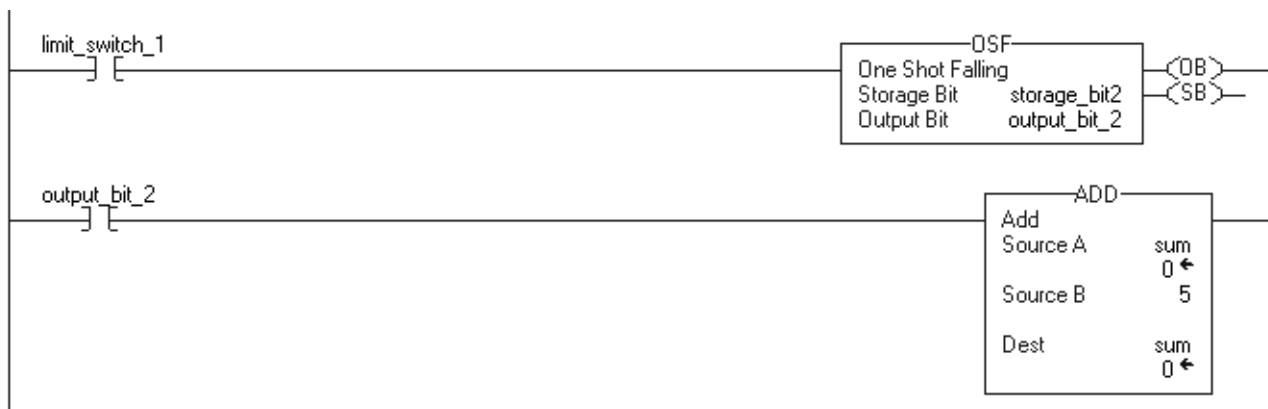
# One Shot Rising with Input (OSRI)

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

This instruction is available in relay ladder as OSR, see .

### Operands:

OSRI(OSRI_tag);

### Structured Text

| Operand | Type | Format | Description |
|---|---|---|---|
| OSRI tag | FBD_ONESHOT | structure | OSRI structure |

### Function Block

OSRI_tag
OSRI
One Shot Rising with Input
InputBit       OutputBit

| Operand | Type | Format | Description |
|---|---|---|---|
| OSRI tag | FBD_ONESHOT | structure | OSRI structure |

### FBD_ONESHOT Structure

| Input Parameter | Data Type | Description |
|---|---|---|
| EnableIn | BOOL | **Function Block:**<br><br>If cleared, the instruction does not execute and outputs are not updated.<br><br>If set, the instruction executes.<br><br>Default is set.<br><br>**Structured Text:**<br><br>No effect. The instruction executes. |
| InputBit | BOOL | Input bit. This is equivalent to rung condition for the relay ladder OSR instruction.<br><br>Default is cleared. |

| Output Parameter | Data Type | Description |
|---|---|---|
| EnableOut | BOOL | The instruction produced a valid result. |
| OutputBit | BOOL | Output bit |

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



InputBit

$InputBit_{n-1}$

OutputBit

instruction is executed

instruction resets during next scan execution

40048

**Arithmetic Status Flags:**  not affected

**Fault Conditions:**  none

**Execution:**

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

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

### Structured Text

```
OSRI_01.InputBit := limit_switch1;


OSRI(OSRI_01);


State := OSRI_01.OutputBit;
```

### Function Block
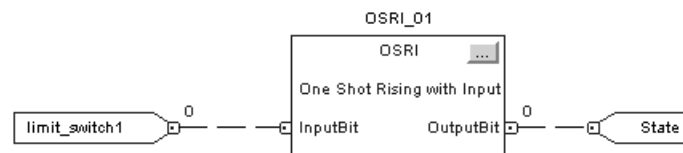
# One Shot Falling with Input (OSFI)

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

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

### Operands:

OSFI(OSFI_tag);

### Structured Text

| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| OSFI tag | FBD_ONESHOT | structure | OSFI structure |

### Function Block



OSFI_tag

OSFI

One Shot Falling with Input

InputBit    OutputBit

| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| OSFI tag | FBD_ONESHOT | structure | OSFI structure |

### FBD_ONESHOT Structure

| Input Parameter | Data Type | Description |
|-----------------|-----------|-------------|
| EnableIn | BOOL | **Function Block:**<br><br>If cleared, the instruction does not execute and outputs are not updated.<br><br>If set, the instruction executes.<br><br>Default is set.<br><br>**Structured Text:**<br><br>No effect. The instruction executes. |
| InputBit | BOOL | Input bit. This is equivalent to rung condition for the relay ladder OSF instruction<br><br>Default is cleared. |

| Output Parameter | Data Type | Description |
|------------------|-----------|-------------|
| EnableOut | BOOL | The instruction produced a valid result. |
| OutputBit | BOOL | Output bit |

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



**Arithmetic Status Flags:**    not affected

**Fault Conditions:**    none

**Execution:**

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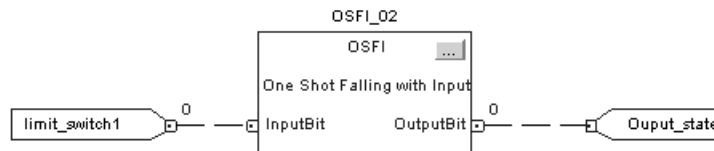
**Notes:**

# 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 | 104 |
| time how long a timer is disabled | TOF | relay ladder | 108 |
| accumulate time | RTO | relay ladder | 112 |
| time how long a timer is enabled with built-in reset in function block | TONR | structured text<br><br>function block | 116 |
| time how long a timer is disabled with built-in reset in function block | TOFR | structure text<br><br>function block | 120 |
| accumulate time with built-in reset in function block | RTOR | structured text<br><br>function block | 124 |
| count up | CTU | relay ladder | 128 |
| count down | CTD | relay ladder | 132 |
| count up and count down in function block | CTUD | structured text<br><br>function block | 136 |
| reset a timer or counter | RES | relay ladder | 141 |

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 .

## 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 ≥ .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 ≥ .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.



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

rung-condition-in is true

```
                              ┌───────────────┐
                              │ examine .DN bit│ ──── .DN bit = 1 ──────────────┐
                              └───────┬────────┘                                │
                                 .DN bit = 0                                    │
                                      │                                         │
                         ┌────────────┴──────┐                                  │
                         │  examine .EN bit   │── .EN bit = 0 ──┐               │
                         └─────────┬──────────┘         ┌───────▼────────┐      │
                              .EN bit = 1               │ .EN bit is set │      │
                                   │                    │ .TT bit is set │      │
                      ┌────────────┴────────────┐       └───────┬────────┘      │
                      │ .TT bit is set          │               │              │
                      │ .ACC = .ACC +           │       ┌───────▼────────┐      │
                      │ (current_time -         │       │ examine .ACC   │── .ACC ≥ .PRE ──┐
                      │  last_time)             │       └───────┬────────┘      │          │
                      └────────────┬────────────┘          .ACC < .PRE         │   ┌──────▼──────┐
                                   │                           │           ◄───┘   │ .DN is set  │
                         ┌─────────┴────────┐                  │                   │ .TT bit is  │
                         │ .ACC value       │── no ────┐       │                   │ cleared     │
                         │ rolls over       │          │       │                   └──────┬──────┘
                         └─────────┬────────┘          │       │                          │
                                 yes                   │  ┌────▼─────────────────┐        │
                                   │                   │  │ rung-condition-out    │◄───────┘
                      ┌────────────▼──────────┐        │  │ is set to **true**    │
                      │ .ACC = 2,147,483,647  │        │  └──────────┬───────────┘
                      └────────────┬──────────┘        │             │
                                   └─────────────────► │           ( end )
```

| | |
|---|---|
| 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.
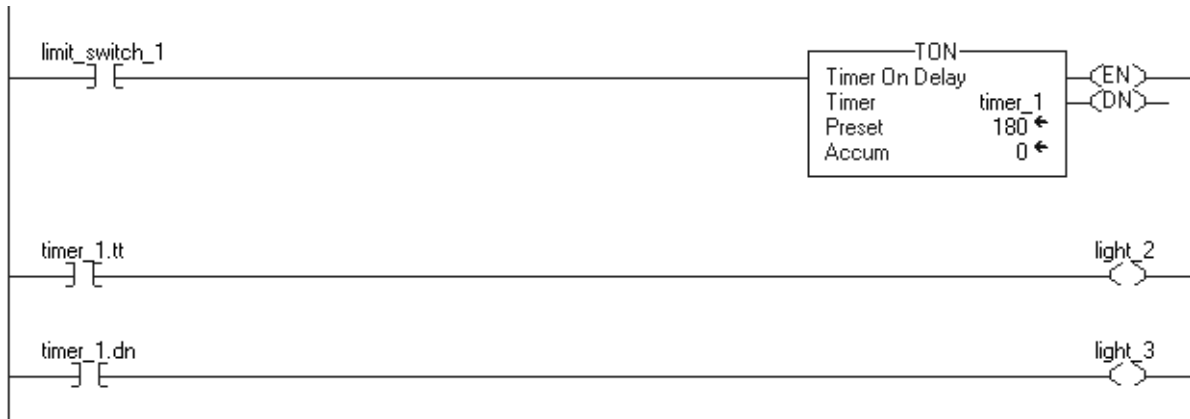
# Timer Off Delay (TOF)

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 .

## Operands:

### Relay Ladder

```
         ┌──TOF──────────────┐
─────────┤ Timer Off Delay   ├──(EN)──
         │ Timer          ?  │  (DN)──
         │ Preset         ?  │
         │ Accum          ?  │
         └───────────────────┘
```

| 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<br><br>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.



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

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

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 | The .EN, .TT, and .DN bits are cleared. |
| | The .ACC value is set to equal the .PRE value. |
| | The rung-condition-out is set to false. |
| rung-condition-in is false |  |
| rung-condition-in is true | The .EN, .TT, and .DN bits are set. |
| | The .ACC value is cleared. |
| | The rung-condition-out is set to true. |
| postscan | The rung-condition-out is set to false. |
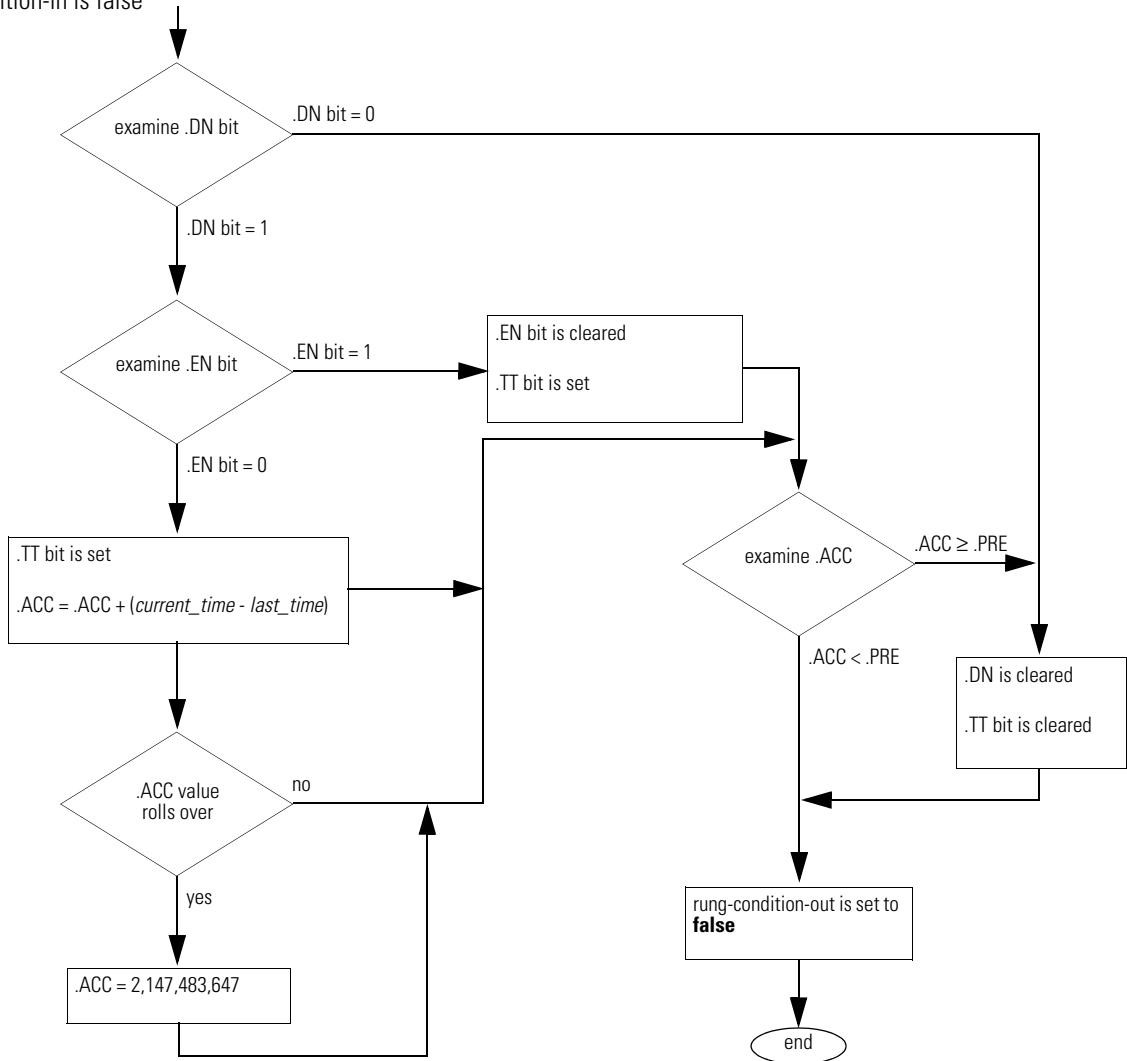
**Example:**    When *limit_switch_2* is cleared, *light_2* is on for 180 msec (*timer_2* is timing). When *timer_2.acc* reaches 180, *light_2* goes off and *light_3* goes on. *Light_3* remains on until the TOF instruction is enabled. If *limit_switch_2* is set while *timer_2* is timing, *light_2* goes off.

```
limit_switch_2                                          ┌──────TOF──────┐
     ] [                                                │ Timer Off Delay│──(EN)
                                                        │ Timer   timer_2│──(DN)──
                                                        │ Preset      180│←
                                                        │ Accum         0│←
                                                        └───────────────┘

timer_2.tt                                                              light_2
     ] [                                                                 ( )

timer_2.dn                                                              light_3
    ]/[                                                                  ( )
```
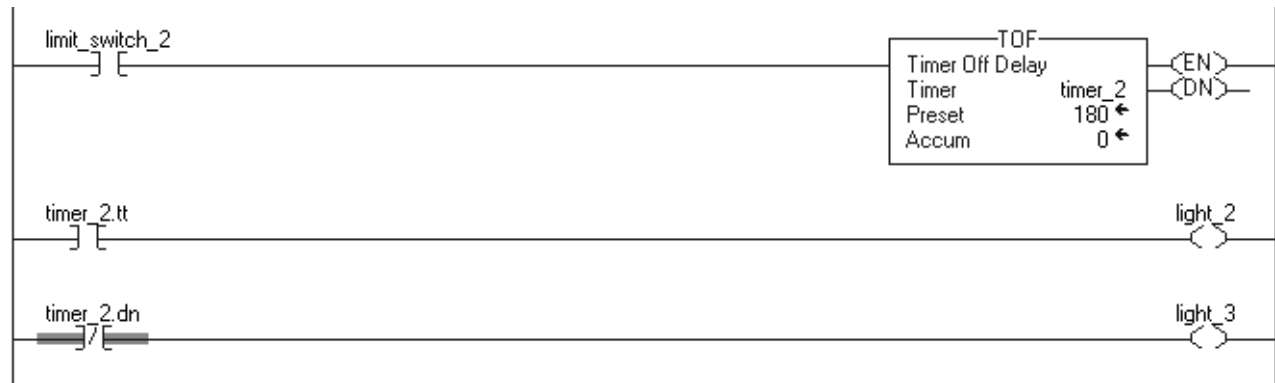
# Retentive Timer On (RTO)

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 .
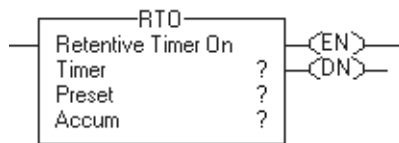
## Operands:

### 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<br><br>initial value is typically 0 |

### TIMER Structure

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

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

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



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 | The .EN, .TT, and .DN bits are cleared. |
| | The .ACC value is not modified. |
| | The rung-condition-out is set to false. |
| rung-condition-in is false | The .EN and .TT bits are cleared. |
| | The .DN bit is not modified. |
| | The .ACC value is not modified. |
| | The rung-condition-out is set to false. |

rung-condition-in is true
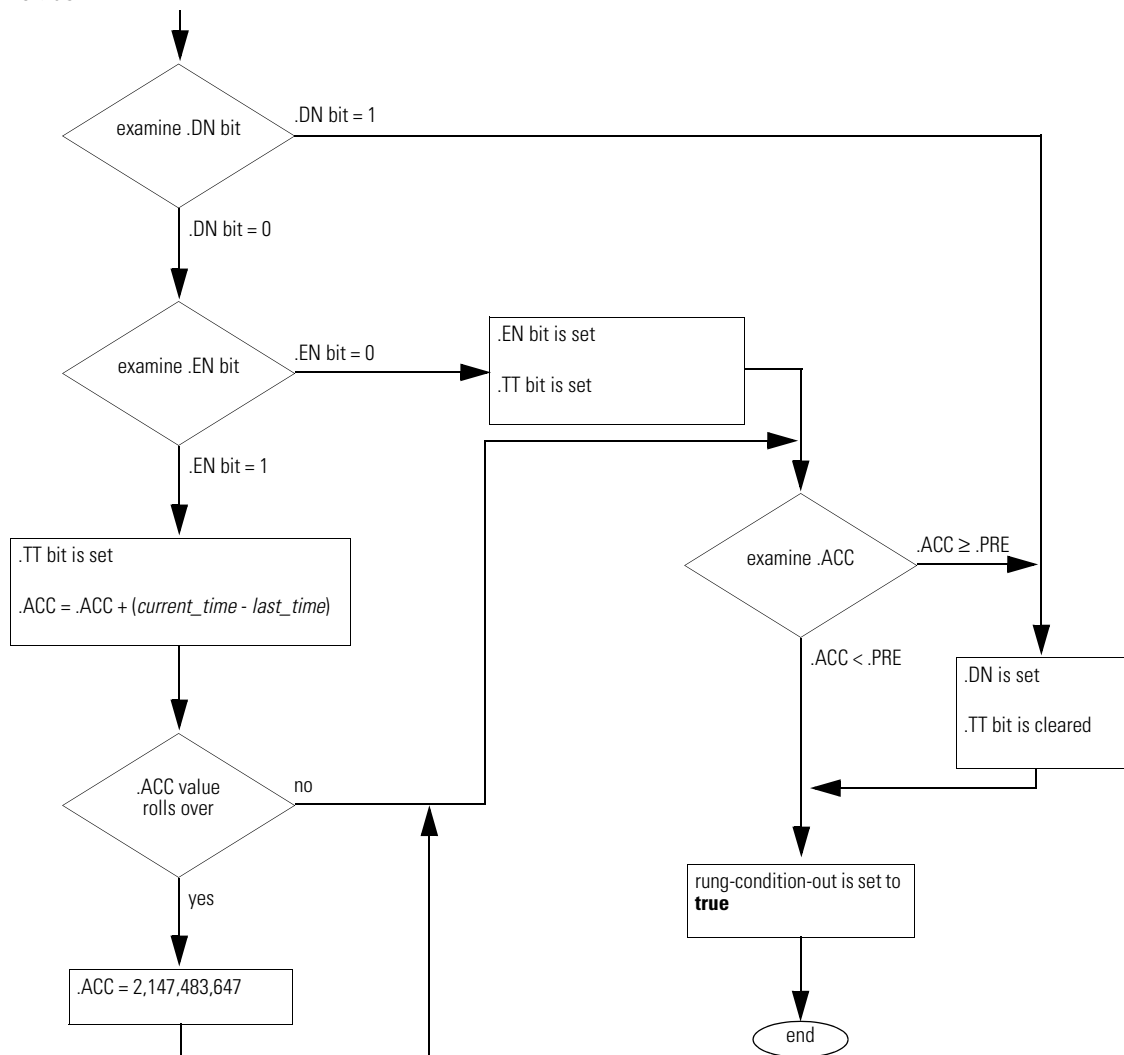


| postscan | The rung-condition-out is set to false. |
| --- | --- |

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

# Timer On Delay with Reset (TONR)

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

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

## Operands:

    TONR(TONR_tag);

### Structured Text

| Variable | Type | Format | Description |
| --- | --- | --- | --- |
| TONR tag | FBD_TIMER | structure | TONR structure |

### Function Block

| Operand | Type | Format | Description |
| --- | --- | --- | --- |
| TONR tag | FBD_TIMER | structure | TONR structure |

### FBD_TIMER Structure

| Input Parameter | Data Type | Description |
| --- | --- | --- |
| EnableIn | BOOL | **Function Block:**<br><br>If cleared, the instruction does not execute and outputs are not updated.<br><br>If set, the instruction executes.<br><br>Default is set.<br><br>**Structured Text:**<br><br>No effect. The instruction executes. |
| TimerEnable | BOOL | If set, this enables the timer to run and accumulate time.<br><br>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.<br><br>Valid = 0 to maximum positive integer |
| Reset | BOOL | Request to reset the timer. When set, the timer resets.<br><br>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 ≥ PRE

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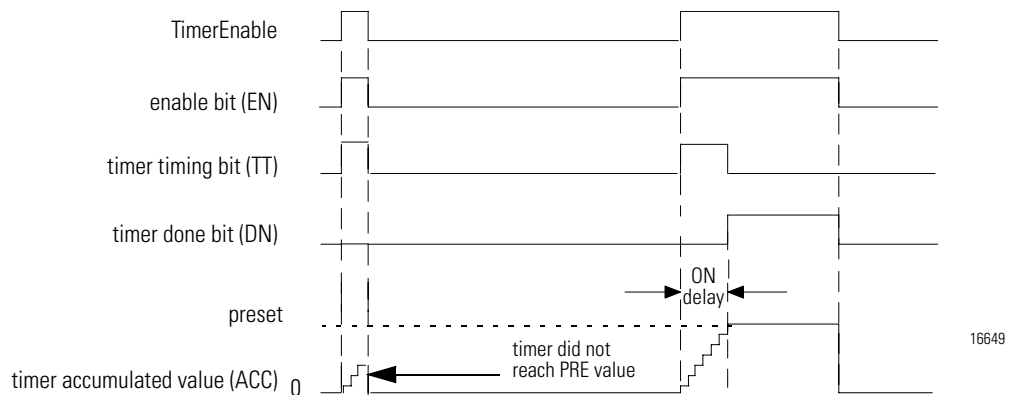


Set the Reset input parameter to reset the instruction. If TimerEnable is set when Reset is set, the TONR instruction begins timing again when Reset is cleared.

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

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

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. | EN, TT and DN are cleared. |
| | ACC value is set to 0. | ACC value is set to 0. |
| instruction first run | EN, TT and DN are cleared. | EN, TT and DN are cleared. |
| | ACC value is set to 0. | 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. | EnableIn is always set. |
| | | The instruction executes. |
| | The instruction executes. | |
| | EnableOut is set. | |
| reset | When the Reset input parameter is set, the instruction clears EN, TT and DN and sets ACC = zero. | When the Reset input parameter is set, the instruction clears EN, TT and DN and sets ACC = zero. |
| postscan | No action taken. | No action taken. |

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

### Structured Text

```
TONR_01.Preset := 500;

TONR_01.Reset : = reset;

TONR_O1.TimerEnable := limit_switch1;


TONR(TONR_01);


timer_state := TONR_01.DN;
```

### Function Block Example
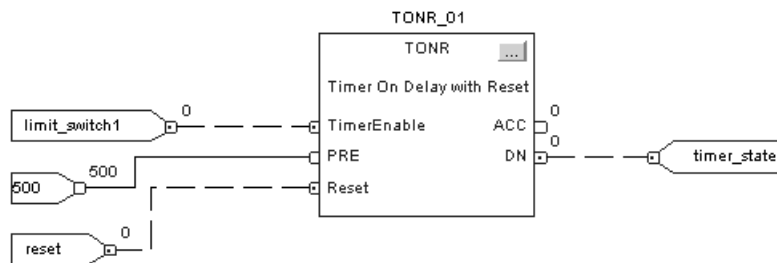
# Timer Off Delay with Reset (TOFR)

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

This instruction is available in relay ladder as two separate instructions: TOF (see ) and RES (see ).

## Operands:

`TOFR(TOFR_tag);`

### Structured Text

| Variable | Type | Format | Description |
|---|---|---|---|
| TOFR tag | FBD_TIMER | structure | TOFR structure |

### Function Block Operands

| Operand | Type | Format | Description |
|---|---|---|---|
| TOFR tag | FBD_TIMER | structure | TOFR structure |

### FBD_TIMER Structure

| Input Parameter | Data Type | Description |
|---|---|---|
| EnableIn | BOOL | **Function Block:** |
| | | If cleared, the instruction does not execute and outputs are not updated. |
| | | If set, the instruction executes. |
| | | Default is set. |
| | | **Structured Text:** |
| | | No effect. The instruction executes. |
| TimerEnable | BOOL | If cleared, this enables the timer to run and accumulate time. |
| | | Default is cleared. |
| PRE | DINT | Timer preset value. This is the value in 1msec units that ACC must reach before timing is finished. If invalid, the instructions sets the appropriate bit in Status and the timer does not execute. |
| | | Valid = 0 to maximum positive integer |
| Reset | BOOL | Request to reset the timer. When set, the timer resets. |
| | | Default is cleared. |

| Output Parameter | Data Type | Description |
|---|---|---|
| EnableOut | BOOL | The instruction produced a valid result. |
| ACC | BOOL | Accumulated time in milliseconds. |
| EN | BOOL | Timer enabled output. Indicates the timer instruction is enabled. |

| 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 ≥ PRE

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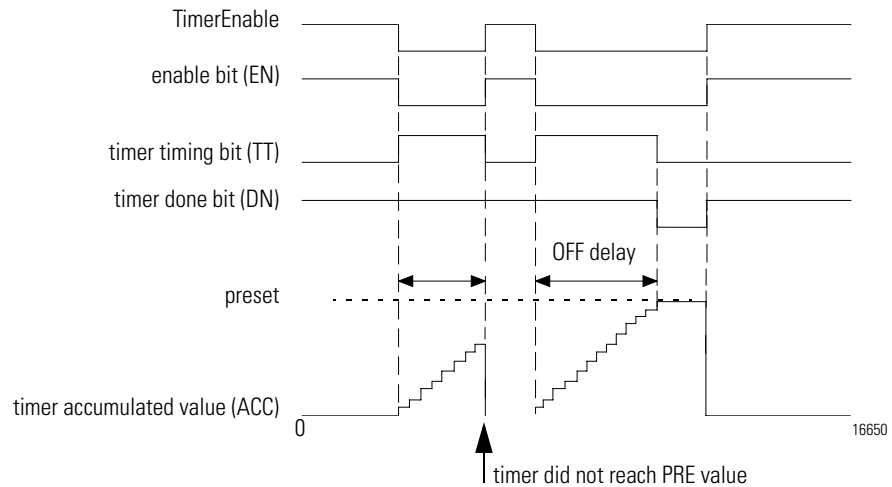


Set the Reset input parameter to reset the instruction. If TimerEnable is cleared when Reset is set, the TOFR instruction does not begin timing again when Reset is cleared.

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.<br><br>ACC value is set to PRE. | EN, TT and DN are cleared.<br><br>ACC value is set to PRE. |
| instruction first run | EN, TT and DN are cleared.<br><br>ACC value is set to PRE. | EN, TT and DN are cleared.<br><br>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.<br><br>The instruction executes.<br><br>EnableOut is set. | EnableIn is always set.<br><br>The instruction executes. |
| reset | When the Reset input parameter is set, the instruction clears EN, TT and DN and sets ACC = PRE. Note that this is different than using a RES instruction on a TOF instruction. | When the Reset input parameter is set, the instruction clears EN, TT and DN and sets ACC = PRE. Note that this is different than using a RES instruction on a TOF instruction. |
| postscan | No action taken. | No action taken. |

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

## Structured Text

```
TOFR_01.Preset := 500

TOFR_01.Reset := reset;

TOFR_O1.TimerEnable := limit_switch1;


TOFR(TOFR_01);


timer_state2 := TOFR_01.DN;
```

## Function Block
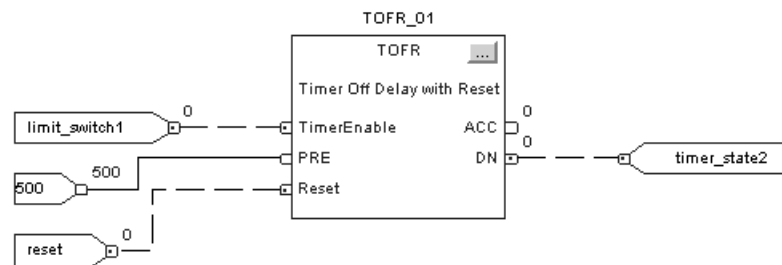
# Retentive Timer On with Reset (RTOR)

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

This instruction is available in relay ladder as two separate instructions: RTO (see ) and RES (see ).

## Operands:

```
RTOR(RTOR_tag);
```

## Structured Text

| Variable | Type | | Format | Description |
|----------|------|---|--------|-------------|
| RTOR tag | FBD_TIMER | | structure | RTOR structure |

## Function Block Operands

| Operand | Type | | Format | Description |
|---------|------|---|--------|-------------|
| RTOR tag | FBD_TIMER | | structure | RTOR structure |

## FBD_TIMER Structure

| Input Parameter | Data Type | Description |
|-----------------|-----------|-------------|
| EnableIn | BOOL | **Function Block:**<br><br>If cleared, the instruction does not execute and outputs are not updated.<br><br>If set, the instruction executes.<br><br>Default is set.<br><br>**Structured Text:**<br><br>No effect. The instruction executes. |
| TimerEnable | BOOL | If set, this enables the timer to run and accumulate time.<br><br>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.<br><br>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.
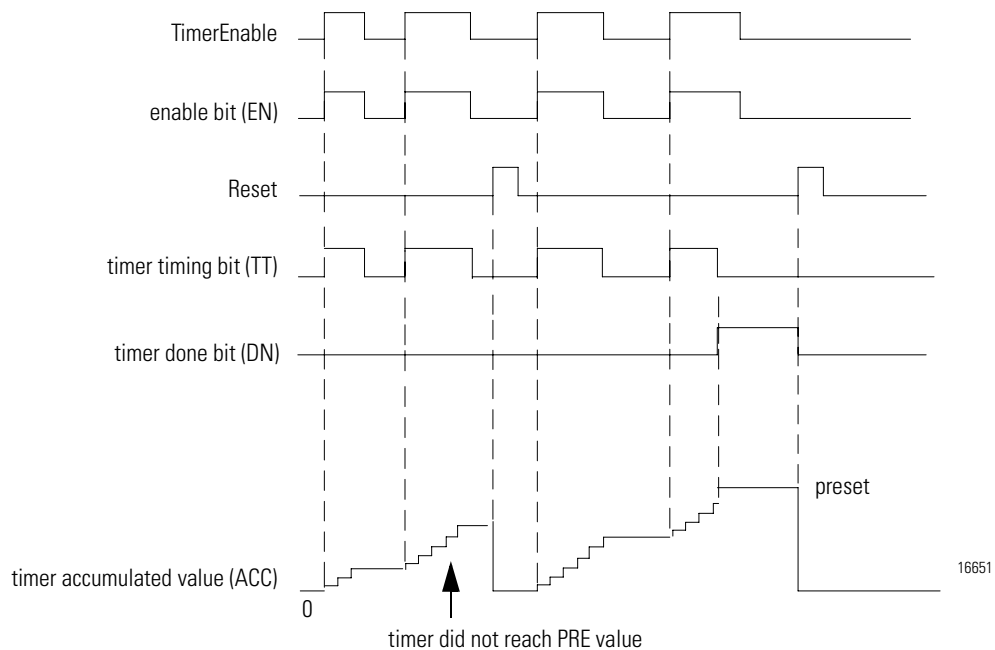


timer did not reach PRE value

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

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

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

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

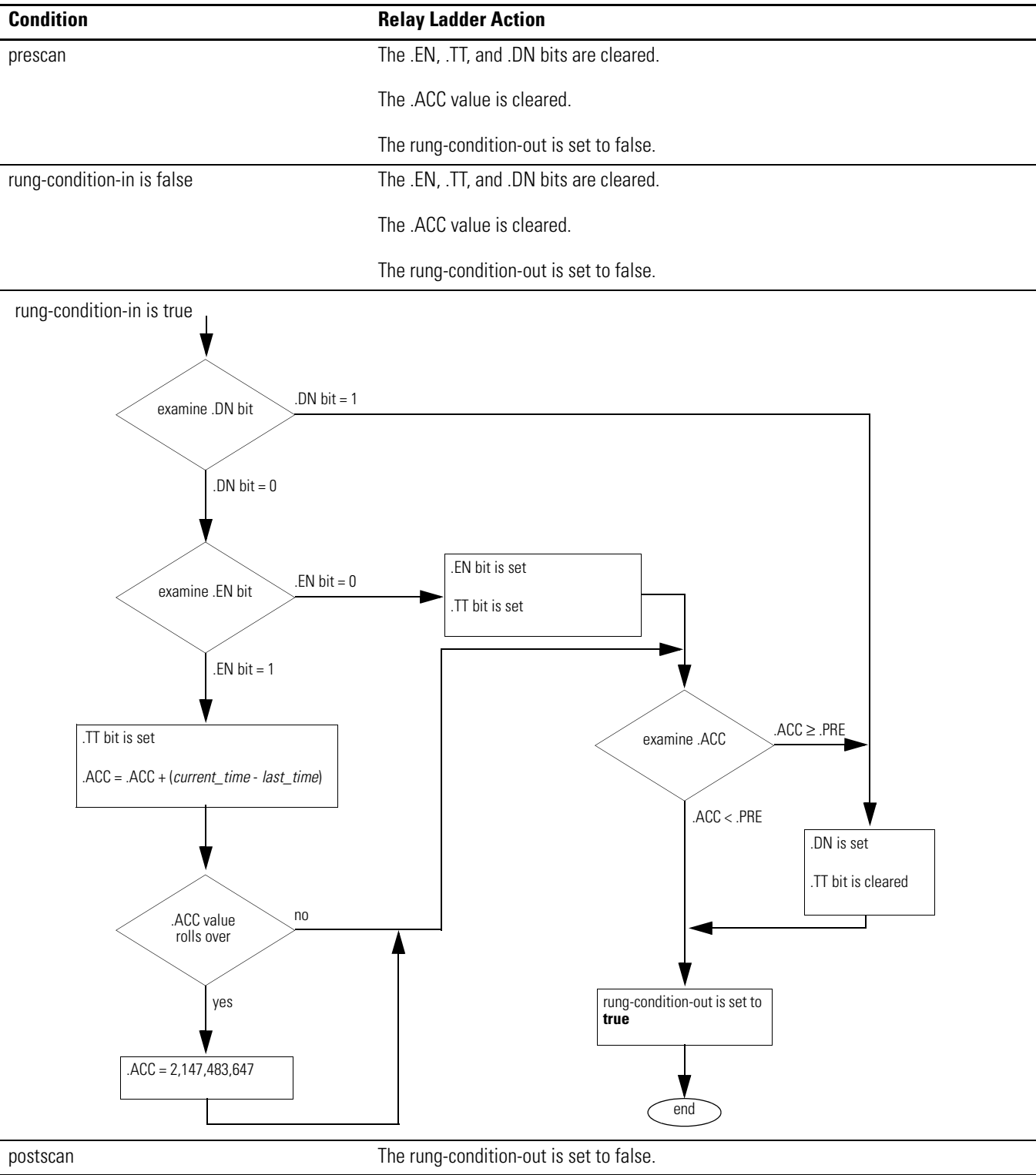| **IMPORTANT** | Make sure to scan the timer at least every 69 minutes while it runs. Otherwise, the ACC value won't be correct. |
|---|---|
| | The *last_time_scanned* value has a range of up to 69 minutes. The timer's calculation rolls over if you don't scan the timer within 69 minutes. The ACC value won't be correct if this happens. |
| | While a timer runs, scan it within 69 minutes if you put it in a: |
| | • subroutine<br>• section of code that is between JMP and LBL instructions<br>• sequential function chart (SFC)<br>• event or periodic task<br>• 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 | EN, TT and DN are cleared |
| | ACC value is not modified | ACC value is not modified |
| instruction first run | EN, TT and DN are cleared | EN, TT and DN are cleared |
| | ACC value is not modified | ACC value is not modified |
| EnableIn is cleared | EnableOut is cleared, the instruction does nothing, and the outputs are not updated. | na |
| EnableIn is set | **Function Block:** | EnableIn is always set. |
| | When EnableIn transitions from cleared to set, the instruction initializes as described for instruction first scan. | The instruction executes. |
| | The instruction executes. | |
| | EnableOut is set. | |
| reset | When the Reset input parameter is set, the instruction clears EN, TT and DN and sets ACC = zero. | When the Reset input parameter is set, the instruction clears EN, TT and DN and sets ACC = zero. |
| postscan | No action taken. | No action taken. |

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

### Structured Text

```
RTOR_01.Preset := 500

RTOR_01.Reset := reset;

RTOR_O1.TimerEnable := limit_switch1;


RTOR(RTOR_01);


timer_state3 := RTOR_01.DN;
```

### Function Block
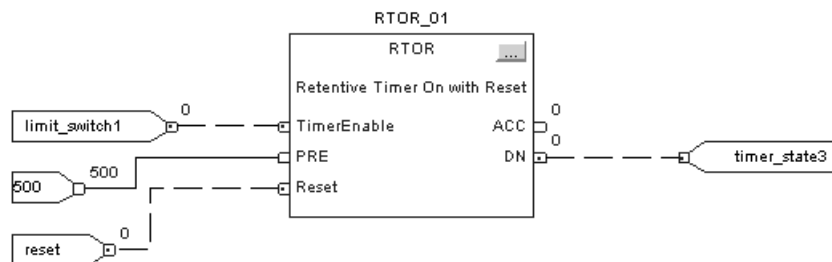
# Count Up (CTU)

The CTU instruction counts upward.

This instruction is available in structured text and function block as CTUD, see .
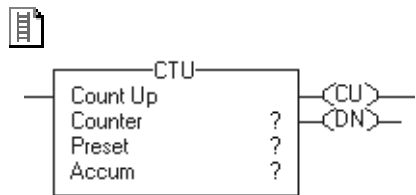
### Operands:

### 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<br><br>initial value is typically 0 |

### COUNTER Structure

| Mnemonic | Data Type | Description |
|----------|-----------|-------------|
| .CU | BOOL | The count up enable bit indicates that the CTU instruction is enabled. |
| .DN | BOOL | The done bit indicates that .ACC ≥ .PRE. |
| .OV | BOOL | The overflow bit indicates that the counter exceeded the upper limit of 2,147,483,647. The counter then rolls over to -2,147,483,648 and begins counting up again. |
| .UN | BOOL | The underflow bit indicates that the counter exceeded the lower limit of -2,147,483,648. The counter then rolls over to 2,147,483,647 and begins counting down again. |
| .PRE | DINT | The preset value specifies the value 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.
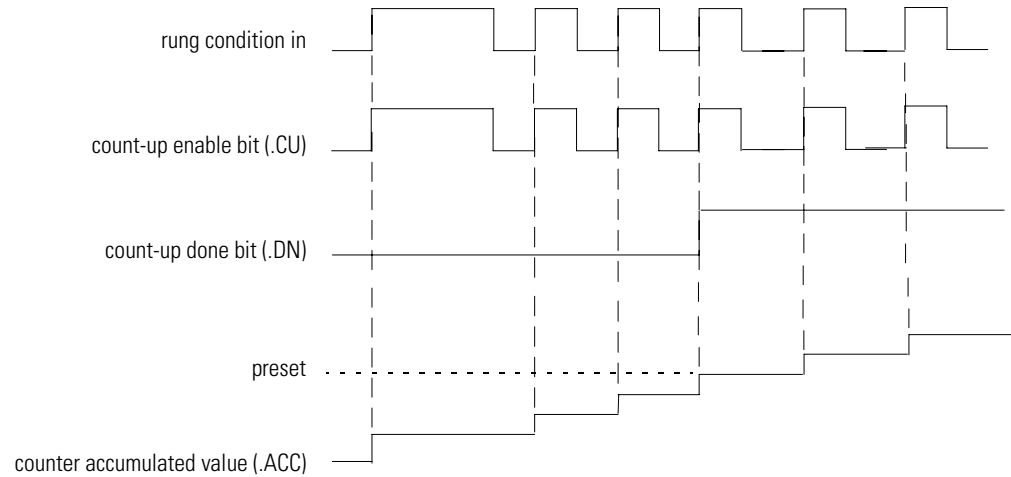


16636

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:

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

rung-condition-in is true



| postscan | The rung-condition-out is set to false. |
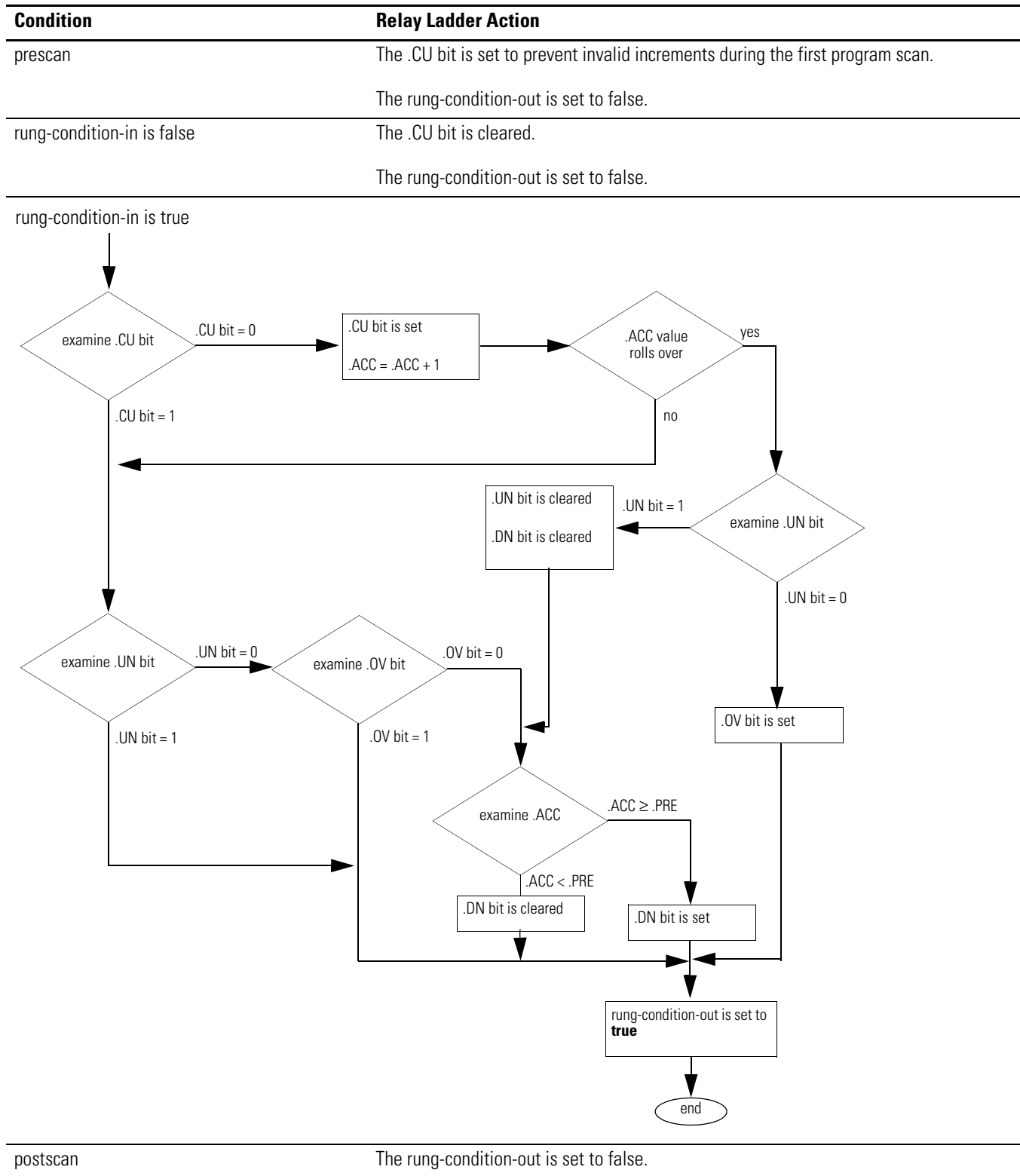
**Example:**    After *limit_switch_1* goes from disabled to enabled 10 times, the .DN bit is set and *light_1* turns on. If *limit_switch_1* continues to go from disabled to enabled, *counter_1* continues to increment its count and the .DN bit remains set. When limit_switch_2 is enabled, the RES instruction resets *counter_1* (clears the status bits and the .ACC value) and *light_1* turns off.
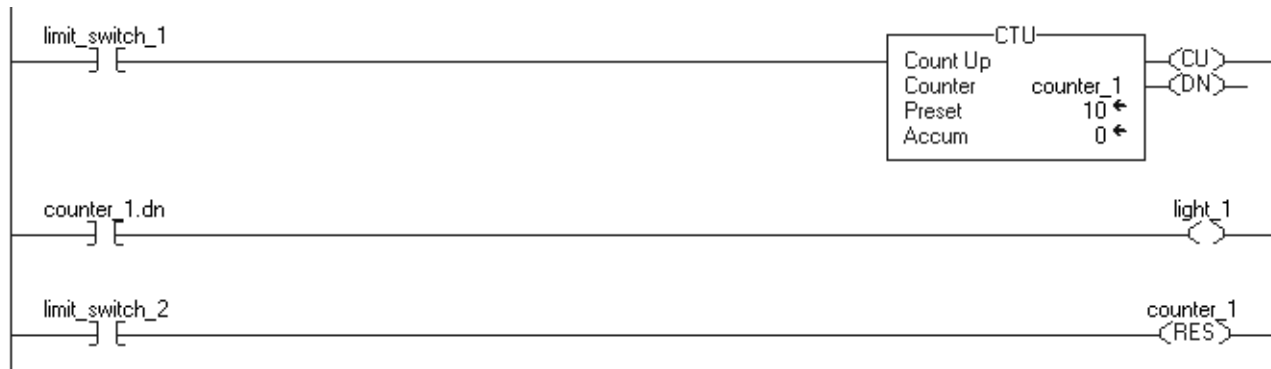
# Count Down (CTD)

The CTD instruction counts downward.

This instruction is available in structured text and function block as CTUD, see .
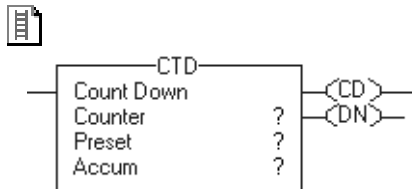
### Operands:

### Relay Ladder

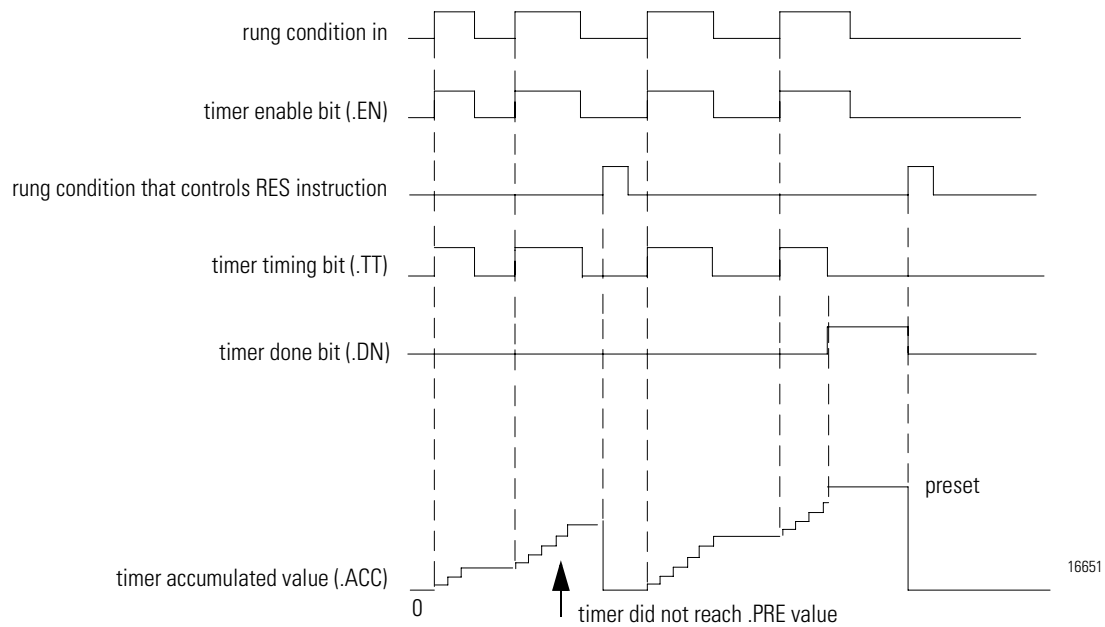| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| Counter | COUNTER | tag | counter structure |
| Preset | DINT | immediate | how low to count |
| Accum | DINT | immediate | number of times the counter has counted<br><br>initial value is typically 0 |

### COUNTER Structure

| Mnemonic | Data Type | Description |
|----------|-----------|-------------|
| .CD | BOOL | The count down enable bit indicates that the CTD instruction is enabled. |
| .DN | BOOL | The done bit indicates that .ACC ≥ .PRE. |
| .OV | BOOL | The overflow bit indicates that the counter exceeded the upper limit of 2,147,483,647. The counter then rolls over to -2,147,483,648 and begins counting up again. |
| .UN | BOOL | The underflow bit indicates that the counter exceeded the lower limit of -2,147,483,648. The counter then rolls over to 2,147,483,647 and begins counting down again. |
| .PRE | DINT | The preset value specifies the value 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.



16637
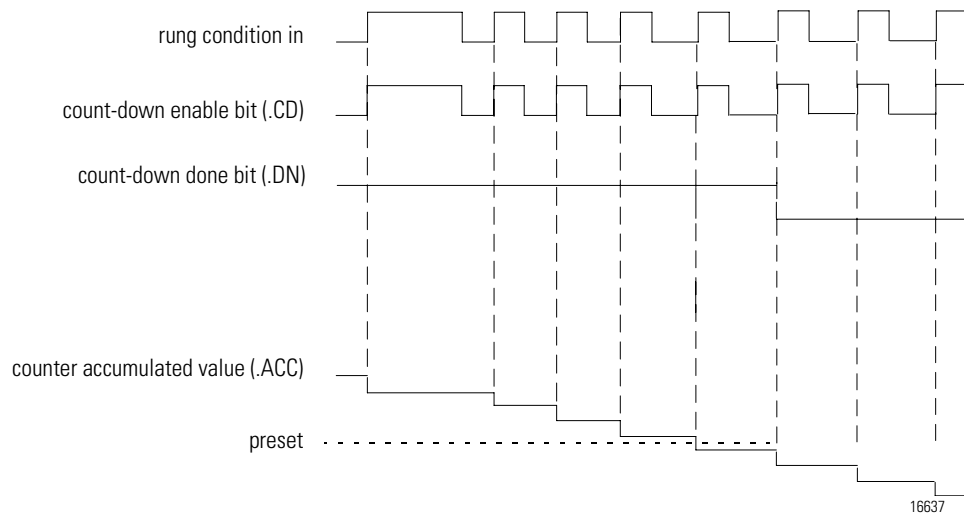
The accumulated value continues decrementing, even after the .DN bit is set. To clear the accumulated value, use a RES instruction that references the counter structure or write 0 to the accumulated value.

**Arithmetic Status Flags:**   not affected

**Fault Conditions:**   none

### Execution:

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

rung-condition-in is true
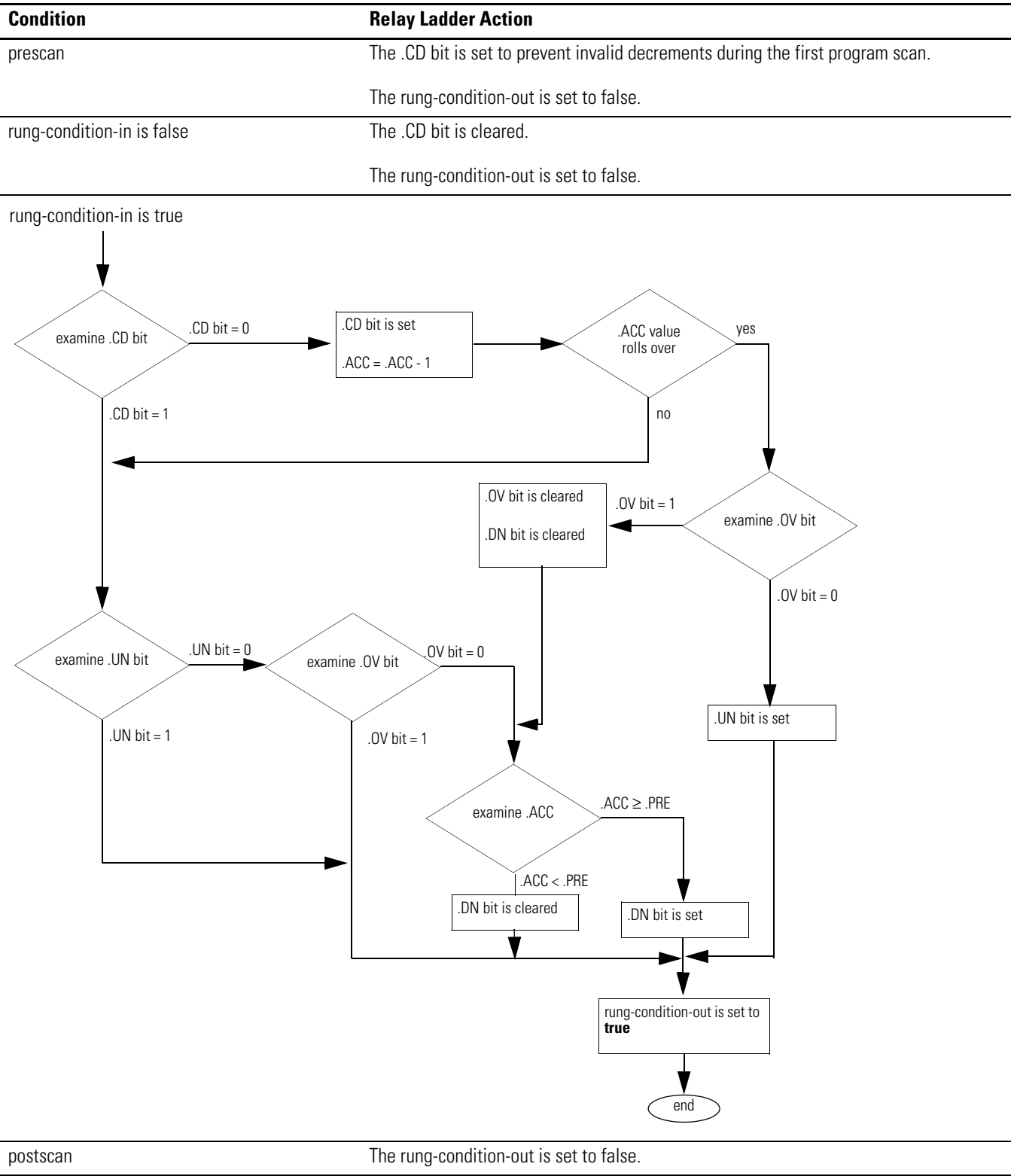


| postscan | The rung-condition-out is set to false. |
|---|---|

**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.
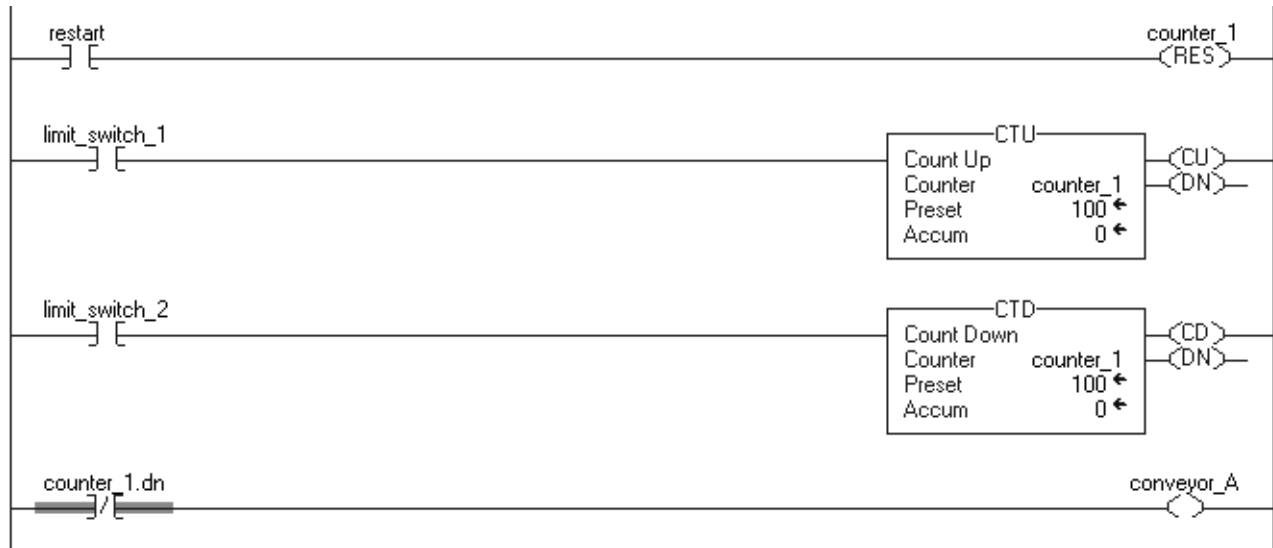
# Count Up/Down (CTUD)

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 128), CTD (see page 132), and RES (see page 141).

## Operands:

CTUD(CTUD_tag);

### Structured Text

| Variable | Type | Format | Description |
|----------|------|--------|-------------|
| CTUD tag | FBD_COUNTER | structure | CTUD structure |

### Function Block

CTUD_01

CTUD [...]

Count Up/Down
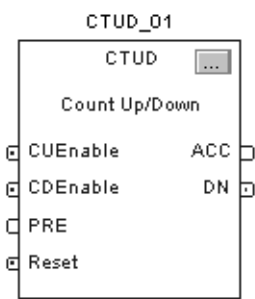
| CUEnable | ACC |
| CDEnable | DN |
| PRE | |
| Reset | |

| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| CTUD tag | FBD_COUNTER | structure | CTUD structure |

### FBD_COUNTER Structure

| Input Parameter | Data Type | Description |
|-----------------|-----------|-------------|
| EnableIn | BOOL | **Function Block:** |
| | | If cleared, the instruction does not execute and outputs are not updated. |
| | | If set, the instruction executes. |
| | | Default is set. |
| | | **Structured Text:** |
| | | No effect. The instruction executes. |
| CUEnable | BOOL | Enable up count. When input toggles from clear to set, accumulator counts up by one. |
| | | Default is cleared. |

| 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*



16636

*Counting Down*



16637
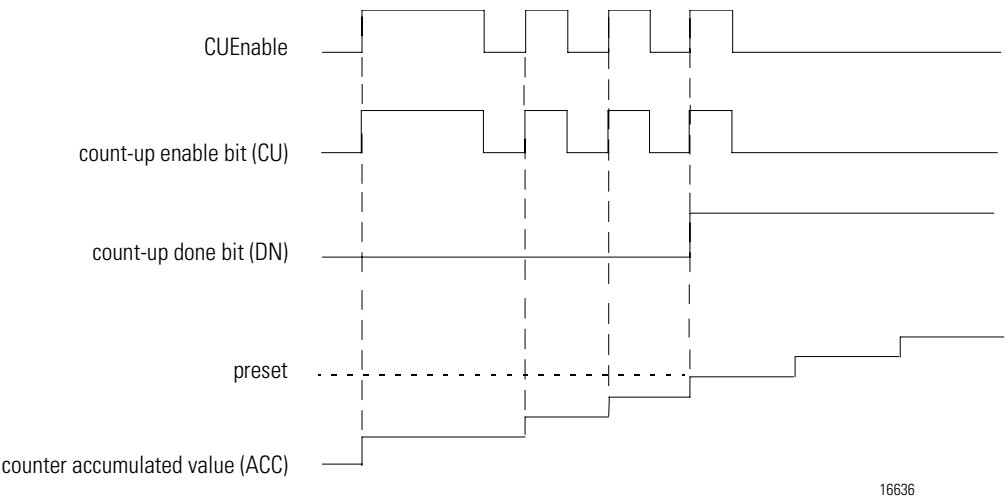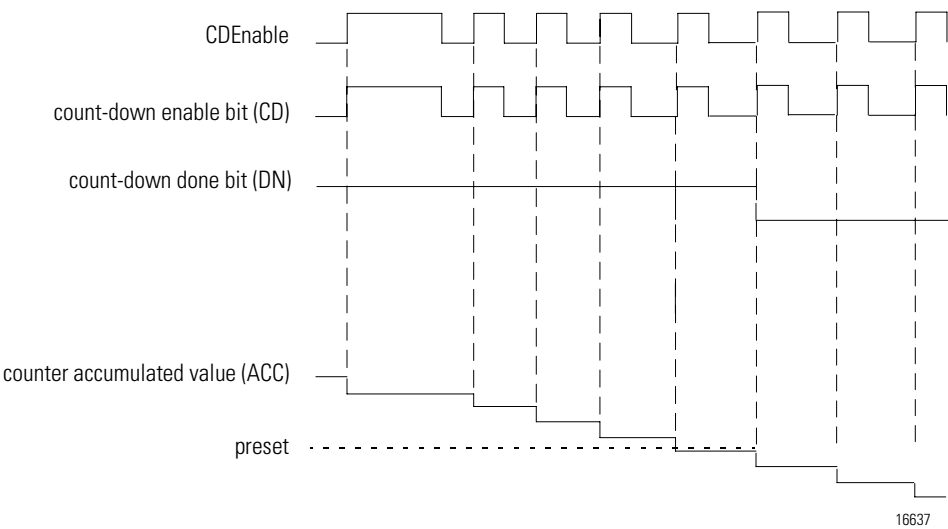
When disabled, the CTUD instruction retains its accumulated value. Set the Reset input parameter of the FBD_COUNTER structure to reset the instruction.

**Arithmetic Status Flags:**    not affected

**Fault Conditions:**    none

**Execution:**

| Condition | Function Block Action | Structured Text Action |
|---|---|---|
| prescan | No initialization required. | No initialization required. |
| instruction first scan | $CUEnable_{n-1}$ and $CDEnable_{n-1}$ are set. | $CUEnable_{n-1}$ and $CDEnable_{n-1}$ are set. |
| instruction first run | $CUEnable_{n-1}$ and $CDEnable_{n-1}$ are set. | $CUEnable_{n-1}$ and $CDEnable_{n-1}$ are set. |
| EnableIn is cleared | EnableOut is cleared, the instruction does nothing, and the outputs are not updated. | na |
| EnableIn is set | The instruction sets $CUEnable_{n-1}$ and $CDEnable_{n-1}$. On a cleared to set transition of EnableIn: <br>• The instruction executes. <br>• EnableOut is set. | The instruction sets $CUEnable_{n-1}$ and $CDEnable_{n-1}$. EnableIn is always set. The instruction executes. |
| reset | When set, the instruction clears $CUEnable_{n-1}$, $CDEnable_{n-1}$, CU, CD, DN, OV, and UN and sets ACC = zero. | When set, the instruction clears $CUEnable_{n-1}$, $CDEnable_{n-1}$, CU, CD, DN, OV, and UN and sets ACC = zero. |
| postscan | No action taken. | No action taken. |

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

**Structured Text**

```
CTUD_01.Preset := 500;

CTUD_01.Reset := Restart;

CTUD_O1.CUEnable := limit_switch1;


CTUD(CTUD_01);


counter_state := CTUD_01.DN;
```

**Function Block**
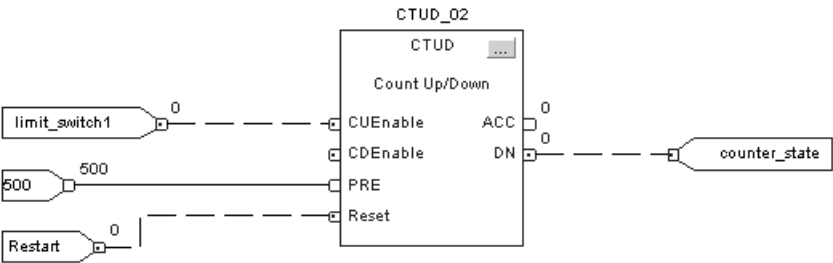
# Reset (RES)

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

## Operands:

📄

### Relay Ladder

—(RES)—
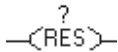
| Operand | Type | Format | Description |
|---|---|---|---|
| structure | TIMER | tag | structure to reset |
| | CONTROL | | |
| | COUNTER | | |

**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. |

| Condition | Relay Ladder Action |
|---|---|
| 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.<br><br>The rung-condition-out is set to true. |
| postscan | The rung-condition-out is set to false. |

# 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 | 144 |
| | | structured text | |
| get controller status information | GSV | relay ladder | 176 |
| | | structured text | |
| set controller status information | SSV | relay ladder | 176 |
| | | structured text | |
| • send output values to an I/O module or consuming controller at a specific point in your logic<br>• trigger an event task in another controller | IOT | relay ladder<br><br>structured text | 201 |

# Message (MSG)

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

### Operands:

### Relay Ladder



```
                  MSG
  Message                    (EN)
  Message Control    ? ... + (DN)
                            (ER)
```

| 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. |

| This bit: | Is this member: |
|---|---|
| 2 | .EW |
| 4 | .ER |
| 5 | .DN |
| 6 | .ST |
| 7 | .EN |
| 8 | .TO |
| 9 | .EN_CC |

**Important:** 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.

| Mnemonic | Data Type | Description |
|---|---|---|
| .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. **Important:** 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. **Important:** 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. **Important:** 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. **Important:** 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. **Important:** 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 4-173 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. |

| If the message: | Then the local array is the: |
|---|---|
| reads data | Destination element |
| writes data | Source element |

| Mnemonic | Data Type | Description |
|---|---|---|
| .Channel | SINT | To send the message out a different channel of the 1756-DHRIO module, set this member to the required value. Use either the ASCII character A or B. |
| .Rack | SINT | To change the rack number for a block transfer message, set this member to the required rack number (octal). |
| .Group | SINT | To change the group number for a block transfer message, set this member to the required group number (octal). |
| .Slot | SINT | To change the slot number for a block transfer message, set this member to the required slot number. |

| If the message goes over this network: | Then specify the slot number in: |
|---|---|
| universal remote I/O | octal |
| ControlNet | decimal (0-15) |

| Mnemonic | Data Type | Description |
|---|---|---|
| .Path | STRING | To send the message to a different controller, set this member to the new path.<br><br>• Enter the path as hexadecimal values.<br>• Omit commas [,]<br><br>For example, for a path of 1, 0, 2, 42, 1, 3, enter $01$00$02$2A$01$03.<br><br>To browse to a device and automatically create a portion or all of the new string, right-click a string tag and choose *Go to Message Path Editor*. |
| .RemoteIndex | DINT | If you use an asterisk [*] to designate the element number of the remote array, the RemoteIndex provides the element number. To change the element number, set this member to the required value. |

| If the message: | Then the remote array is the: |
|---|---|
| reads data | Source element |
| writes data | Destination element |

| Mnemonic | Data Type | Description |
|---|---|---|
| .RemoteElement | STRING | To specify a different tag or address in the controller to which the message is sent, set this member to the required value. Enter the tag or address as ASCII characters. |

| If the message: | Then the remote array is the: |
|---|---|
| reads data | Source element |
| writes data | Destination element |

| Mnemonic | Data Type | Description |
|---|---|---|
| .UnconnnectedTimeout | DINT | Time out for an unconnected message or for making a connection. The default value is 30 seconds. |

| If the message is | Then |
|---|---|
| unconnected | The ER bit turns on if the controller doesn't get a response within the UnconnectedTimeout time. |
| connected | The ER bit turns on if the controller doesn't get a response for making the connection within the UnconnectedTimeout time. |

| Mnemonic | Data Type | Description |
|---|---|---|
| .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 | |

- This time out applies only after the connection is made.
- The time out = ConnectionRate x TimeoutMultiplier.
- The default ConnectionRate is 7.5 seconds.
- The default TimeoutMultiplier is 0 (which is a multiplication factor of 4).
- The default time out for connected messages is 30 seconds (7.5 seconds x 4 = 30 seconds).
- To change the time out, change the ConnectionRate and leave the TimeoutMultiplier at the default value.

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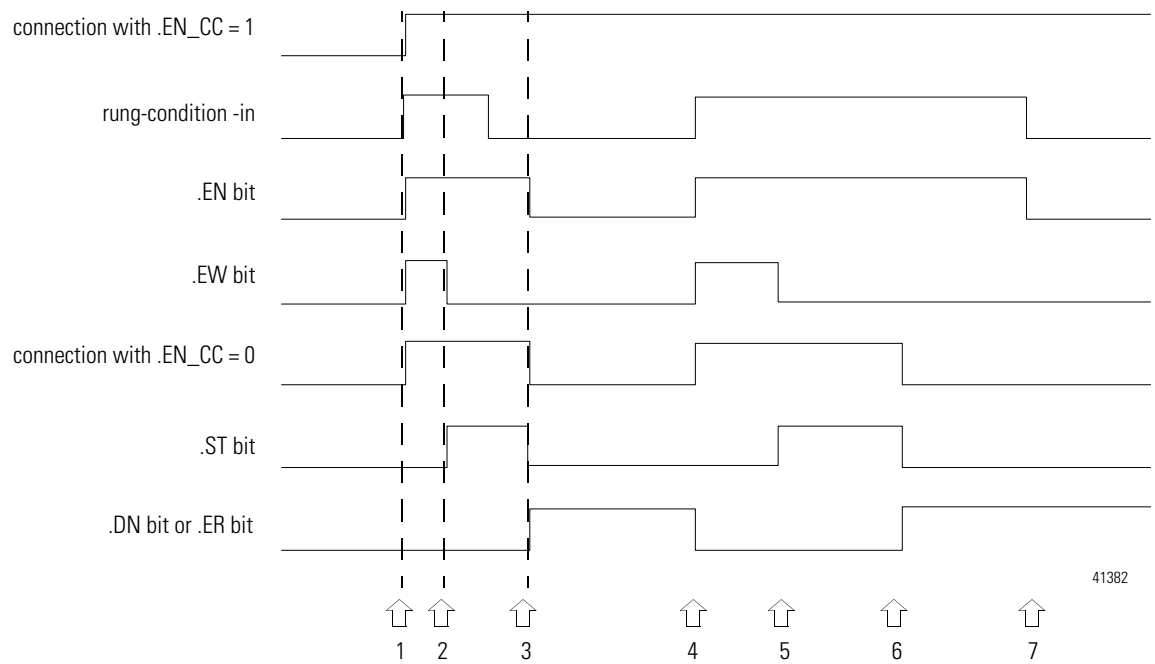
| Where | Description | Where | Description |
|---|---|---|---|
| 1 | rung-condition-in is true<br><br>.EN is set<br><br>.EW is set<br><br>connection is opened* | 5 | message is sent<br><br>.ST is set<br><br>.EW is cleared |
| 2 | message is sent<br><br>.ST is set<br><br>.EW is cleared | 6 | message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

```
                                    .EN bit = 1
                                   ┌────────── examine .EN bit ──────────┐
                                   ▼                          .EN bit = 0 │
             .EW bit = 1                                                  │
          ┌────────── examine .EW bit                                     │
          │              │ .EW bit = 0                                    │
          │              ▼                                                │
   .ST bit = 1                                                            │
   ◄────────── examine .ST bit                                            │
          │       │ .ST bit = 0                                           │
          │       ▼                                                       │
   .DN bit =                                                              │
   ◄────────── examine .DN bit                                            │
          │       │ .DN bit = 0                                           │
          │       ▼                                                       │
   .ER bit = 1           .ER bit = 0      block-transfer   yes   module path valid   no
   ◄────────── examine .ER bit ──────────► command  ──────────► module path valid ──────┐
          │                                    │ no                 │ yes               │
          ▼                                    │                    ▼                   │
                        .DN bit = 1            │           module          no           │
   examine .DN bit ──────────────┐             │           connection ──────────────────┤
          │                      │             │           running                      │
          │ .DN bit = 0          ▼             │              │ yes                      │
          ▼              .EN bit is cleared    ▼       ◄───────┘                         │
                                        execute message request                         │
                .ER bit = 1                    ▼                    ▼                     │
   examine .ER bit ──────────┐          .EW bit is set      .ER bit is set               │
          │                  │                  │                  │                     │
          │ .ER bit = 0      ▼                  ▼                  ▼                     │
          └──────────────────┴──────────────────┴──────────────────┴─────────────────────┘
                                                │
                                      rung-condition-out is set
                                      to **false**
                                                │
                                                ▼
                                             ( end )
```

| Condition | Relay Ladder Action | Structured Text Action |
|---|---|---|
| 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
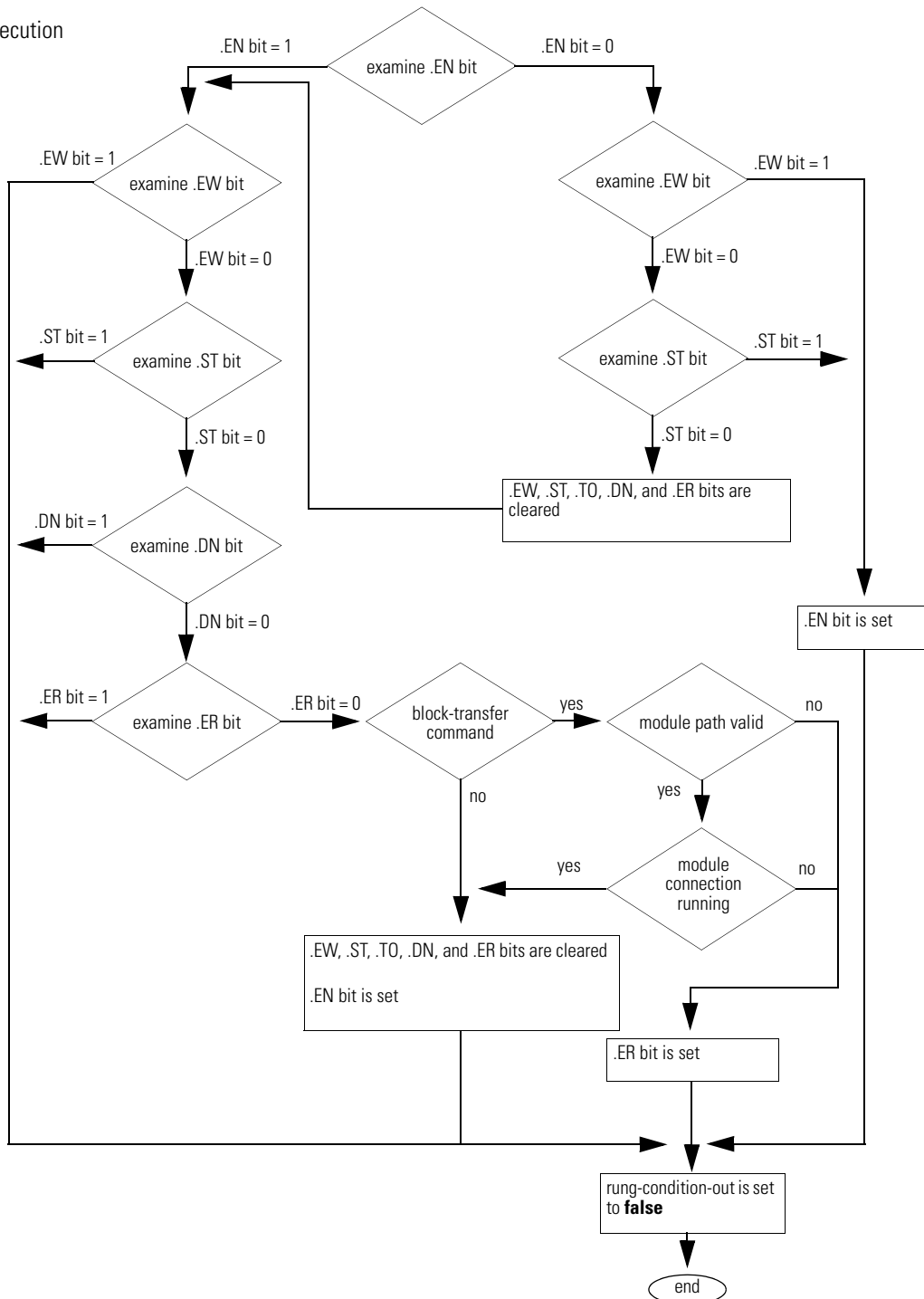


| postscan | The rung-condition-out is set to false. | No action taken. |

**Arithmetic Status Flags:**  not affected

**Fault Conditions:**  none

# MSG Error Codes

The error codes depend on the type of MSG instruction.

## Error Codes

RSLogix 5000 software does not always display the full description.

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

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

## Extended Error Codes

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

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

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

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

| Extended Error Code (Hex) | Description |
|---|---|
| 0203 | Connection timeout |

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

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

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

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

# PLC and SLC Error Codes (.ERR)

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

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

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

**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<br><br>Situation changed during multipacket operation |
| 00F0 | 0009 | F009 | | Data or file is too large<br><br>Memory unavailable |

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

| R9.x And Earlier | | R10.x And Later | | Description |
|---|---|---|---|---|
| .ERR | .EXERR | .ERR | .EXERR | |
| 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<br><br>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<br><br>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.



Click here to configure the MSG instruction
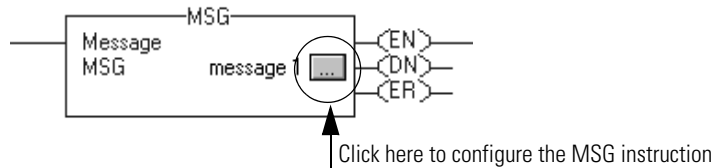
The details you configure depend on the message type you select.



42976

| If The Target Device Is a | Select One Of These Message Types | See Page |
|---|---|---|
| Logix5000 controller | CIP Data Table Read | 160 |
| | CIP Data Table Write | |
| I/O module that you configure using RSLogix 5000 software | Module Reconfigure | 161 |
| | CIP Generic | 162 |
| PLC-5 controller | PLC5 Typed Read | 163 |
| | PLC5 Typed Write | |
| | PLC5 Word Range Read | |
| | PLC5 Word Range Write | |
| SLC controller | SLC Typed Read | 165 |
| MicroLogix controller | SLC Typed Write | |
| Block-transfer module | Block-Transfer Read | 165 |
| | Block-Transfer Write | |
| PLC-3 processor | PLC3 typed read | 166 |
| | PLC3 typed write | |
| | PLC3 word range read | |
| | PLC3 word range write | |
| PLC-2 processor | PLC2 unprotected read | 167 |
| | PLC2 unprotected write | |

You must specify this configuration information.

| For This Property | Specify |
|---|---|
| Source Element | • If you select a read message type, the Source Element is the address of the data you want to read in the target device. Use the addressing syntax of the target device.<br><br>• If you select a write message type, the Source Tag is the first element of the tag that you want to send to the target device. |
| Number of Elements | The number of elements you read/write depends on the type of data you are using.  An element refers to one "chunk" of related data.  For example, tag *timer1* is one element that consists of one timer control structure. |
| Destination Element | • If you select a read message type, the Destination Element is the first element of the tag in the Logix5000 controller where you want to store the data you read from the target device.<br><br>• If you select a write message type, the Destination Element is the address of the location in the target device where you want to write the data. |

## Specify CIP Data Table Read and Write messages

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

| Select This Command | If You Want To |
|---|---|
| CIP Data Table Read | read data from another controller.<br><br>The Source and Destination types must match. |
| CIP Data Table Write | write data to another controller.<br><br>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
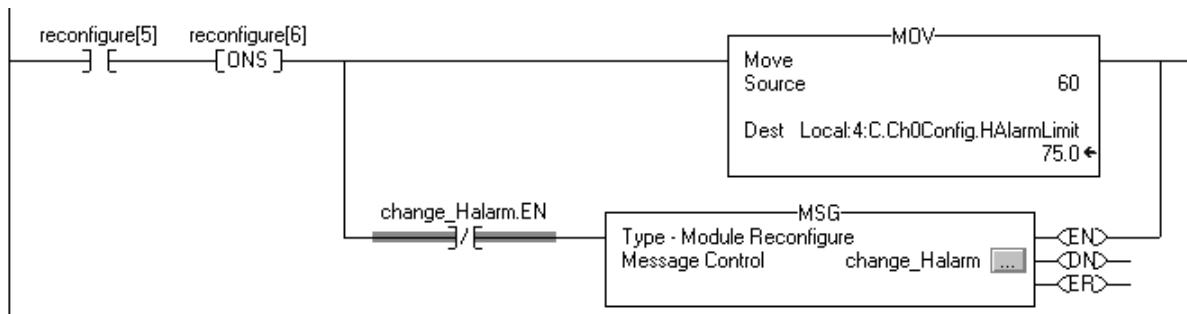


### Structured Text

```
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 | *tag_name* of type INT [5] | |
| | | This array contains: | |
| | | *tag_name*[0] | bit mask of points to test (test only one point at a time) |
| | | *tag_name*[1] | reserved, leave 0 |
| | | *tag_name*[2] | pulse width (hundreds of μsecs, usually 20) |
| | | *tag_name*[3] | zero cross delay for ControlLogix I/O (hundreds of μsecs, usually 40) |
| | | *tag_name*[4] | verify delay |
| | Destination | leave blank | |
| Reset electronic fuses on a digital output module | Message Type | CIP Generic | |
| | Service Type | Reset Electronic Fuse | |
| | Source | *tag name* of type DINT | |
| | | This tag represents a bit mask of the points to reset fuses on. | |
| | Destination | leave blank | |
| Reset latched diagnostics on a digital input module | Message Type | CIP Generic | |
| | Service Type | Reset Latched Diagnostics (I) | |
| | Source | *tag_name* of type DINT | |
| | | This tag represents a bit mask of the points to reset diagnostics on. | |
| Reset latched diagnostics on a digital output module | Message Type | CIP Generic | |
| | Service Type | Reset Latched Diagnostics (O) | |
| | Source | *tag_name* of type DINT | |
| | | This tag represents a bit mask of the points to reset diagnostics on. | |

| 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:<br><br>• Unlatch All Alarms (I)<br>• Unlatch Analog High Alarm (I)<br>• Unlatch Analog High High Alarm (I)<br>• Unlatch Analog Low Alarm (I)<br>• Unlatch Analog Low Low Alarm (I)<br>• Unlatch Rate Alarm (I) |
| | Instance | Channel of the alarm that you want to unlatch |
| Unlatch the alarm of an analog output module | Message Type | CIP Generic |
| | Service Type | Select which alarm that you want to unlatch:<br><br>• Unlatch All Alarms (O)<br>• Unlatch High Alarm (O)<br>• Unlatch Low Alarm (O)<br>• Unlatch Ramp Alarm (O) |
| | 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 164. |
| 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 164 |
| PLC5 Word Range Read | Read a contiguous range of 16-bit words in PLC-5 memory regardless of data type.<br><br>This command starts at the address specified as the Source Element and reads sequentially the number of 16-bit words requested.<br><br>The data from the Source Element is stored, starting at the address specified as the Destination Tag. |
| PLC5 Word Range Write | Write a contiguous range of 16-bit words from Logix5000 memory regardless of data type to PLC-5 memory.<br><br>This command starts at the address specified as the Source Tag and reads sequentially the number of 16-bit words requested.<br><br>The data from the Source Tag is stored, starting at the address specified as the Destination Element in the PLC-5 processor. |

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

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

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

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

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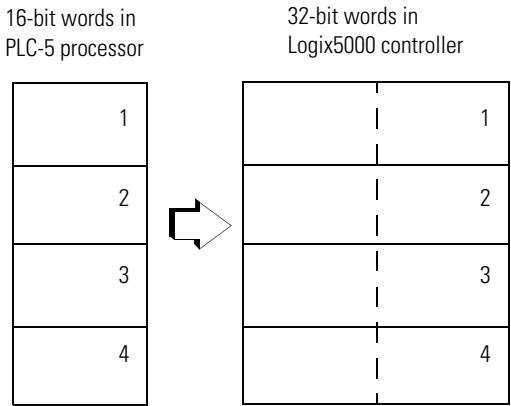
**Typed read command**

16-bit words in
PLC-5 processor

32-bit words in
Logix5000 controller



The typed commands maintain data structure and value.

**Word-range read command**

16-bit words in
PLC-5 processor

32-bit words in
Logix5000 controller
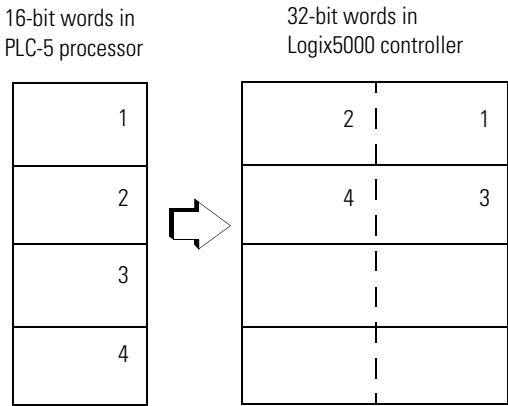


The word-range commands fill the destination tag contiguously. Data structure and value change depending on the destination data type.

## 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.<br><br>This message type replaces the BTR instruction. | Block-Transfer Read |
| write data to a block-transfer module.<br><br>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 | read integer or REAL type data. |
| | For integers, this command reads 16-bit integers from the PLC-3 processor and stores them in SINT, INT, or DINT data arrays in the Logix5000 controller and maintains data integrity. |
| | This command also reads floating-point data from the PLC-3 and stores it in a REAL data type tag in the Logix5000 controller. |
| PLC3 Typed Write | write integer or REAL type data. |
| | This command writes SINT or INT data, to the PLC-3 integer file and maintains data integrity.  You can write DINT data as long as it fits within an INT data type (-32,768 ≥ data ≤32,767). |
| | This command also writes REAL type data from the Logix5000 controller to a PLC-3 floating-point file. |
| PLC3 Word Range Read | read a contiguous range of 16-bit words in PLC-3 memory regardless of data type. |
| | This command starts at the address specified as the Source Element and reads sequentially the number of 16-bit words requested. |
| | The data from the Source Element is stored, starting at the address specified as the Destination Tag. |
| PLC3 Word Range Write | write a contiguous range of 16-bit words from Logix5000 memory regardless of data type to PLC-3 memory. |
| | This command starts at the address specified as the Source Tag and reads sequentially the number of 16-bit words requested. |
| | The data from the Source Tag is stored, starting at the address specified as the Destination Element in the PLC-3 processor. |

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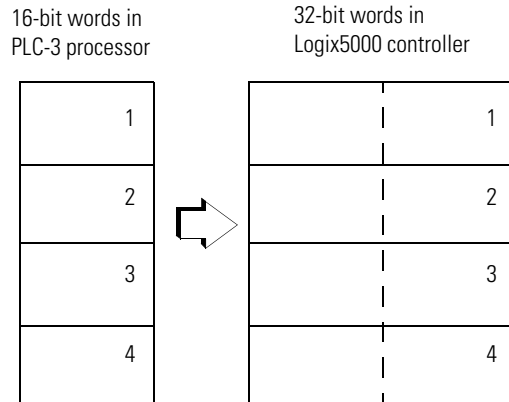
**Typed read command**

16-bit words in
PLC-3 processor

32-bit words in
Logix5000 controller



The typed commands maintain data structure and value.

**Word-range read command**

16-bit words in
PLC-3 processor

32-bit words in
Logix5000 controller
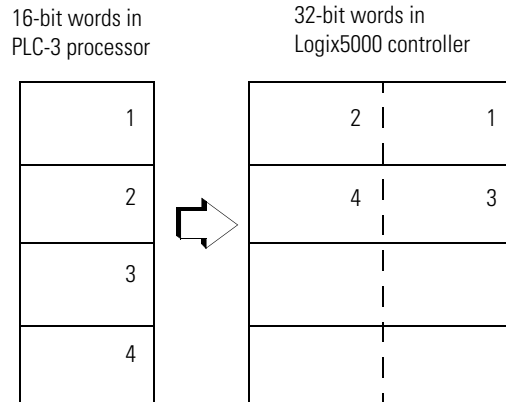


The word-range commands fill the destination tag contiguously. Data structure and value change depending on the destination data type.

## Specify PLC-2 messages

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

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

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

## MSG Configuration Examples

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

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

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

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

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

# Specify the Communication Details

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



Specify a path

Specify a Communication Method Or Module Address

Choose a cache option
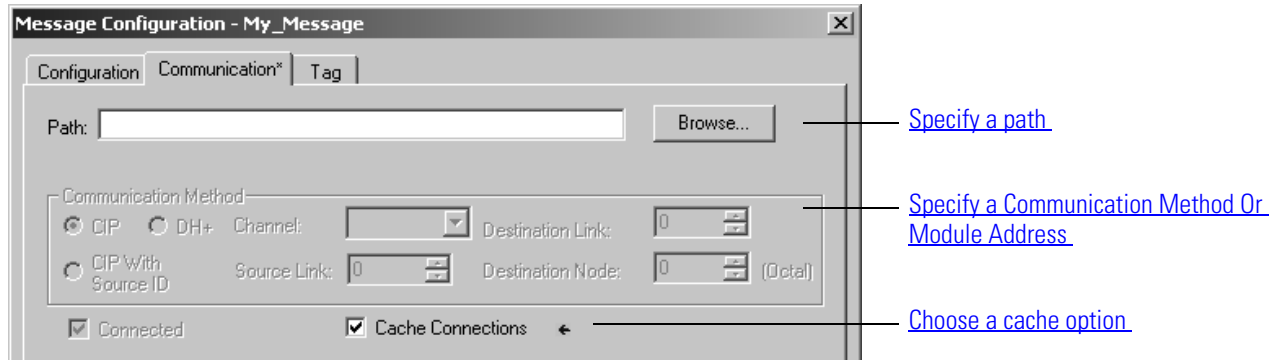
## Specify a path

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 *Browse* button to select the module. |
| The I/O configuration of the controller has only the local communication module. | 1. Use the *Browse* button to select the local communication module.<br>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 | Browse...

Peer_Controller

---

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 | Browse...

LocalENB, 2, 10.10.10.10, 1, 0

---

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 | Browse...

1, 1, 2, 4, 1, 0

---

To type a path, use this format:

p*ort, next_address,* p*ort, next_address, …*

| Where | Is | |
|---|---|---|
| | **For this network** | **Type** |
| *port* | backplane | 1 |
| | DF1 (serial, serial channel 0) | 2 |
| | ControlNet | |
| | EtherNet/IP | |
| | DH+ channel A | |
| | DH+ channel B | 3 |
| | DF1 channel 1 (serial channel 1) | |
| *next_address* | backplane | slot number of the module |
| | DF1 (serial) | station address (0-254) |
| | ControlNet | node number (1-99 decimal) |
| | DH+ | 8# followed by the node number (1-77 octal) |
| | | For example, to specify the octal node address of 37, type 8#37. |
| | EtherNet/IP | You can specify a module on an EtherNet/IP network 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 | • local communication module (for example, 1756-CNB module)<br>• remote adapter module (for example, 1771-ACN module) |
| universal remote I/O | • local communication module (for example, 1756-DHRIO module)<br>• one emote adapter module (for example, 1771-ASB module) for each rack, or portion of a rack, in the chassis<br>• block-transfer module (optional) |

## Specify a Communication Method Or Module Address
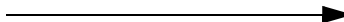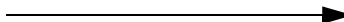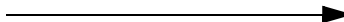
Use the following table to select a communication method or module address for the message.

| If The Destination Device Is 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. | |

| If The Destination Device Is a | Then Select | And Specify | |
|---|---|---|---|
| Application on a workstation that is receiving an unsolicited message routed over an EtherNet/IP or ControlNet network through RSLinx | **CIP with Source ID**<br><br>(This lets the application receive data from a controller.) | Source Link: | Remote ID of the topic in RSLinx software |
| | | Destination Link: | Virtual Link ID set up in RSLinx (0-65535) |
| | | Destination Node: | Destination ID (0-77 octal) provided by the application to RSLinx. For a DDE topic in RSLinx, use 77. |
| | | The slot number of the ControlLogix controller is used as the Source Node. | |
| block transfer module over a universal remote I/O network | RIO | Channel: | Channel A or B of the 1756-DHRIO module that is connected to the RIO network |
| | | Rack | Rack number (octal) of the module |
| | | Group | Group number of the module |
| | | Slot | Slot number that the module is in |
| block transfer module over a ControlNet network | ControlNet | Slot | Slot number that the module is in |

## Choose a cache option

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

| This Type Of Message | And This Communication Method | Uses A Connection |
|---|---|---|
| CIP data table read or write | ⟶ | ✓ |
| PLC2, PLC3, PLC5, or SLC (all types) | CIP | |
| | CIP with Source ID | |
| | DH+ | ✓ |
| CIP generic | ⟶ | your option [1] |
| block-transfer read or write | ⟶ | ✓ |

[1]  You can connect CIP generic messages. But for most applications we recommend you leave CIP generic messages unconnected.
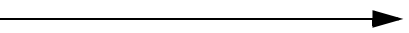
If a MSG instruction uses a connection, you have the option to leave the connection open (cache) or close the connection when the message is done transmitting.

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

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

| If You Have This Software And Firmware Revision | Then You Can Cache |
|---|---|
| 11.x or earlier | • block transfer messages for up to 16 connections<br>• other types of messages for up to 16 connections |
| 12.x or later | up to 32 connections |

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

| IF THE MSG Instructions Are To | And They Are | Then |
|---|---|---|
| different devices | ⟶ | Each MSG instruction uses 1 connection. |
| same device | enabled at the same time | Each MSG instruction uses 1 connection. |
|  | NOT enabled at the same time | The MSG instructions share the connection. (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.

# 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.<br>• Data type = MESSAGE<br>• Scope = controller<br>• The tag *cannot* be part of an array or a user-defined data type. |
| 2. Keep the source and/or destination data at the controller scope. | A MSG instruction can access only tags that are in the Controller Tags folder (controller scope). |
| 3. If your MSG is to a device that uses 16-bit integers, use a buffer of INTs in the MSG and DINTs throughout the project. | If your message is to a device that uses 16-bit integers, such as a PLC-5® or SLC 500™ controller, and it transfers integers (not REALs), use a buffer of INTs in the message and DINTs throughout the project.<br><br>This increases the efficiency of your project because Logix controllers execute more efficiently and use less memory when working with 32-bit integers (DINTs).<br><br>To convert between INTs and DINTs, see *Logix5000 Controllers Common Procedures*, publication 1756-PM001. |
| 4. Cache the connected MSGs that execute most frequently. | Cache the connection for those MSG instructions that execute most frequently, up to the maximum number permissible for your controller revision.<br><br>This optimizes execution time because the controller does not have to open a connection each time the message executes. |
| 5. If you want to enable more than 16 MSGs at one time, use some type of management strategy. | If you enable more than 16 MSGs at one time, some MSG instructions may experience delays in entering the queue. To guarantee the execution of each message, use one of these options:<br>• Enable each message in sequence.<br>• Enable the messages in groups.<br>• Program a message to communicate with multiple devices. For more information, see *Logix5000 Controllers Common Procedures*, publication 1756-PM001.<br>• Program logic to coordinate the execution of messages. For more information, see *Logix5000 Controllers Common Procedures*, publication 1756-PM001. |
| 6. Keep the number of unconnected and uncached MSGs less than the number of unconnected buffers. | The controller can have 10 - 40 unconnected buffers. The default number is 10.<br>• If all the unconnected buffers are in use when an instruction leaves the message queue, the instruction errors and does not transfer the data.<br>• You can increase the number of unconnected buffers (40 max.), but continue to follow guideline 5.<br>• To increase the number of unconnected buffers, see *Logix5000 Controllers Common Procedures*, publication 1756-PM001. |

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

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

## Operands:

### Relay Ladder

```
        ─GSV─                              ─SSV─
   Get System Value                   Set System Value
   Class name          ?              Class name          ?
   Instance name       ?              Instance name       ?
   Attribute Name      ?              Attribute Name      ?
   Dest                ?              Source              ?
                      ??                                 ??
```
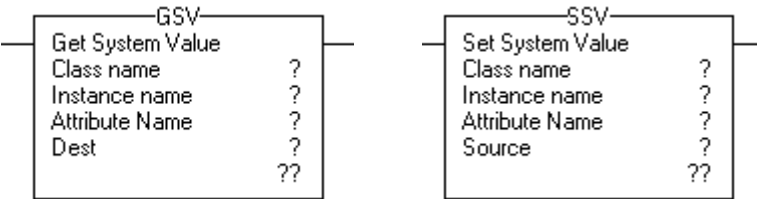
| Operand | Type | Format | Description |
|---|---|---|---|
| Class name | | name | name of object |
| Instance name | | name | name of specific object, when object requires name |
| Attribute Name | | name | attribute of object |
| | | | data type depends on the attribute you select |
| Destination (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**
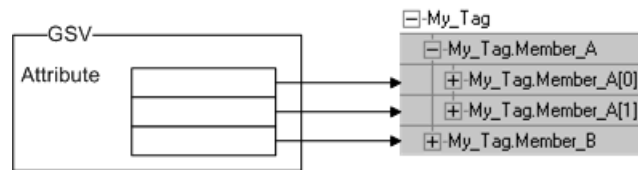


Member_A is too small for the attribute. So the GSV instruction writes the last value to Member_B.

**Example 2**
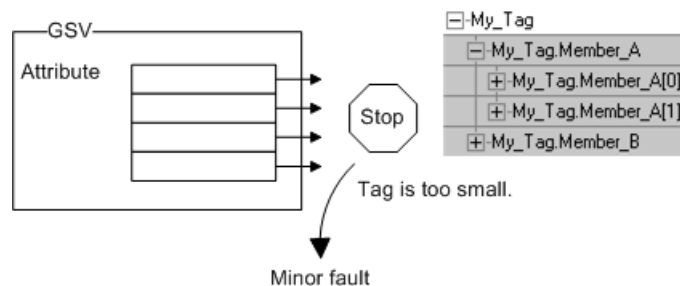


My_Tag is too small for the attribute. So the GSV instruction stops and logs a minor fault.

---

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.

<table>
<tr>
<td>**ATTENTION**<br></td>
<td>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.</td>
</tr>
</table>

You can access these objects:

| For Information About This Object | See This Page Or Publication |
| --- | --- |
| AXIS | *ControlLogix Motion Module Setup and Configuration Manual*, publication 1756-UM006 |
| CONTROLLER | 180 |
| CONTROLLERDEVICE | 181 |
| CST | 183 |
| DF1 | 184 |
| FAULTLOG | 187 |
| MESSAGE | 188 |
| MODULE | 190 |
| MOTIONGROUP | 191 |
| PROGRAM | 192 |
| ROUTINE | 193 |
| SERIALPORT | 193 |
| TASK | 195 |
| WALLCLOCKTIME | 197 |

# Access the CONTROLLER object

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

| Attribute | Data Type | Instruction | Description |
|---|---|---|---|
| TimeSlice | INT | GSV<br><br>SSV | Percentage of available CPU that is assigned to communications.<br><br>Valid values are 10-90. This value cannot be changed when the controller keyswitch is in the run position. |
| ControllerLogTotalEntryCount | DINT | SSV<br><br>GSV | Number of controller log entries since the last firmware upgrade.<br><br>The number will be reset if RAM enters a bad state.<br><br>The number is capped at the largest DINT. |
| ControllerLogExecutionModificationCount | DINT | SSV<br><br>GSV | Number of controller log entries that originated from a program/task properties change, an online edit, or a controller timeslice change. It can also be configured to include log entries originating from forces.<br><br>The number will be reset if RAM enters a bad state.<br><br>The number is not capped at the largest DINT, and a rollover can occur. |
| ControllerLogUnsavedEntryCount | DINT | GSV | Number of entries in the controller log that have yet to be stored to removable media.<br><br>Range from 0 to maximum number of entries. |
| ControllerLogAutoWrite | BOOL | MSG | Flag used to determine if the automatic write of the controller log to removable media is enabled.<br><br>0 = auto write is disabled (fault).<br><br>1 = controller log will attempt to write to removable media when the log is 80% full. |
| ExecutionCountConfigureMask | DINT | MSG | Bit array used to determine what will cause the Modify Execution Count to increment.<br><br>0 = default (everything but forces).<br><br>1 = forces included (everything and forces). |

# 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 |

| 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. |

| Attribute | Data Type | Instruction | Description |
|---|---|---|---|
| Status | INT | GSV | Bits identify status: |

Bits 3-0 are reserved

**Device Status Bits**

| Bits 7-4: | Meaning: |
|---|---|
| 0000 | reserved |
| 0001 | flash update in progress |
| 0010 | reserved |
| 0011 | reserved |
| 0100 | flash is bad |
| 0101 | faulted |
| 0110 | run |
| 0111 | program |

**Fault Status Bits**

| Bits 11-8: | Meaning: |
|---|---|
| 0001 | recoverable minor fault |
| 0010 | unrecoverable minor fault |
| 0100 | recoverable major fault |
| 1000 | unrecoverable major fault |

**Logix5000 Specific Status Bits**

| Bits 13-12: | Meaning: |
|---|---|
| 01 | keyswitch in run |
| 10 | keyswitch in program |
| 11 | keyswitch in remote |

| Bits 15-14 | Meaning |
|---|---|
| 01 | controller is changing modes |
| 10 | debug mode if controller is in run mode |

| 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 |

## 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<br>01 = time master node<br>10 = time relay node<br>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.<br><br>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 | The amount of time to wait for an acknowledgment to a message transmission (point-to-point and master only). Valid value 0-32,767. Delay in counts of 20 msec periods. Default is 50 (1 second). |
| DiagnosticCounters | INT[19] | GSV | Array of diagnostic counters for the DF1 communication driver. |

| word offset | DF1 point-to-point | DF1 slave | master |
|---|---|---|---|
| 0 | signature (0x0043) | signature (0x0042) | signature (0x0044) |
| 1 | modem bits | modem bits | modem bits |
| 2 | packets sent | packets sent | packets sent |
| 3 | packets received | packets received | packets received |
| 4 | undelivered packets | undelivered packets | undelivered packets |
| 5 | unused | messages retried | messages retried |
| 6 | NAKs received | NAKs received | unused |
| 7 | ENQs received | poll packets received | unused |
| 8 | bad packets NAKed | bad packets not ACKed | bad packets not ACKed |
| 9 | no memory sent NAK | no memory not ACKed | unused |
| 10 | duplicate packets received | duplicate packets received | duplicate packets received |
| 11 | bad characters received | unused | unused |
| 12 | DCD recoveries count | DCD recoveries count | DCD recoveries count |
| 13 | lost modem count | lost modem count | lost modem count |
| 14 | unused | unused | priority scan time maximum |
| 15 | unused | unused | priority scan time last |
| 16 | unused | unused | normal scan time maximum |
| 17 | unused | unused | normal scant time last |
| 18 | ENQs sent | unused | unused |

| Attribute | Data Type | Instruction | Description |
|---|---|---|---|
| DuplicateDetection | SINT | GSV | Enables duplicate message detection. |

| Value: | Meaning: |
|---|---|
| 0 | duplicate message detection disabled |
| non zero | duplicate message detection disabled |

| Attribute | Data Type | Instruction | Description |
|---|---|---|---|
| EmbeddedResponseEnable | SINT | GSV | Enables embedded response functionality (point-to-point only). |

| Value: | Meaning: |
|---|---|
| 0 | initiated only after one is received (default) |
| 1 | enabled unconditionally |

| Attribute | Data Type | Instruction | Description |
|---|---|---|---|
| ENQTransmitLimit | SINT | GSV | The number of inquiries (ENQs) to send after an ACK timeout (point-to-point only). Valid value 0-127.  Default setting is 3. |
| EOTSuppression | SINT | GSV | Enable suppressing EOT transmissions in response to poll packets (slave only). |

| Value: | Meaning: |
|---|---|
| 0 | EOT suppression disabled (disabled) |
| non zero | EOT suppression enabled |

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

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

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

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 |
|---|---|
| *msg_1* | message whose attribute you want to change |
| *msg_1_type* | tag that stores the value of the MessageType attribute |
| *tag_a* | tag that stores a 0. |
| *msg_1_path* | array tag that stores the new connection path for the message |

**Relay Ladder**
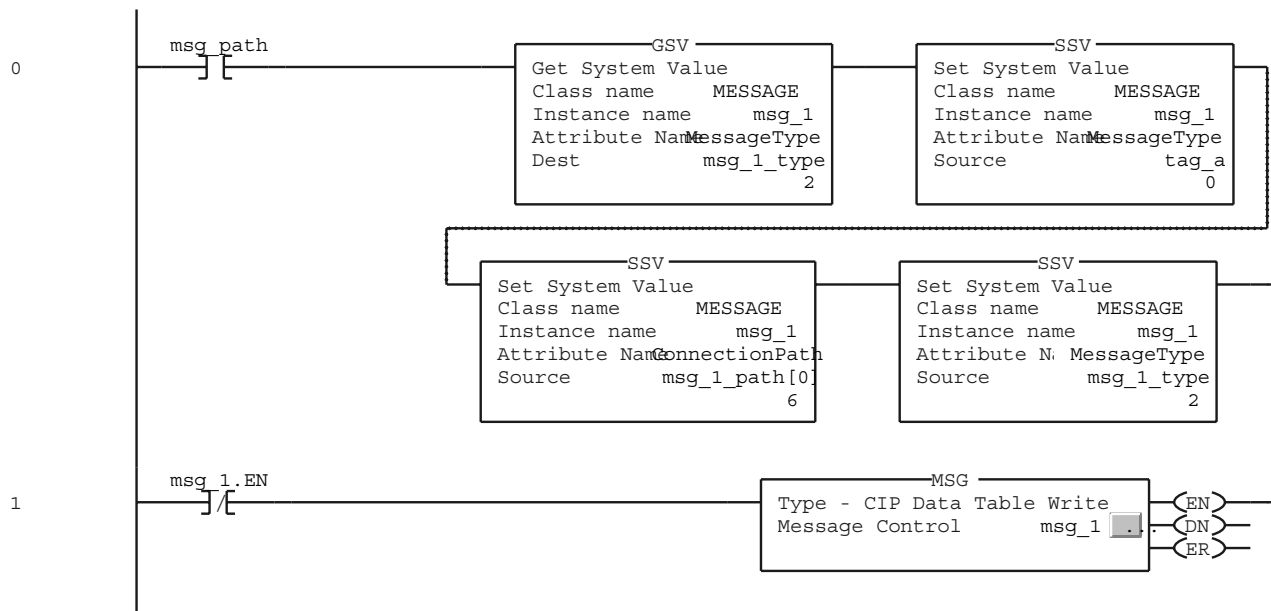


**Structured Text**

```
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. |
| | | | **Value:**  **Meaning:**<br>16#0000  **Standby:** the controller is powering up. |
| | | | 16#1000  **Faulted:** any of the MODULE object's connections to the associated module fail. This value should not be used to determine if the module failed because the MODULE object leaves this state periodically when trying to reconnect to the module. Instead, test for Running state (16#4000). Check for FaultCode not equal to 0 to determine if a module is faulted. When Faulted, the FaultCode and FaultInfo attributes are valid until the fault condition is corrected. |
| | | | 16#2000  **Validating:** the MODULE object is verifying MODULE object integrity prior to establishing connections to the module.<br>16#3000  **Connecting**: the MODULE object is initiating connections to the module. |
| | | | 16#4000  **Running:** all connections to the module are established and data is successfully transferring. |
| | | | 16#5000  **Shutting down:** the MODULE object is in the process of shutting down all connections to the module. |
| | | | 16#6000  **Inhibited:** the MODULE object is inhibited (the inhibit bit in the Mode attribute is set). |
| | | | 16#7000  **Waiting:** the parent MODULE object upon which this MODULE object depends is not running. |
| FaultCode | INT | 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. |
| | | | **Bit:**  **Meaning:**<br>0  forces installed (1=yes, 0-no)<br>1  forces enabled (1=yes, 0=no)<br>2-15  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. |

| | | | **Value:** | **Meaning:** |
|---|---|---|---|---|
| | | | 0 | **LED off:** No MODULE objects are configured for the controller (there are no modules in the I/O Configuration section of the controller organizer). |
| | | | 1 | Flashing red: None of the MODULE objects are Running. |
| | | | 2 | Flashing green: At least one MODULE object is not Running. |
| | | | 3 | Solid green: All the Module objects are Running. |

**Note:** You do not enter an object name with this attribute because this attribute applies to the entire collection of modules.

| Attribute | Data Type | Instruction | Description |
|---|---|---|---|
| Mode | INT | GSV | Specifies the current mode of the MODULE object. |

| | | | **Bit:** | **Meaning:** |
|---|---|---|---|---|
| | | SSV | 0 | If set, causes a major fault to be generated if any of the MODULE object connections fault while the controller is in Run mode. |
| | | | 2 | If set, causes the MODULE object to enter Inhibited state after shutting down all the connections to the module. |

## 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 | **Value:** **Meaning:**<br>0     execution enabled<br>1     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. Time is in microseconds. |
| | | SSV | |
| 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: |

| Name: | Data Type: | Style: | Description: |
|---|---|---|---|
| TimeLow | DINT | Decimal | lower 32 bits of fault timestamp value |
| TimeHigh | DINT | Decimal | upper 32 bits of fault timestamp value |
| Type | INT | Decimal | fault type (program, I/O, 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) |

| Attribute | Data Type | Instruction | Description |
|---|---|---|---|
| MaxScanTime | DINT | GSV | Maximum recorded execution time for this program.  Time is in microseconds. |
| | | SSV | |
| 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: |

| Name: | Data Type: | Style: | Description: |
|---|---|---|---|
| TimeLow | DINT | Decimal | lower 32 bits of fault timestamp value |
| TimeHigh | DINT | Decimal | upper 32 bits of fault timestamp value |
| Type | INT | Decimal | fault type (program, I/O, 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) |

| Attribute | Data Type | Instruction | Description |
|---|---|---|---|
| SFCRestart | INT | GSV | unused - reserved for future use |
| | | SSV | |

## 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.<br><br>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.<br><br>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.<br><br>**Value:** **Meaning:**<br>7 — 7 data bits (ASCII only)<br>8 — 8 data bits (default) |
| Parity | SINT | GSV | Specifies the parity.<br><br>**Value:** **Meaning:**<br>0 — no parity (no default)<br>1 — odd parity (ASCII only)<br>2 — even parity |
| RTSOffDelay | INT | GSV | Amount of time to delay turning off the RTS line after the last character has been transmitted.<br><br>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.<br><br>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.<br><br>**Value:** **Meaning:**<br>1 — 1 stop bit (default)<br>2 — 2 stop bits (ASCII only) |
| PendingBaudRate | DINT | SSV | Pending value for the BaudRate attribute. |

| Attribute | Data Type | Instruction | Description |
|---|---|---|---|
| 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 | **To:** / **Set the attribute to:** |
| | | | enable the processing of outputs at the end of the task / 0 |
| | | | disable the processing of outputs at the end of the task / 1 (or any non-zero value) |
| EnableTimeOut | DINT | GSV | Enables or disables the timeout function of an event task. |
| | | SSV | **To:** / **Set the attribute to:** |
| | | | disable the timeout function / 0 |
| | | | enable the timeout function / 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 | |
| | | | **To:** / **Set the attribute to:** |
| | | | enable the task / 0 (default) |
| | | | inhibit (disable) the task / 1 (or any non-zero value) |
| Instance | DINT | GSV | Provides the instance number of this TASK object. <br><br> 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. <br><br> A value of 0 indicates 1 or less executions of the task. |
| | | SSV | |
| 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. <br><br> A value of 0 indicates 1 or less executions of the task. |
| | | SSV | |
| OverlapCount | DINT | GSV | Number of times that the task was triggered while it was still executing. Valid for an event or a periodic task. <br><br> To clear the count, set the attribute to 0. |
| | | SSV | |

| Attribute | Data Type | Instruction | Description | |
|---|---|---|---|---|
| Priority | INT | GSV | Relative priority of this task as compared to the other tasks. | |
| | | SSV | Valid values 1…15. | |
| Rate | DINT | GSV | **If the task type is:** | **Then the Rate attribute specifies the:** |
| | | SSV | periodic | Period for the task. Time is in microseconds. |
| | | | event | The timeout value for the task. Time is in microseconds. |
| 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 | **To determine if:** | **Examine this bit:** |
| | | | An EVNT instruction triggered the task (event task only). | 0 |
| | | | A timeout triggered the task (event task only). | 1 |
| | | | An overlap occurred for this task. | 2 |
| 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: | |
| | | | **Time:** | **Task Type:** |
| | | | 0.5 sec | periodic or event |
| | | | 5.0 sec | 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<br><br>SSV | Positive offset from the CurrentValue of the CST object (coordinated system time, see page 183). DINT[0] contains the lower 32 bits of the value; DINT[1] contains the upper 32 bits of the value.<br><br>Value in µsecs. The default is 0. |
| CurrentValue | DINT[2] | GSV<br><br>SSV | Current value of the wall clock time.  DINT[0] contains the lower 32 bits of the value; DINT[1] contains the upper 32 bits of the value.<br><br>The value is the number of microseconds that have elapsed since 0000 hrs 1 January 1972.<br><br>The CST and WALLCLOCKTIME objects are mathematically related in the controller. For example, if you add the CST CurrentValue and the WALLCLOCKTIME CTSOffset, the result is the WALLCLOCKTIME CurrentValue. |
| DateTime | DINT[7] | GSV<br><br>SSV | The date and time in a readable format.<br><br>DINT[0]    year<br><br>DINT[1]    integer representation of month (1-12)<br><br>DINT[2]    integer representation of day (1-31)<br><br>DINT[3]    hour (0-23)<br><br>DINT[4]    minute (0-59)<br><br>DINT[5]    seconds (0-59)<br><br>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
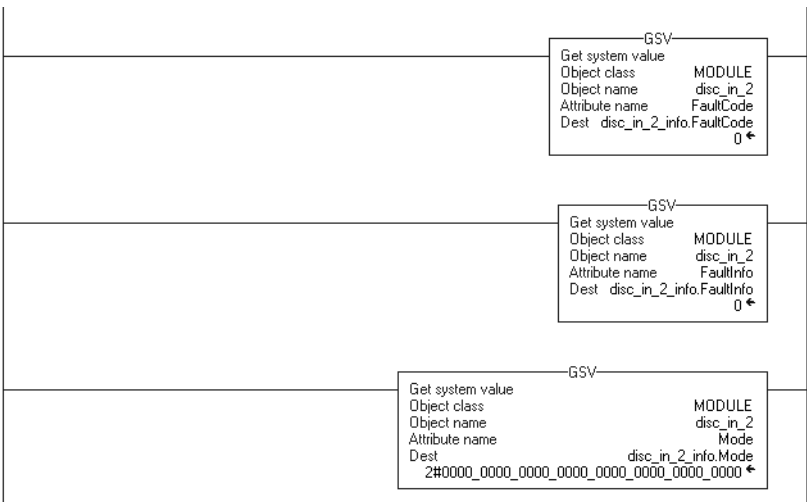


### Structured Text

```
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
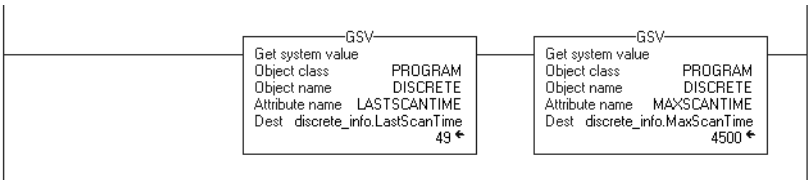


### Structured Text

```
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
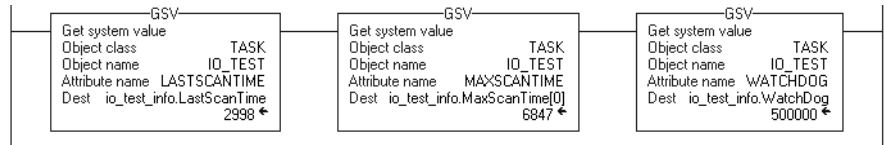


## Structured Text

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);

## 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
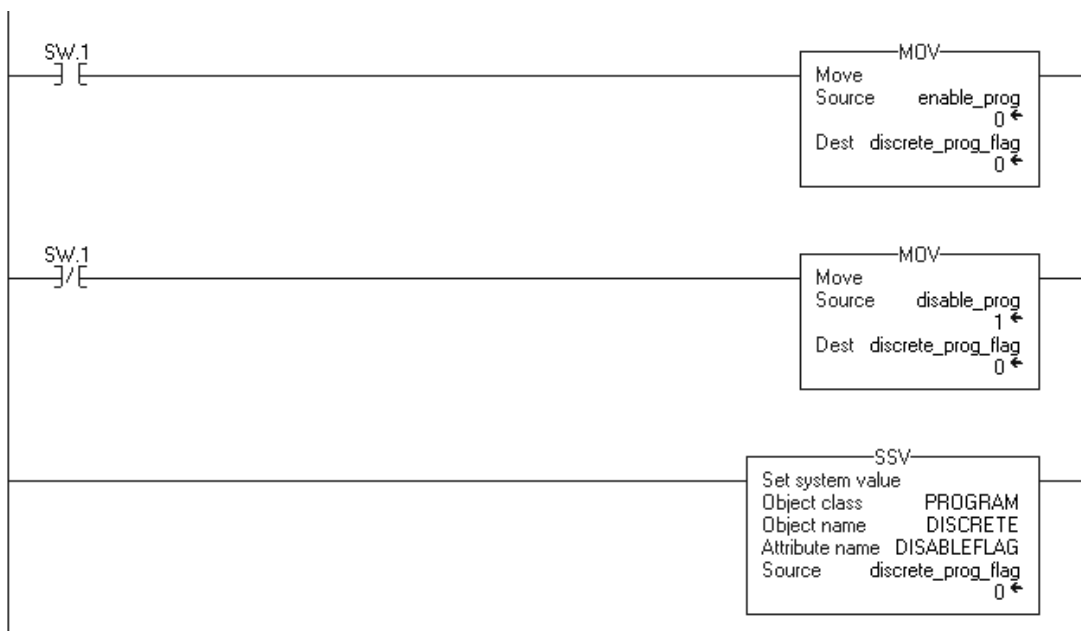


### Structured Text

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

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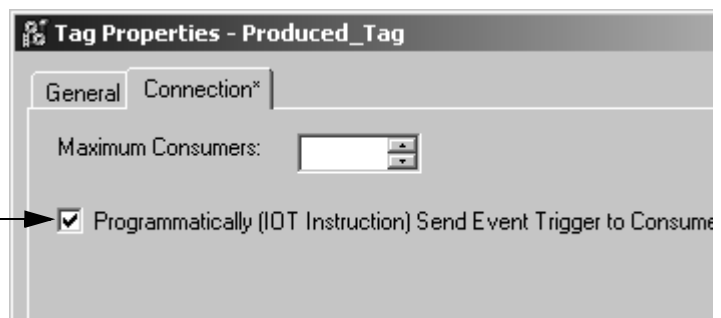
This configures the tag to update its event trigger only via an IOT instruction.

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.

**EtherNet/IP network**                                    **ControlNet network**



**Arithmetic Status Flags:**   not affected

**Fault Conditions:**   none

### Execution:

| Condition | Relay Ladder Action | Structured Text Action |
|---|---|---|
| prescan | The rung-condition-out is set to false. | No action taken. |
| rung-condition-in is false | The rung-condition-out is set to false. | 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:<br>• updates the connection of the specified tag.<br>• resets the RPI timer of the connection | |
| postscan | The rung-condition-out is set to false. | No action taken. |

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

### Relay Ladder
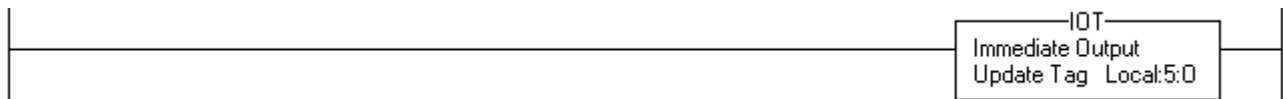


### Structured Text

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

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

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

```
Tag Properties - Produced_Tag

General | Connection*

Maximum Consumers:     [     ] [↕]

☑ Programmatically (IOT Instruction) Send Event Trigger to Consume
```
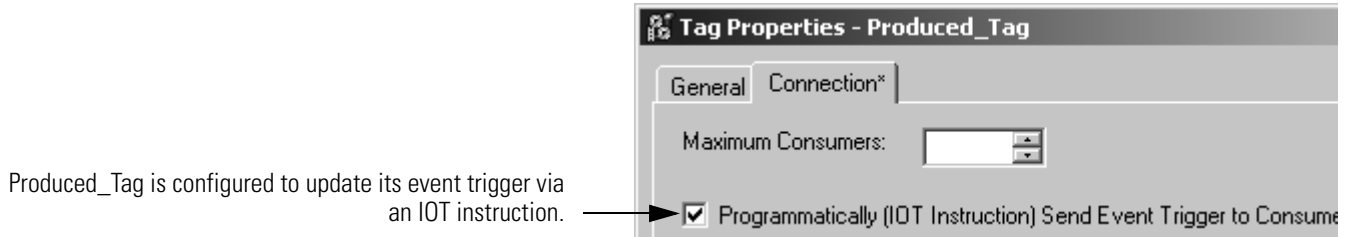
### Relay Ladder

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.

```
New_Data   Trigger_Consumer                    ┌──CPS──────────────────┐      ┌──IOT─────────────────┐
──┤ ├────────────[ONS]──────────────────┤ Synchronous Copy File │      │ Immediate Output     │
                                         │ Source    Source_Tag  ├──────┤ Update Tag  Produced_Tag │
                                         │ Dest    Produced_Tag  │      └──────────────────────┘
                                         │ Length           1    │
                                         └───────────────────────┘
```
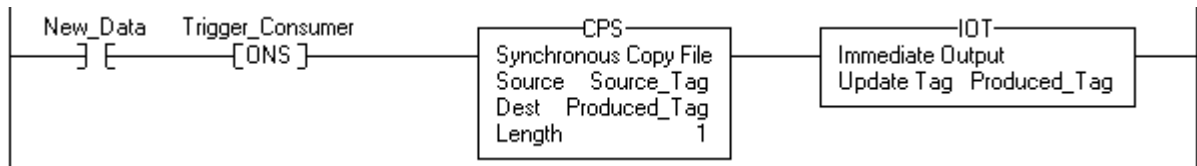
### Structured Text

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

# Compare Instructions

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

## Introduction

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

| If You Want To | Use This Instruction | Available In These Languages | See Page |
|---|---|---|---|
| compare values based on an expression | CMP | relay ladder<br>structured text[1] | 206 |
| test whether two values are equal | EQU | relay ladder<br>structured text[2]<br>function block | 211 |
| test whether one value is greater than or equal to a second value | GEQ | relay ladder<br>structured text[1]<br>function block | 215 |
| test whether one value is greater than a second value | GRT | relay ladder<br>structured text[1]<br>function block | 219 |
| test whether one value is less than or equal to a second value | LEQ | relay ladder<br>structured text[1]<br>function block | 223 |
| test whether one value is less than a second value | LES | relay ladder<br>structured text[1]<br>function block | 227 |
| test whether one value is between two other values | LIM | relay ladder<br>structured text[1]<br>function block | 231 |
| pass two values through a mask and test whether they are equal | MEQ | relay ladder<br>structured text[1]<br>function block | 237 |
| test whether one value is not equal to a second value | NEQ | relay ladder<br>structured text[1]<br>function block | 242 |

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

[2] There is no equivalent structured text instruction. Use the operator in an expression.

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

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

# Compare (CMP)

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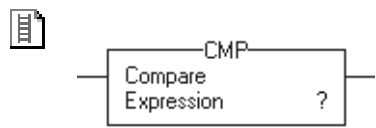
## Operands:

### Relay Ladder

| Operand | Type | Format | Description |
|---|---|---|---|
| Expression | SINT | 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 C, Structured Text Programming 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:**

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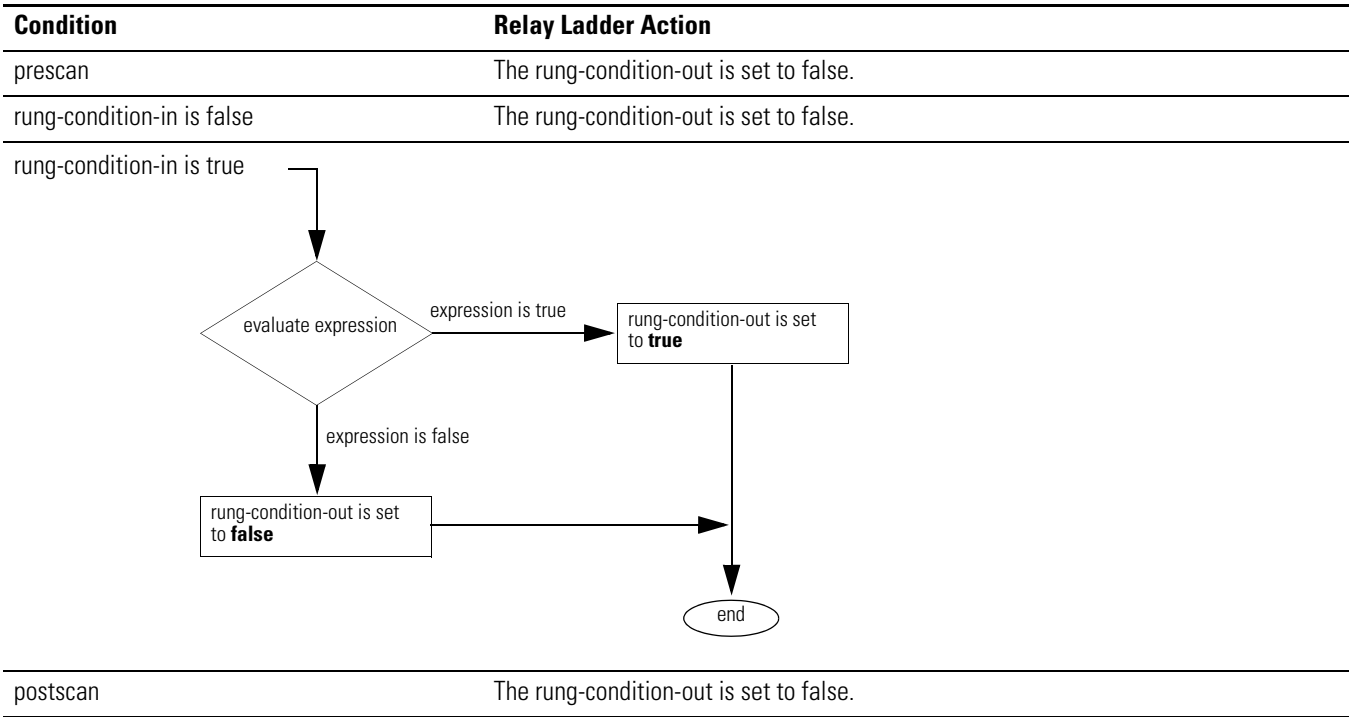
**Examples:** If the CMP instruction finds the expression true, the rung-condition-out is set to true.
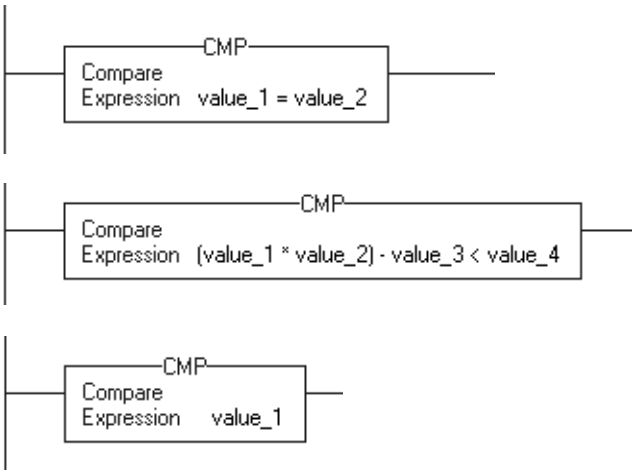


If you enter an expression without a comparison operator, such as *value_1 + value_2*, or *value_1*, the instruction evaluates the expression as:

| 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 | • *tag_b* + 5 <br> • *tag_c* AND *tag_d* <br> • (*tag_e* \*\* 2) MOD (*tag_f* / *tag_g*) |

## Determine The Order of Operation

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

Operations of equal order are performed from left to right.

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

# Use  Strings In an Expression

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

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

| Operator | Description |
|----------|-------------|
| = | equal |
| < | less than |
| <= | less than or equal |
| > | greater than |
| >= | greater than or equal |
| <> | not equal |

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

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

lesser / 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.

### Structured Text

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

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

See Appendix C, Structured Text Programming for information on the syntax of expressions within structured text.
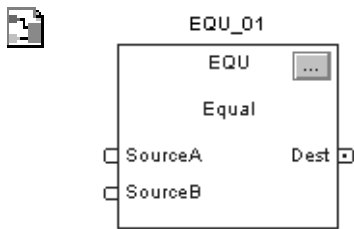
## Function Block

| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| EQU tag | FBD_COMPARE | structure | EQU structure |

## FBD_COMPARE Structure

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

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

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

**Arithmetic Status Flags:**  not affected
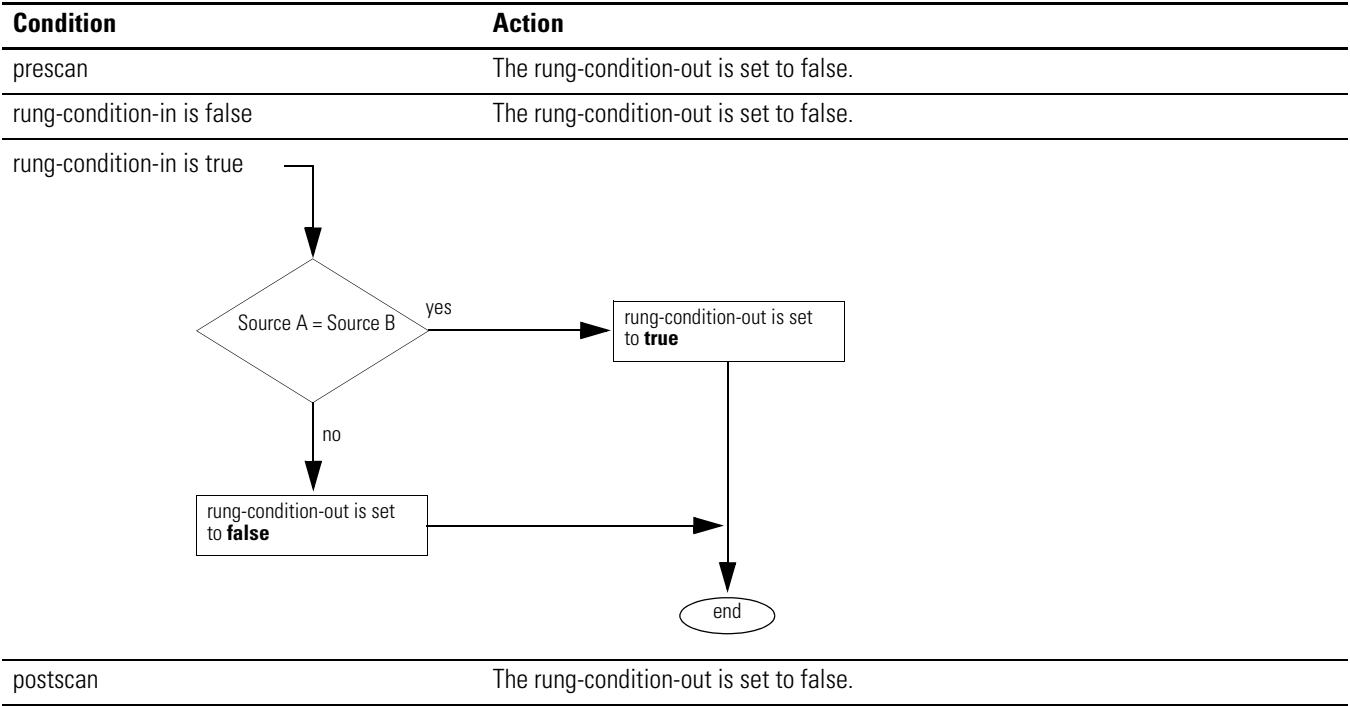
**Fault Conditions:**  none

**Execution:**

**Relay Ladder**

| Condition | Action |
|---|---|
| prescan | The rung-condition-out is set to false. |
| rung-condition-in is false | The rung-condition-out is set to false. |

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. |
| 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
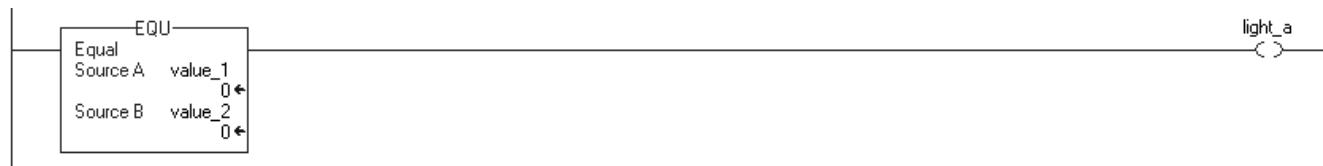


### Structured Text

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

### Function Block
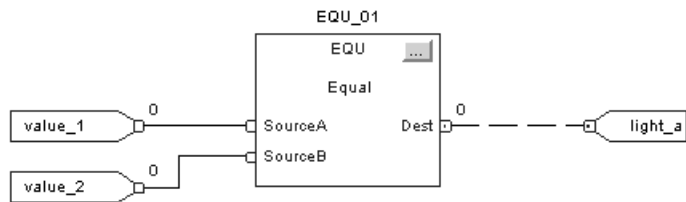
# Greater than or Equal to (GEQ)

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.

### Structured Text



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

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

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

## Function Block

GEQ_01

GEQ        [...]

Grtr Than or Eql (A>=B)

SourceA        Dest

SourceB

| 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. <br><br> Default is set. |
| SourceA | REAL | Value to test against SourceB. <br><br> Valid = any float |
| SourceB | REAL | Value to test against SourceA. <br><br> Valid = any float |
| **Output Parameter** | **Data Type** | **Description** |
| EnableOut | BOOL | The instruction produced a valid result. |
| Dest | BOOL | Result of the instruction. This is equivalent to rung-condition-out for the relay ladder GEQ instruction. |

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

When you compare strings:

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

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

lesser ↑    greater ↓

AB < B

a > B

**Arithmetic Status Flags:**    not affected

**Fault Conditions:**    none

**Execution:**

📄    **Relay Ladder**

| Condition | Action |
|---|---|
| prescan | The rung-condition-out is set to false. |
| rung-condition-in is false | The rung-condition-out is set to false. |

rung-condition-in is true

```
                          ┌──────┐
                          ▼
                    ╱ Source A ≥ Source B ╲  ──yes──▶  ┌─────────────────────┐
                    ╲                     ╱             │ rung-condition-out is set │
                                                        │ to **true**              │
                          │ no                          └─────────────────────┘
                          ▼                                        │
              ┌─────────────────────┐                              │
              │ rung-condition-out is set │  ──────────────────▶   │
              │ to **false**             │                          ▼
              └─────────────────────┘                          ( end )
```

| | |
|---|---|
| postscan | The rung-condition-out is set to false. |
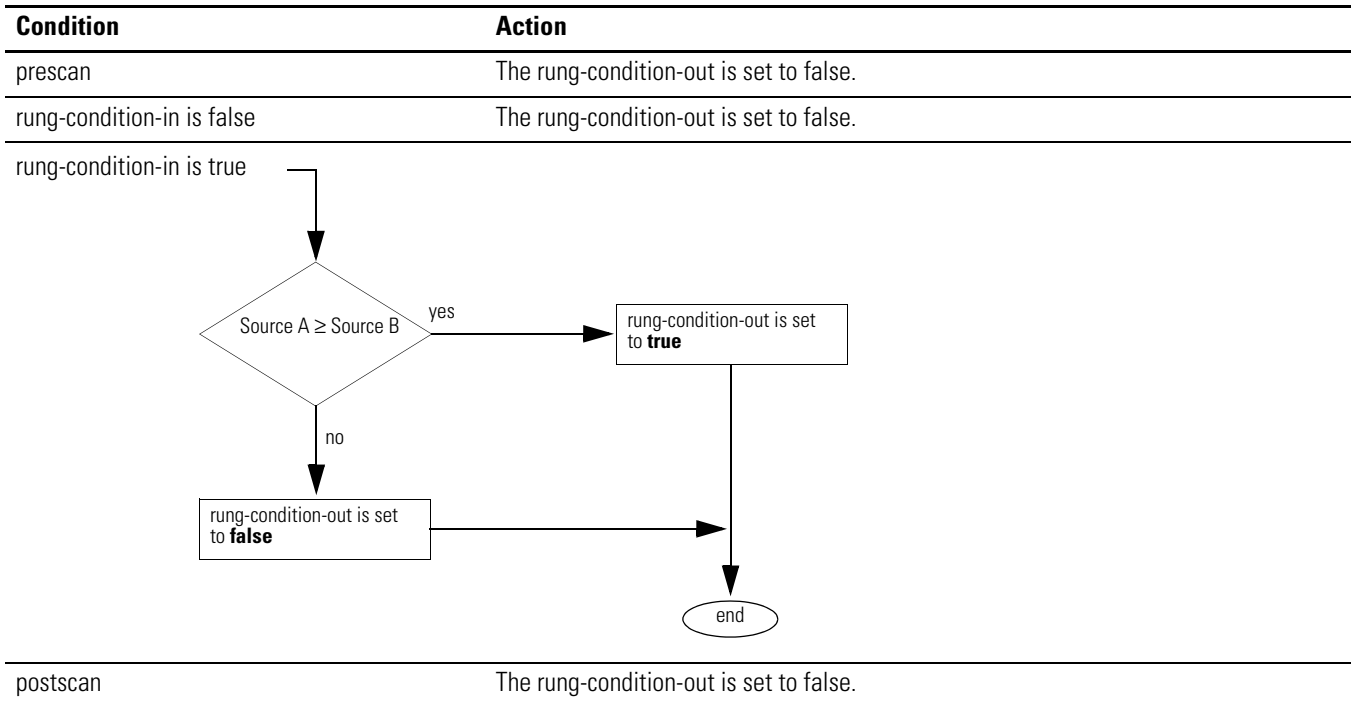
🗔    **Function Block**

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

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

### Relay Ladder



### Structured Text

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

### Function Block
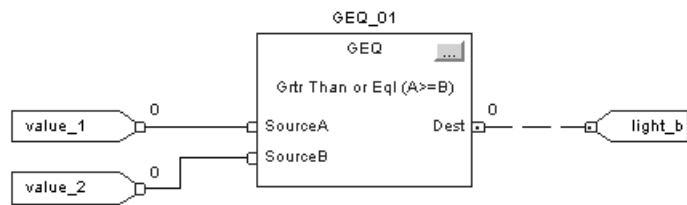
# Greater Than (GRT)

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

## Operands:

📄

```
——GRT——
Greater Than (A>B)
Source A        ?
                ??
Source B        ?
                ??
```

### 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.

### Structured Text

📄

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

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

See [Appendix C, Structured Text Programming](#) for information on the syntax of expressions within structured text.

### Function Block

📄

```
GRT_01
——————————
GRT      [...]
Greater Than (A>B)
□ SourceA    Dest □
□ SourceB
```
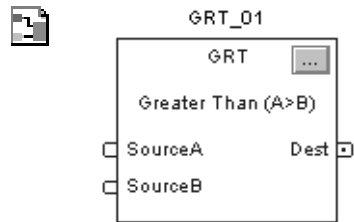
| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| GRT tag | FBD_COMPARE | structure | GRT structure |

## 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 |

AB < B

a > B

**Arithmetic Status Flags:**   not affected

**Fault Conditions:**   none

**Execution:**

### Relay Ladder

| Condition | Action |
|---|---|
| prescan | The rung-condition-out is set to false. |
| rung-condition-in is false | The rung-condition-out is set to false. |

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. |
| EnableIn is set | The instruction executes.<br><br>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

```
            ─GRT──────────
            Greater Than (A>B)                                                          light_1
            Source A    value_1                                                          ─( )─
                              0←
            Source B    value_2
                              0←
```

### Structured Text

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

### Function Block
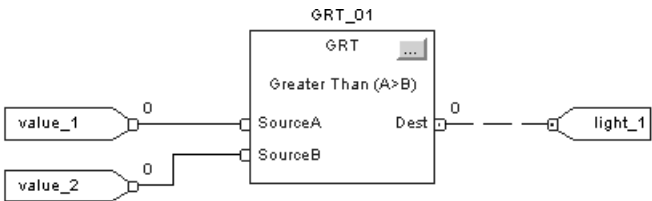
## Less Than or Equal to (LEQ)

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

### Operands:

📄

```
            ─LEQ──
  Less Than or Eql (A<=B)
  Source A                ?
                         ??
  Source B                ?
                         ??
```

### 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 C, Structured Text Programming for information on the syntax of expressions within structured text.
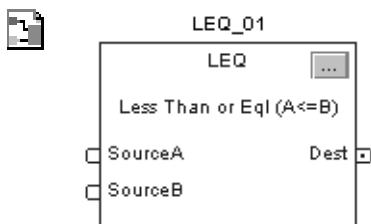
## Function Block

| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| LEQ tag | FBD_COMPARE | structure | LEQ structure |

## FBD_COMPARE Structure

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

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

When you compare strings:

- 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 |

AB < B

a > B

**Arithmetic Status Flags:**    not affected

**Fault Conditions:**    none

**Execution:**

### Relay Ladder

| Condition | Action |
|---|---|
| prescan | The rung-condition-out is set to false. |
| rung-condition-in is false | The rung-condition-out is set to false. |

rung-condition-in is true

```
                    ┌───
                    ▼
            ╱ Source A ≤ Source B ╲ ──yes──►  ┌────────────────────────┐
            ╲                     ╱            │ rung-condition-out is set │
                    │                          │ to **true**               │
                   no                          └────────────────────────┘
                    ▼                                      │
          ┌────────────────────────┐                      │
          │ rung-condition-out is set │ ──────────►        │
          │ to **false**              │                    ▼
          └────────────────────────┘              ( end )
```

| | |
|---|---|
| postscan | The rung-condition-out is set to false. |

### Function Block

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

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

### Relay Ladder



### Structured Text

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

### Function Block
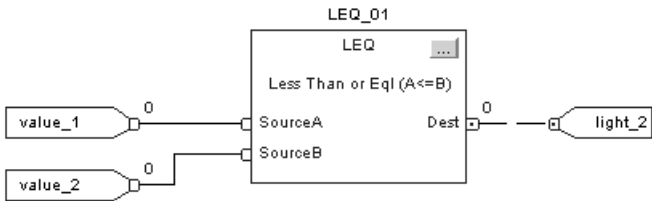
# Less Than (LES)

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

## Operands:

📄



### Relay Ladder

| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| Source A | SINT | 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.

### Structured Text

📄

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

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

See [Appendix C, Structured Text Programming](#) for information on the syntax of expressions within structured text.

### Function Block

📄



| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| LES tag | FBD_COMPARE | structure | LES structure |

## FBD_COMPARE Structure

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

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

When you compare strings:

- 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 |

AB < B

a > B

lesser / greater

**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 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
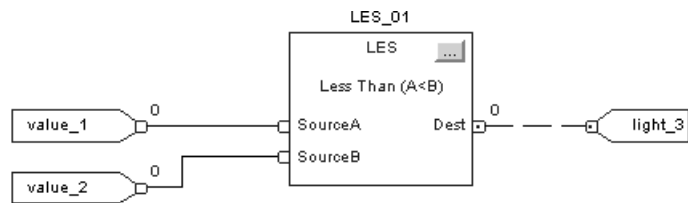
## Limit (LIM)

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

### Operands:

### Relay Ladder

```
        ┌─LIM──────────────┐
        │Limit Test (CIRC) │
        │Low Limit       ? │
        │               ?? │
        │Test            ? │
        │               ?? │
        │High Limit      ? │
        │               ?? │
        └──────────────────┘
```
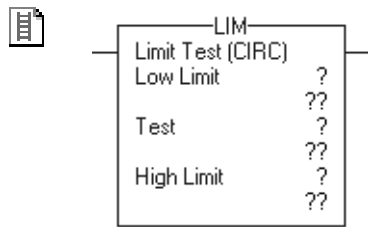
| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| Low limit | SINT | immediate | value of lower limit |
| | INT | tag | |
| | **DINT** | | |
| | **REAL** | | |
| | A SINT or INT tag converts to a DINT value by sign-extension. | | |
| Test | SINT | immediate | value 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;
```
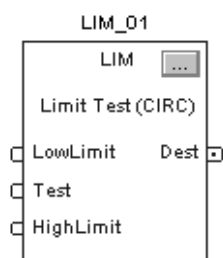
## Function Block

| 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 |
|--------------|-------------------|---------------------------|
| ≤High Limit | equal to or between limits | true |
| | not equal to or outside limits | false |
| ≥ High Limit | equal to or outside limits | true |
| | not equal to or inside limits | false |

Signed integers "roll over" from the maximum positive number to the maximum negative number when the most significant bit is set. For example, in 16-bit integers (INT type), the maximum positive integer is 32,767, which is represented in hexadecimal as 16#7FFF (bits 0 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
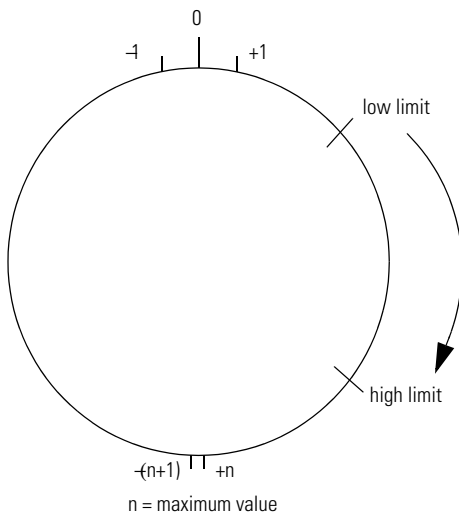
**Low Limit ≥ High Limit**

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

## Relay Ladder

```
        ┌─────LIM─────────────┐
        │ Limit Test (CIRC)   │                                      light_1
        │ Low Limit        0  │                                       ( )
        │                     │
        │ Test          value │
        │                  0 ← │
        │ High Limit      100 │
        └─────────────────────┘
```

## 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

```
                              LIM_01
                              LIM    [...]
                              Limit Test (CIRC)
              0
       [ 0 ]──────┐
                  └──────LowLimit    Dest ├─ 0 ──┐
              0                            │      └─[ light_1 ]
     [value]──────────────Test            │
              100                          │
      [100]──────┐                         │
                 └───────HighLimit         │
```
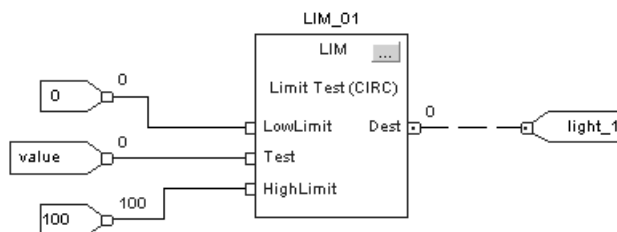
**Example 2:**   **Low Limit ≥ High Limit:**

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

**Relay Ladder**



**Structured Text**

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

        light_1 := 1;

ELSE

        light_1 := 0;

END_IF;
```

**Function Block**
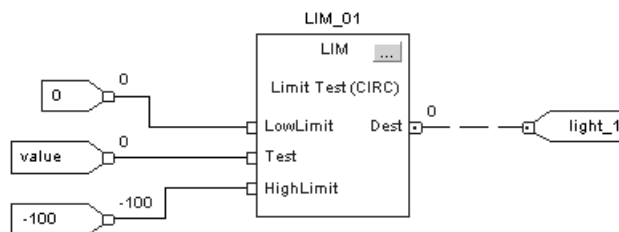
## Mask Equal to (MEQ)

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

### Operands:
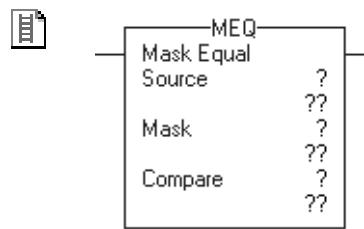
### Relay Ladder

| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| Source | SINT | immediate | value to test against Compare |
| | INT | tag | |
| | **DINT** | | |
| | 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 | |
| | **DINT** | | |
| | A SINT or INT tag converts to a DINT value by zero-fill. | | |
| Compare | SINT | immediate | value to test against Source |
| | INT | tag | |
| | **DINT** | | |
| | 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;
```
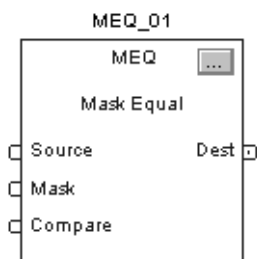
### Function Block

| 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**

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

| *value_1* | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

| *mask_1* | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

| Masked *value_1* | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | x | x | x | x |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

| *value_2* | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

| *mask_1* | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

| Masked *value_2* | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | x | x | x | x |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

## Relay Ladder



## Structured Text

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

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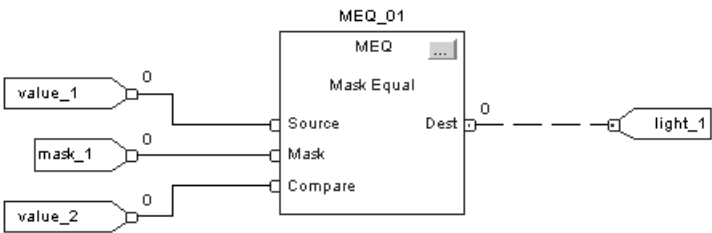
**Example 2:** If the masked *value_1* is equal to the masked *value_2*, set *light_1*. If the masked *value_1* is not equal to the masked *value_2*, clear *light_1*. This example shows that the masked values are not equal. A 0 in the mask restrains the instruction from comparing that bit (shown by x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

| *value_1* | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

| *mask_1* | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

| Masked *value_1* | x | x | x | x | x | x | x | x | x | x | x | x | 1 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

| *value_2* | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

| *mask_1* | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

| Masked *value_2* | x | x | x | x | x | x | x | x | x | x | x | x | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

**Relay Ladder**



```
-------MEQ-------
Mask Equal
Source            value_1
  2#0101_0101_1111_1111 ←
Mask              mask_1
  2#0000_0000_0000_1111 ←
Compare           value_2
  2#0101_0101_1111_0000 ←
```
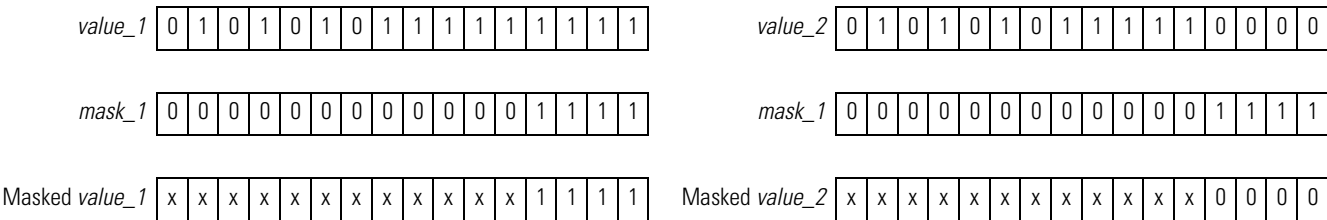
**Structured Text**

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

**Function Block**
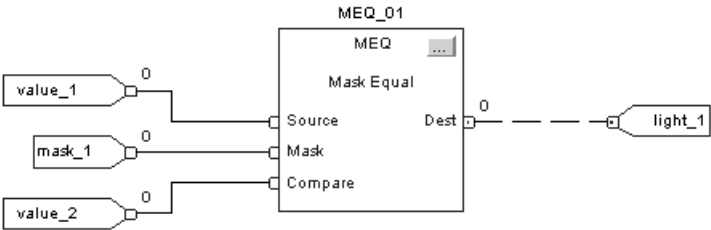
# Not Equal to (NEQ)

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.

### Structured Text

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

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 <u>Appendix C, Structured Text Programming</u> for information on the syntax of expressions within structured text.
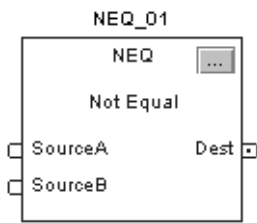
## Function Block



| 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 (up) greater (down)

AB < B

a > B

**Arithmetic Status Flags:**  not affected

**Fault Conditions:**  none

### Execution:

### Relay Ladder

| Condition | Action |
| --- | --- |
| prescan | The rung-condition-out is set to false. |
| rung-condition-in is false | The rung-condition-out is set to false. |

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. |
| EnableIn is set | The instruction executes.<br><br>EnableOut is set. |
| postscan | No action taken. |

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

### Relay Ladder



### Structured Text

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

### Function Block
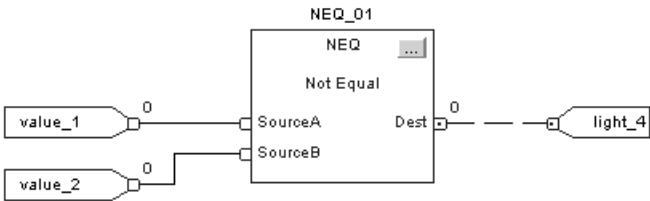
**Notes:**

# Compute/Math Instructions

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

## Introduction

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

| If You Want To | Use This Instruction | Available In These Languages | See Page |
|---|---|---|---|
| evaluate an expression | CPT | relay ladder<br>structured text[1] | 248 |
| add two values | ADD | relay ladder<br>structured text[2]<br>function block | 252 |
| subtract two values | SUB | relay ladder<br>structured text[2]<br>function block | 255 |
| multiply two values | MUL | relay ladder<br>structured text[2]<br>function block | 258 |
| divide two values | DIV | relay ladder<br>structured text[2]<br>function block | 261 |
| determine the remainder after one value is divided by another | MOD | relay ladder<br>structured text[2]<br>function block | 266 |
| calculate the square root of a value | SQR<br><br>SQRT[3] | relay ladder<br>structured text<br>function block | 270 |
| take the opposite sign of a value | NEG | relay ladder<br>structured text[2]<br>function block | 274 |
| take the absolute value of a value | ABS | relay ladder<br>structured text<br>function block | 277 |

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

[2] There is no equivalent structured text instruction. Use the operator in an expression.

[3] Structured text only.

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

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

# Compute (CPT)

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

## Operands:

```
      ┌─CPT──────────────┐
      │ Compute          │
      │ Dest          ?  │
      │               ?? │
      │ Expression     ? │
      └──────────────────┘
```
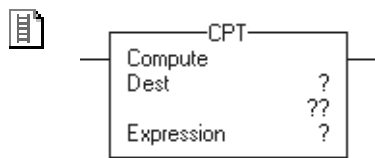
### Relay Ladder

| Operand | Type | Format: | Description |
|---------|------|---------|-------------|
| Destination | SINT<br>INT<br>**DINT**<br>**REAL** | tag | tag to store the result |
| Expression | SINT<br>INT<br>**DINT**<br>**REAL** | immediate<br><br>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 <u>Appendix C, Structured Text Programming</u> 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*.

```
                         ┌─CPT──────────────────────┐
                         │ Compute                  │
                         │ Dest              result_1│
                         │                      0.0 ←│
                         │ Expression  (value_1*5)/(value_2/7) │
                         └──────────────────────────┘
```
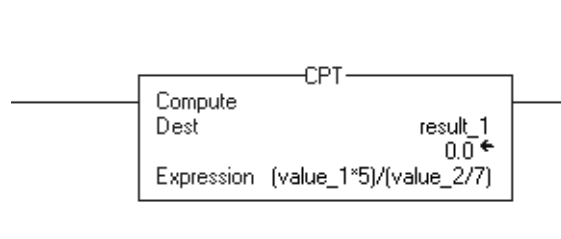
**Example 2:** When enabled, the CPT instruction truncates *float_value_1* and *float_value_2*, raises the truncated *float_value_2* to the power of two and divides the truncated *float_value_1* by that result, and stores the remainder after the division in *float_value_result_cpt*.

```
                    ┌─CPT──────────────────────────────┐
                    │ Compute                          │
                    │ Dest             float_value_result_cpt│
                    │                       14.000061 ←│
                    │ Expression  (trn(float_value_1))MOD(trn(float_value_2)**2) │
                    └──────────────────────────────────┘
```

# 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(*tag_a*) |
| two operands | operand_a operator operand_b | • *tag_b* + 5 <br> • *tag_c* AND *tag_d* <br> • (*tag_e* ** 2) MOD (*tag_f* / *tag_g*) |

## Determine the order of operation

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

Operations of equal order are performed from left to right.

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

# Add (ADD)

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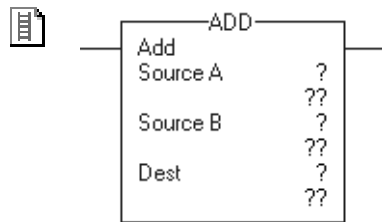
## Operands:

### 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** | | |

### Structured Text

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

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

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

### Function Block

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

## **FBD_MATH Structure**

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

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

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

**Fault Conditions:** none

**Execution:**

📖 **Relay Ladder**

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

## Function Block

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

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

### Relay Ladder
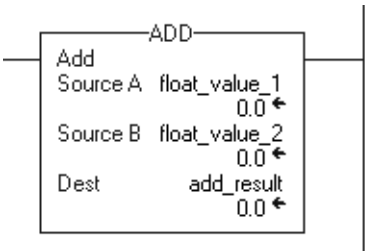


### Structured Text

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

### Function Block
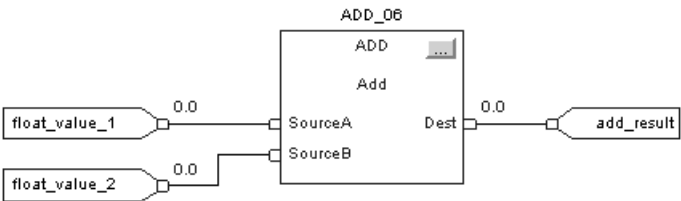
# Subtract (SUB)

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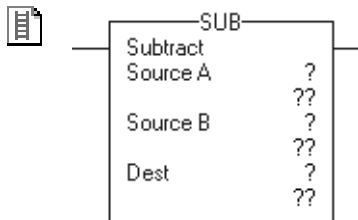
**Operands:**

## Relay Ladder

| Operand: | Type: | Format: | Description: |
|---|---|---|---|
| Source A | SINT | 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** | | |

## Structured Text

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

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

See [Appendix C, Structured Text Programming](#) for information on the syntax of expressions within structured text.
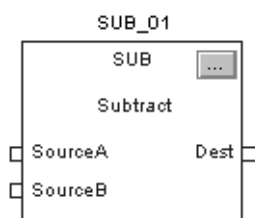
## Function Block

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

## FBD_MATH Structure

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

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

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

**Fault Conditions:** none

**Execution:**

### Relay Ladder

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

## Function Block

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

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

### Relay Ladder
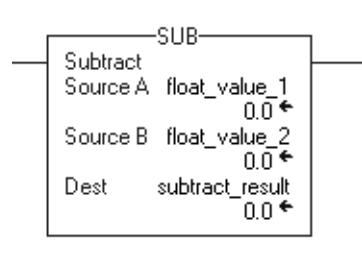


### Structured Text

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

### Function Block
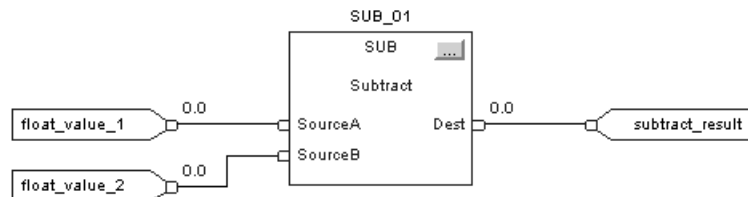
# Multiply (MUL)

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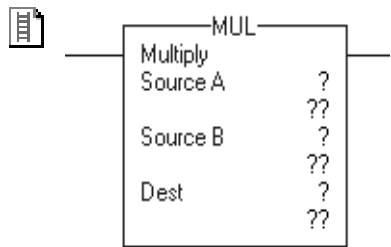
## Operands:

### Relay Ladder

| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| Source A | SINT | 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** | | |

### Structured Text

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

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

See [Appendix C, Structured Text Programming](#) for information on the syntax of expressions within structured text.
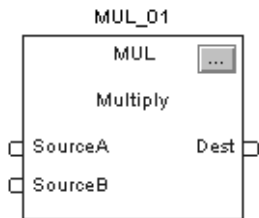
### Function Block

| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| MUL tag | FBD_MATH | structure | MUL structure |

## FBD_MATH Structure

| Input Parameter | Data Type | Description |
|---|---|---|
| EnableIn | BOOL | Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set. |
| Source A | REAL | Value of the multiplicand. Valid = any float |
| Source B | REAL | Value of the multiplier. Valid = any float |

| Output Parameter | Data Type | Description |
|---|---|---|
| EnableOut | BOOL | The instruction produced a valid result. |
| Dest | REAL | Result of the math instruction. Arithmetic status flags are set for this output. |

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

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

**Fault Conditions:** none

**Execution:**

### Relay Ladder

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

## Function Block

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

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

### Relay Ladder
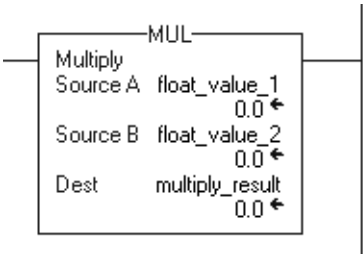


### Structured Text

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

### Function Block
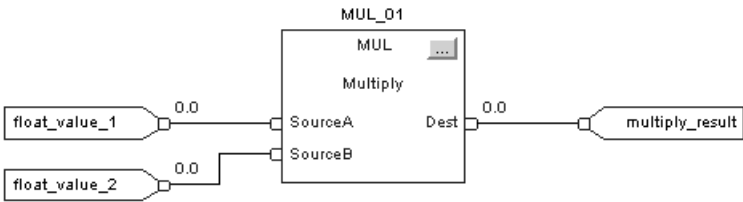
# Divide (DIV)

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

## Operands:

```
           ─DIV──────
          Divide
          Source A      ?
                       ??
          Source B      ?
                       ??
          Dest          ?
                       ??
```

### 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** | | |

### Structured Text

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

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

See Appendix C, Structured Text Programming for information on the syntax of expressions within structured text.
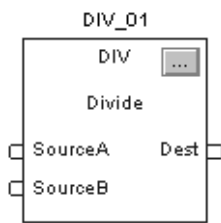
## Function Block

| Operand | Type | | Format | Description |
|---------|------|--|--------|-------------|
| DIV tag | FBD_MATH | | structure | DIV structure |

## FBD_MATH Structure

| Input Parameter | Data Type | Description |
|-----------------|-----------|-------------|
| EnableIn | BOOL | Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set. |
| Source A | REAL | Value of the dividend. Valid = any float |
| Source B | REAL | Value of the divisor. Valid = any float |

| Output Parameter | Data Type | Description |
|------------------|-----------|-------------|
| EnableOut | BOOL | The instruction produced a valid result. |
| Dest | REAL | Result of the math instruction. Arithmetic status flags are set for this output. |

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

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

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<br><br>The rung-condition-out is set to true. |
| postscan | The rung-condition-out is set to false. |

## Function Block

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

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

### Relay Ladder
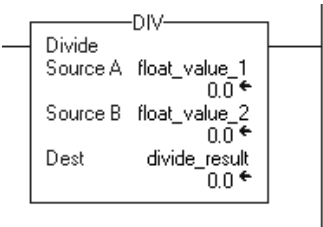


### Structured Text

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

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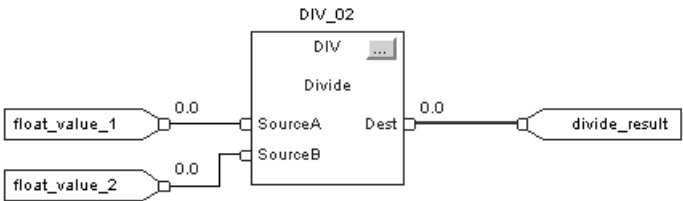
**Example 2:**  The DIV and MOV instructions work together to divide two integers, round the result, and place the result in an integer tag:

- The DIV instruction divides *dint_a* by *dint_b*.

- To round the result, the Destination is a REAL tag. (If the destination was an integer tag (SINT, INT, or DINT), the instruction would truncate the result.)

- The MOV instruction moves the rounded result (*real_temp*) from the DIV to *divide_result_rounded*.

- Since *divide_result_rounded* is a DINT tag the value from *real_temp* is rounded and placed in the DINT destination.

### Relay Ladder
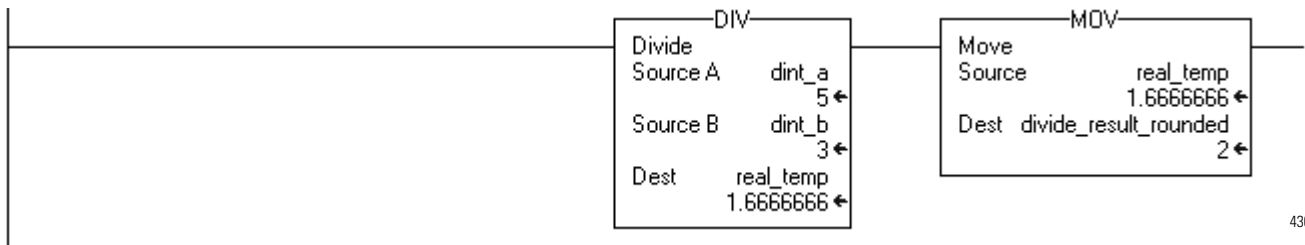
# Modulo (MOD)

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

## Operands:

### Relay Ladder

| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| Source A | SINT | 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** | | |

### Structured Text

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

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

See [Appendix C, Structured Text Programming](#) for information on the syntax of expressions within structured text.
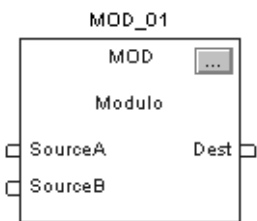
### Function Block

| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| MOD tag | FBD_MATH | structure | MOD structure |

## FBD_MATH Structure

| Input Parameter | Data Type | Description |
|---|---|---|
| EnableIn | BOOL | Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set. |
| Source A | REAL | Value of the dividend. Valid = any float |
| Source B | REAL | Value of the divisor. Valid = any float |

| Output Parameter | Data Type | Description |
|---|---|---|
| EnableOut | BOOL | The instruction produced a valid result. |
| Dest | REAL | Result of the math instruction. Arithmetic status flags are set for this output. |

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

- 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 )<br><br>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

```
        ┌──MOD──────────────┐
        │ Modulo            │
        │ Source A  dividend │
        │              10 ←  │
        │                    │
        │ Source B   divisor │
        │               3 ←  │
        │ Dest     remainder │
        │               1 ←  │
        └────────────────────┘
```
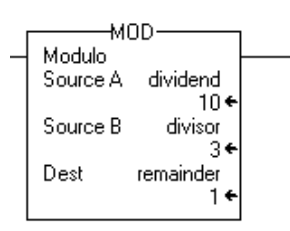
### Structured Text

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

### Function Block
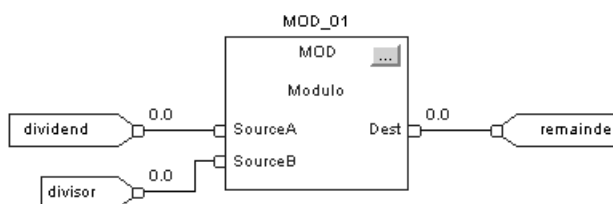
## Square Root (SQR)

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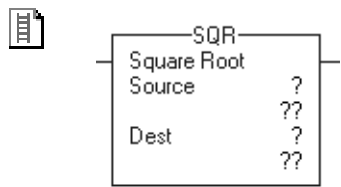
### Operands:



### 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** | | |

### Structured Text



`dest := SQRT(source);`

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

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

## Function Block

```
      SQR_01
       SQR      [...]
    Square Root

  Source      Dest
```
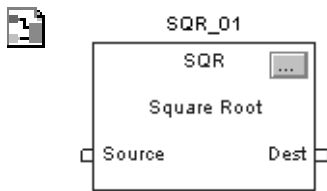
| Operand | Type | Format | Description |
|---|---|---|---|
| SQR tag | FBD_MATH_ADVANCED | structure | SQR structure |

## 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
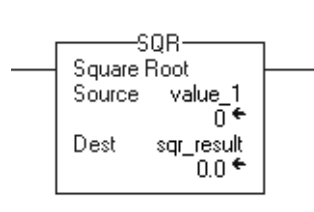


### Structured Text

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

### Function Block
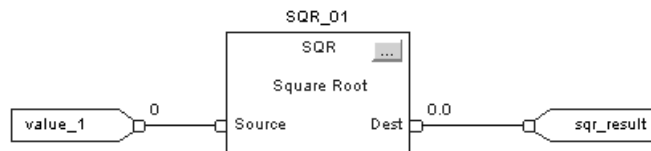
# Negate (NEG)

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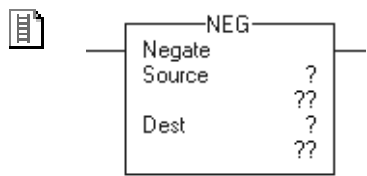
## Operands:

### Relay Ladder

| Operand | Type | Format | Description |
|---|---|---|---|
| Source | SINT | immediate | value to negate |
| | INT | tag | |
| | **DINT** | | |
| | **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** | | |

### Structured Text

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

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

See <u>Appendix C, Structured Text Programming</u> for information on the syntax of expressions within structured text.
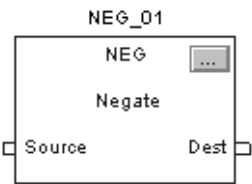
### Function Block

| Operand | Type | Format | Description |
|---|---|---|---|
| NEG tag | FBD_MATH_ADVANCED | structure | NEG structure |

## FBD_MATH Structure

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

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

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

**Fault Conditions:** none

**Execution:**

### Relay Ladder

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

### Function Block

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

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

### Relay Ladder

```
            ─NEG─────────
   ──────┤  Negate
         │  Source    value_1
         │                0 ←
         │  Dest  negate_result
         │                0 ←
```

### Structured Text

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

### Function Block
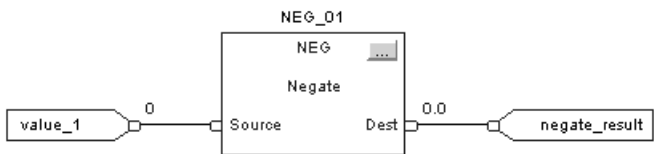
## Absolute Value (ABS)

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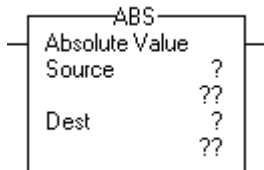
**Operands:**

### 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** | | |

### Structured Text

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

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

See Appendix C, Structured Text Programming for information on the syntax of expressions within structured text.
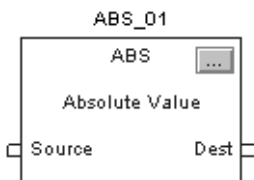
### Function Block

| Operand | Type | Format | Description |
|---|---|---|---|
| ABS tag | FBD_MATH_ADVANCED | structure | ABS structure |

## FBD_MATH_ADVANCED Structure

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

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

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

**Fault Conditions:**  none

**Execution:**



### Relay Ladder

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



### Function Block

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

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

### Relay Ladder
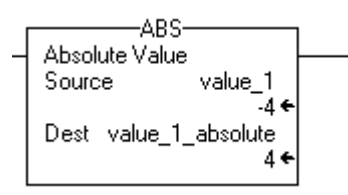


### Structured Text

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

### Function Block
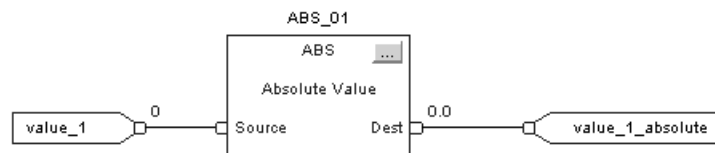
**Notes:**

# Move/Logical Instructions
## (MOV, MVM, BTD, 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 | 283 |
| | | structured text[1] | |
| copy a specific part of an integer | MVM | relay ladder | 285 |
| copy a specific part of an integer in function block | MVMT | structured text | 288 |
| | | function block | |
| move bits within an integer or between integers | BTD | relay ladder | 291 |
| move bits within an integer or between integers in function block | BTDT | structured text | 294 |
| | | function block | |
| clear a value | CLR | structured text[1] | 297 |
| | | relay ladder | |
| rearrange the bytes of a INT, DINT, or REAL tag | SWPB | relay ladder | 299 |
| | | structured text | |

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

The logical instructions perform operations on bits.

| If you want to: | Use this instruction: | Available in these languages | See page |
|---|---|:---:|---|
| bitwise AND operation | Bitwise AND | relay ladder | 303 |
| | &[1] | structured text[2] | |
| | | function block | |
| bitwise OR operation | Bitwise OR | relay ladder | 306 |
| | | structured text[2] | |
| | | function block | |
| bitwise, exclusive OR operation | Bitwise XOR | relay ladder | 310 |
| | | structured text[2] | |
| | | function block | |
| bitwise NOT operation | Bitwise NOT | relay ladder | 314 |
| | | structured text[2] | |
| | | function block | |
| logically AND as many as eight boolean inputs. | Boolean AND (BAND) | structured text[2] | 317 |
| | | function block | |
| logically OR as many as eight boolean inputs. | Boolean OR (BOR) | structured text[2] | 320 |
| | | function block | |
| perform an exclusive OR on two boolean inputs. | Boolean Exclusive OR (BXOR) | structured text[2] | 323 |
| | | function block | |
| complement a boolean input. | Boolean NOT (BNOT) | structured text[2] | 326 |
| | | function block | |

[1]    Structured text only.

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

## Move (MOV)

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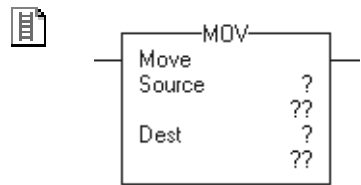
### Operands:

### Relay Ladder

| Operand: | Type: | Format | Description: |
|----------|-------|--------|-------------|
| Source | SINT | 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** | | |

### Structured Text

```
dest := source;
```

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

See Appendix C, 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
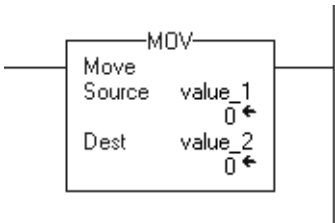


### Structured Text

```
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 .

## Operands:

### Relay Ladder

```
-------MVM-------
Masked Move
Source          ?
               ??
Mask            ?
               ??
Dest            ?
               ??
```
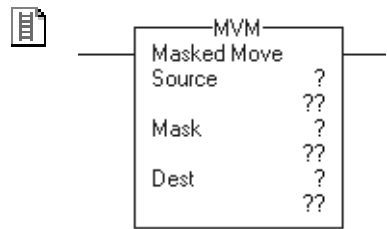
| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| Source | SINT | immediate | value to move |
| | INT | tag | |
| | **DINT** | | |
| | A SINT or INT tag converts to a DINT value by zero-fill. | | |
| Mask | SINT | immediate | which bits to block or pass |
| | INT | 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** | | |

### Structured Text

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

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

See Appendix C, 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:

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

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

| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

| 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

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

### Relay Ladder

```
                        ┌MVM──────────────────────────────────┐
                        │ Masked Move                          │
────────────────────────┤ Source                      value_a │──────────
                        │   2#0101_0101_0101_0101_0101_0101_0101_0101 ←│
                        │ Mask                         mask_2  │
                        │   2#1111_0000_1111_0000_1111_0000_1111_0000 ←│
                        │ Dest                        value_b │
                        │   2#1111_1111_1111_1111_1111_1111_1111_1111 ←│
                        └──────────────────────────────────────┘
```

### 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 .

### Operands:

MVMT(MVMT_tag);

### Structured Text

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

### Function Block

MVMT_01

MVMT

Move With Mask with Target

Source          Dest

Mask

Target
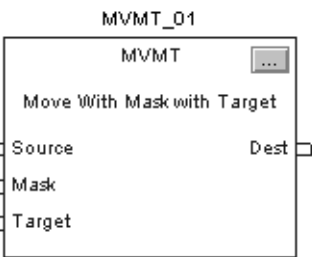
| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| MVMT tag | FBD_MASKED_MOVE | structure | MVMT structure |

### FBD_MASKED_MOVE Structure

| Input Parameter | Data Type | Description |
|-----------------|-----------|-------------|
| EnableIn | BOOL | **Function Block** |
| | | If cleared, the instruction does not execute and outputs are not updated. |
| | | If set, the instruction executes. |
| | | Default is set. |
| | | **Structured Text** |
| | | No effect. The instruction executes. |
| Source | DINT | Input value to move to Destination based on value of Mask. |
| | | Valid = any integer |
| Mask | DINT | Mask of bits to move from Source to Dest. All bits set to one cause the corresponding bits to move from Source to Dest. All bits that are set to zero cause the corresponding bits not to move from Source to Dest. |
| | | Valid = any integer |
| Target | DINT | Input value to move to Dest prior to moving Source bits through the Mask. |
| | | Valid = any integer |

| Output Parameter | Data Type | Description |
|---|---|---|
| EnableOut | BOOL | The instruction produced a valid result. |
| Dest | DINT | Result of masked move instruction. Arithmetic status flags are set for this output. |

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

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

## Enter an Immediate Mask Value 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. | EnableIn is always set. |
| | EnableOut is set. | The instruction executes. |
| postscan | No action taken. | No action taken. |

**Example:**    **1.** Copy Target into Dest.

| Target | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

| Dest | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
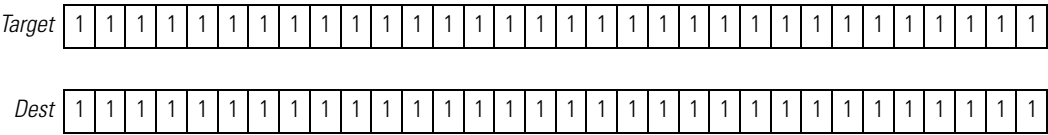
**2.** Mask Source and compare it to Dest. Any required changes are made in Dest. Source and Target remain unchanged. A 0 in the mask restrains the instruction from comparing that bit (shown by x in the example).

| Source | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

| Mask1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

| Dest | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
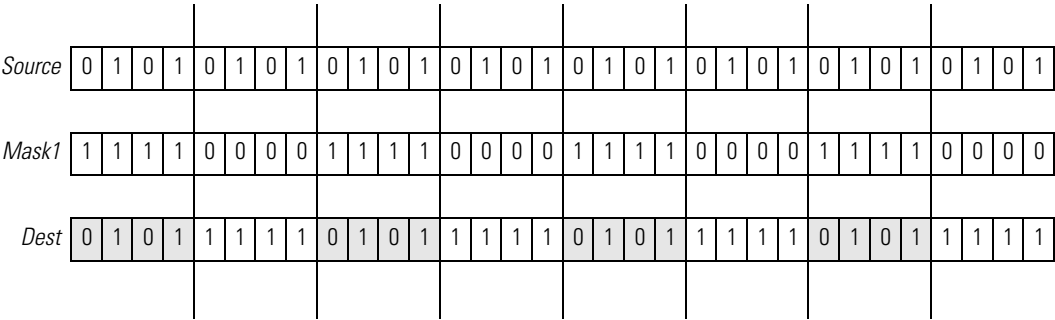
The shaded boxes show the bits that changed.

### Structured Text

```
MVMT_01.Source := value_1;
MVMT_01.Mask := mask1;
MVMT_01.Target := target;

MVMT(MVMT_01);

value_masked := MVMT_01.Dest;
```

### Function Block
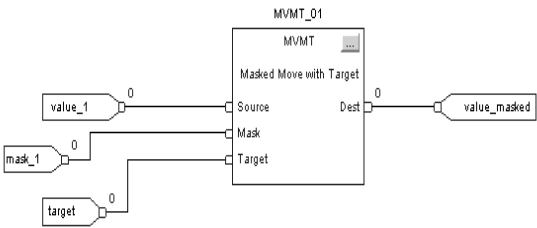
# Bit Field Distribute (BTD)

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

This instruction is available in structured text and function block as BTDT, see page 294.

## Operands:
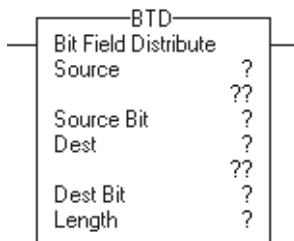
### Relay Ladder

| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| Source | SINT | immediate | tag that contains the bits to move |
| | INT | tag | |
| | **DINT** | | |
| | 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 | | |
| | **DINT** | | |
| 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 (1-32) | number of bits to move |

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

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

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

**Arithmetic Status Flags:** not affected

**Fault Conditions:**   none

**Execution:**

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

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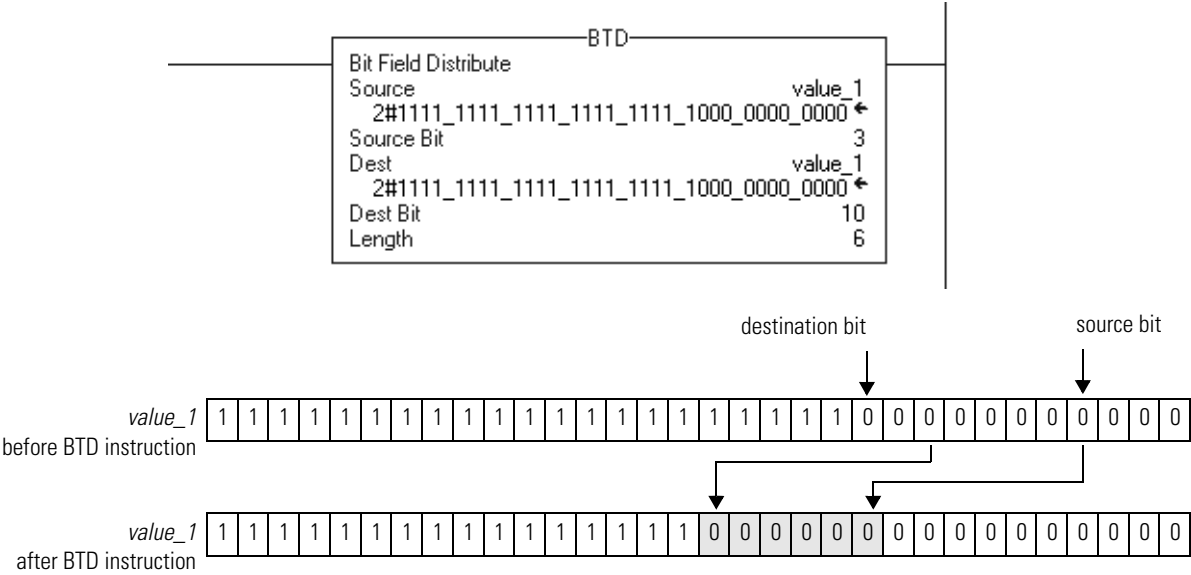


The shaded boxes show the bits that changed in *value_1*.

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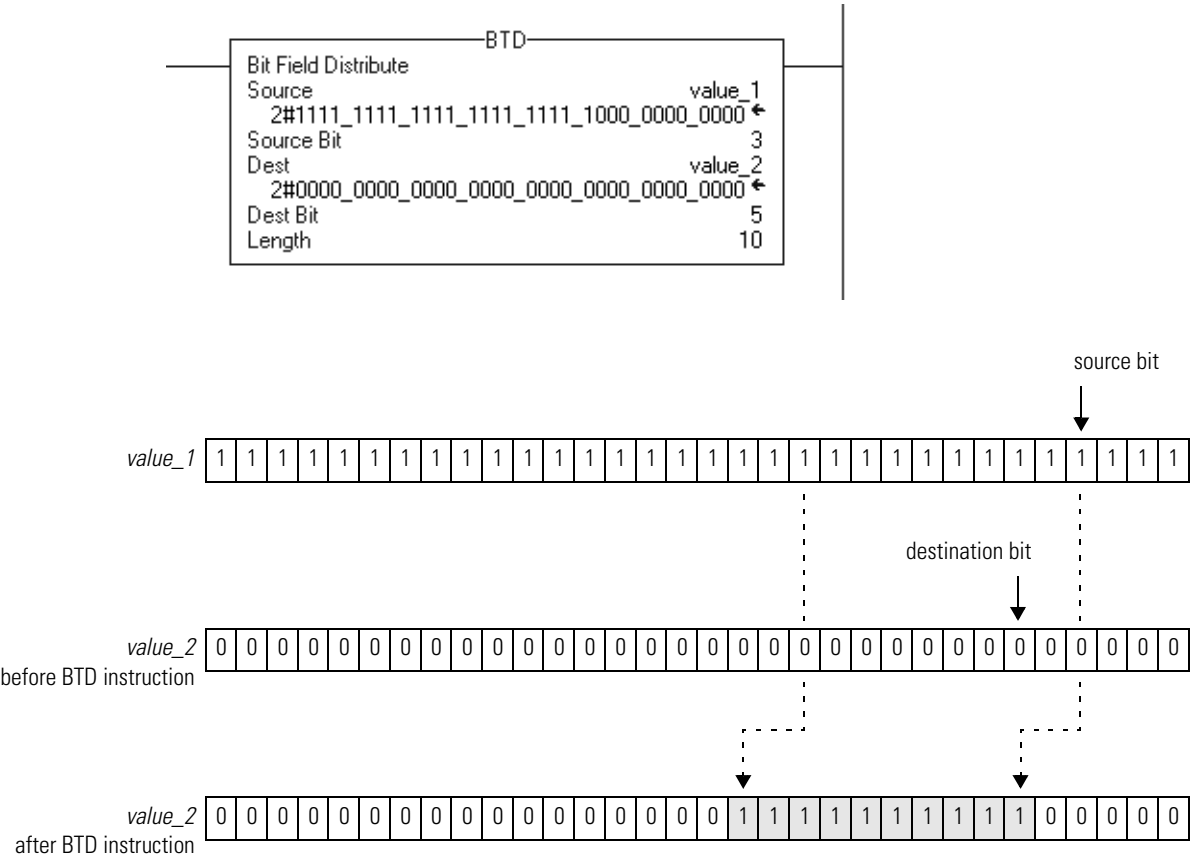


The shaded boxes show the bits that changed in *value_2*.

# Bit Field Distribute with Target (BTDT)

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

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

### Operands:

BTDT(BTDT_tag);

### Structured Text

| Variable | Type | Format | Description |
|---|---|---|---|
| BTDT tag | FBD_BIT_FIELD_DISTRIBUTE | structure | BTDT structure |

### Function Block

BTDT_01

BTDT

Bit Field Distribute with Target

Source          Dest
SourceBit
Length
DestBit
Target
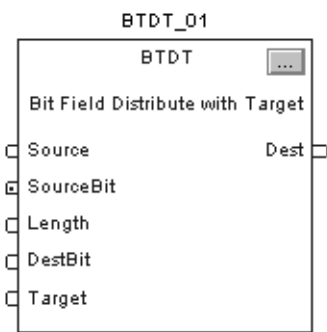
| Operand | Type | Format | Description |
|---|---|---|---|
| BTDT tag | FBD_BIT_FIELD_DISTRIBUTE | structure | BTDT structure |

### FBD_BIT_FIELD_DISTRIBUTE Structure

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

| 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. | EnableIn is always set. |
| | EnableOut is set. | The instruction executes. |
| postscan | No action taken. | No action taken. |

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

*Target*  | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

*Dest*  | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
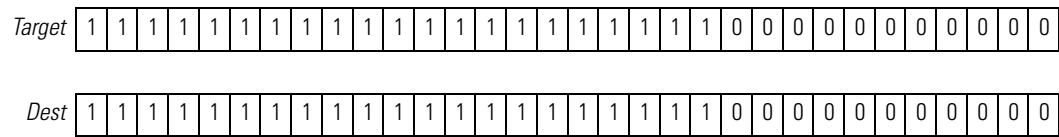
**2.**  The SourceBit and the Length specify which bits in Source to copy into Dest, starting at DestBit. Source and Target remain unchanged.
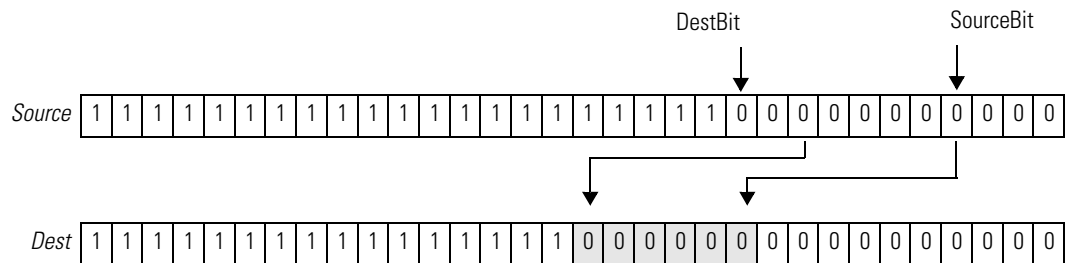


### Structured Text

```
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
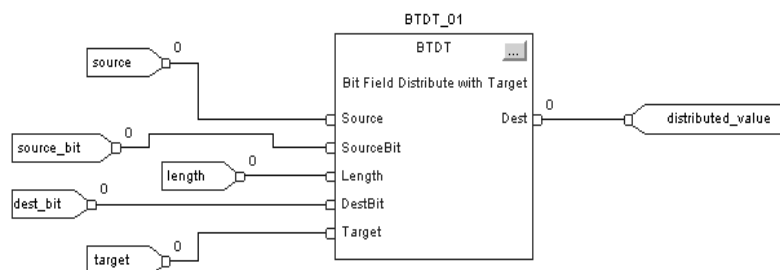
# Clear (CLR)

The CLR instruction clears all the bits of the Destination.

### Operands:
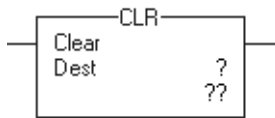


#### Relay Ladder

| 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 <u>Appendix C, Structured Text Programming</u> 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
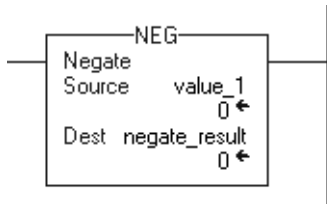
### Execution:

| 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

```
                  ┌──CLR────────┐
                  │Clear        │
──────────────────┤Dest         │
                  │     value   │
                  │   999999 ←  │
                  └─────────────┘
```
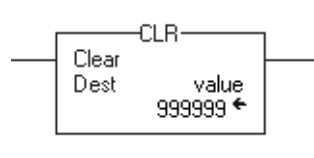
## Structured Text

```
value := 0;
```

# Swap Byte (SWPB)

The SWPB instruction rearranges the bytes of a value.

## Operands:

### Relay Ladder



```
┌─SWPB──────────────┐
│ Swap Byte          │
│ Source          ?  │
│                 ?? │
│ Order Mode      ?  │
│ Dest            ?  │
│                 ?? │
└────────────────────┘
```
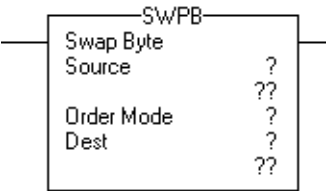
| Operand | Type | Format | Enter |
|---------|------|--------|-------|
| Source | INT | tag | tag that contains the bytes that you want to rearrange |
|  | DINT |  |  |
|  | REAL |  |  |

| Order Mode |  |  | **If the Source Is an** | **And You Want To Change the Bytes To This Pattern (Each Letter Represents a Different Byte)** | **Then Select** |
|---|---|---|---|---|---|
|  |  |  | INT | n/a | any of the options |
|  |  |  | DINT | ABCD ⇒DCBA | REVERSE (or enter 0) |
|  |  |  | REAL | ABCD ⇒CDAB | WORD (or enter 1) |
|  |  |  |  | ABCD ⇒BADC | HIGH/LOW (or enter 2) |

| Destination | INT | tag | tag to store the bytes in the new order | |
|---|---|---|---|---|
|  | DINT |  | **If the Source Is an** | **Then the Destination Must Be an** |
|  | REAL |  | 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.

bar code reader



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

**Arithmetic Status Flags:**    not affected

**Fault Conditions:**    none

**Execution:**

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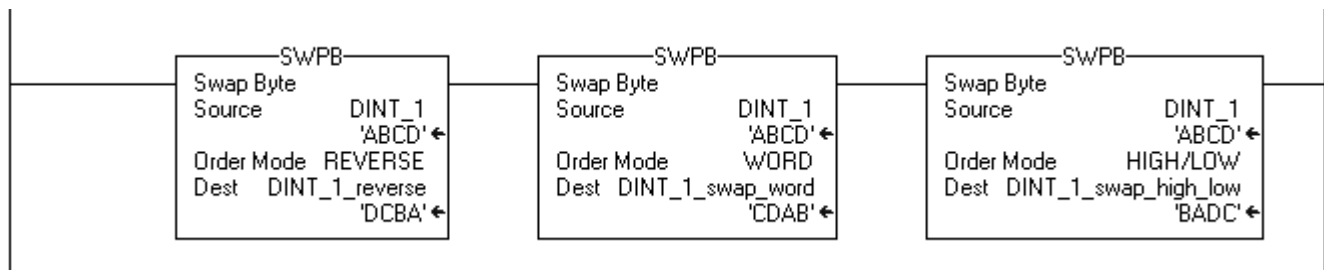


### Structured Text

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

## Relay Ladder

Initialize the tags.

```
                                        ┌CLR─────────┐      ┌SIZE──────────────┐
                                        │Clear       │      │Size in Elements  │
                                        │Dest   index│      │Source    array[0]│
                                        │          0←│      │          '1234'← │
                                        └────────────┘      │Dim. To Vary    0 │
                                                            │Size  array_length│
                                                            │               4← │
                                                            └──────────────────┘
```

Reverse the bytes.

```
 swap_bytes                      ┌SWPB─────────────────────┐   ┌ADD──────────┐
─┤ LBL ├─────────────────────────│Swap Byte                │───│Add          │
                                 │Source         array[index]│  │Source A    1│
                                 │                   '1234' │   │             │
                                 │Order Mode      REVERSE   │   │Source B index│
                                 │Dest array_bytes_reverse[index]│         0← │
                                 │                   '4321' │   │Dest    index │
                                 └──────────────────────────┘   │           0←│
                                                                └─────────────┘
```
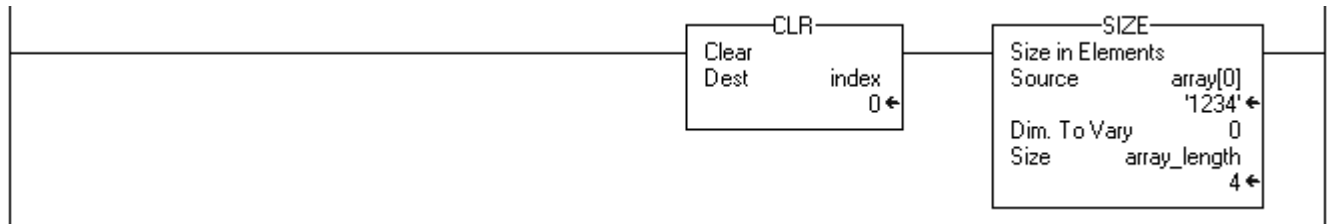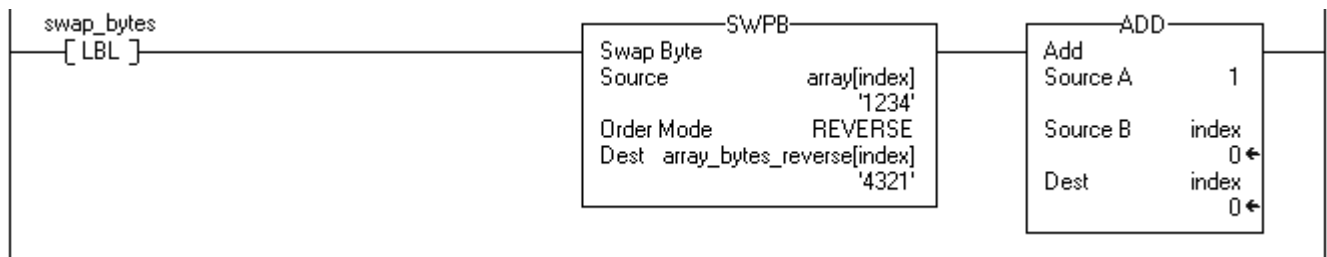
Determine whether the SWPB instruction has acted on all the elements in the array.

```
       ┌LES──────────────┐                                      swap_bytes
       │Less Than (A<B)   │                                      ─⟨JMP⟩─
       │Source A     index│
       │               0← │
       │Source B array_length│
       │               4← │
       └──────────────────┘
```

## 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 317.

### Operands:

### Relay Ladder

| Operand | Type | Format | Description |
|---|---|---|---|
| Source A | SINT | immediate | value to AND 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 AND 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** | | |

### Structured Text

`dest := sourceA AND sourceB`

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

See Appendix C, Structured Text Programming for information on the syntax of expressions within structured text.
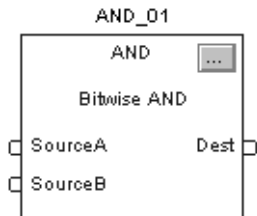
### Function Block

| 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.



*SourceA* | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 1 1 1 1 1 1

*SourceB* | 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

*Dest* | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0

**Relay Ladder**



```
                                    -AND-
          Bitwise AND
          Source A                              value_1
            2#0000_0000_0000_0101_0101_0101_1111_1111 ←
          Source B                              value_2
            2#0000_0000_0000_1111_1111_0000_0000_0000 ←
          Dest                           value_result_and
            2#0000_0000_0000_0101_0101_0000_0000_0000 ←
```
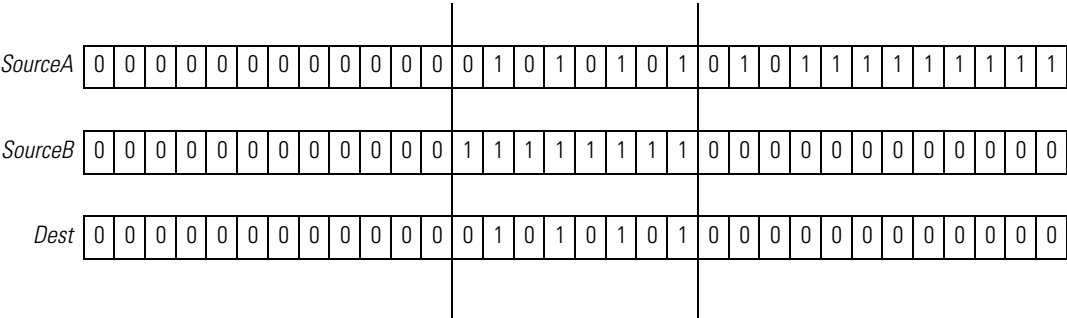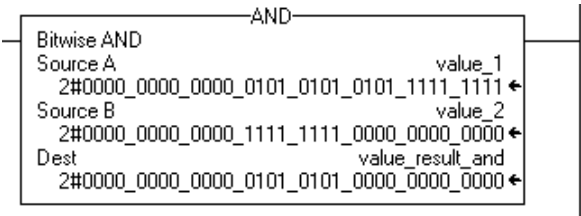
**Structured Text**

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

**Function Block**
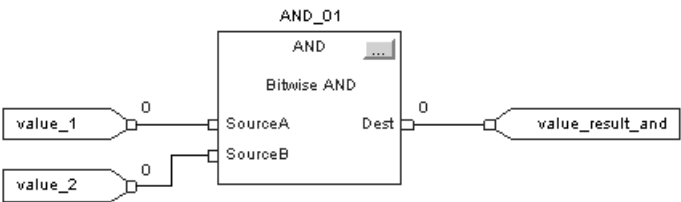
# Bitwise OR (OR)

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 .

### Operands:



### Relay Ladder

| Operand | Type | Format | Description |
|---|---|---|---|
| Source A | SINT | immediate | value to OR 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 OR 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** | | |

### Structured Text



`dest := sourceA OR sourceB`

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

See Appendix C, 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**



```
                         ─OR─
        Bitwise Inclusive OR
        Source A                                  value_1
           2#0000_0000_0000_0101_0101_0101_1111_1111 ←
        Source B                                  value_2
           2#0000_0000_0000_1111_1111_0000_0000_0000 ←
        Dest                                 value_result_or
           2#0000_0000_0000_1111_1111_0101_1111_1111 ←
```

**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 .

## 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 | | |

### Structured Text

```
dest := sourceA XOR sourceB
```

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

See Appendix C, 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 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| *value_2* | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 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 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

**Relay Ladder**

```
                         ┌────────XOR────────┐
                         │ Bitwise Exclusive OR            │
                    ─────┤ Source A                value_1 │─────
                         │   2#0000_0000_0000_0101_0101_0101_1111_1111 ←│
                         │ Source B                value_2 │
                         │   2#0000_0000_0000_1111_1111_0000_0000_0000 ←│
                         │ Dest            value_result_xor │
                         │   2#0000_0000_0000_1010_1010_0101_1111_1111 ←│
                         └───────────────────┘
```
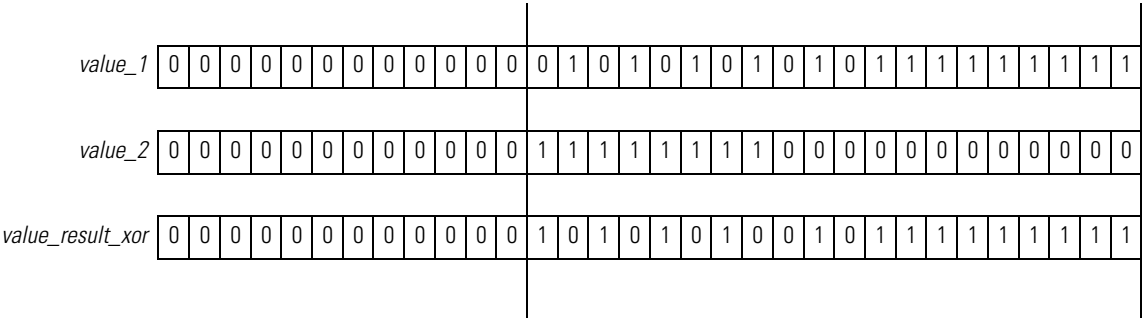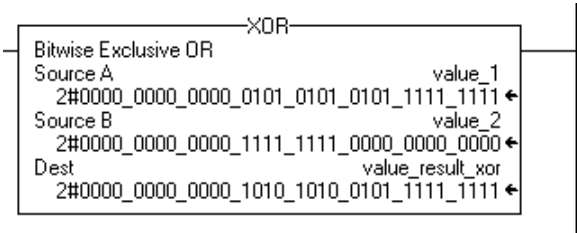
**Structured Text**

```
value_result_xor := value_1 XOR value_2;
```

**Function Block**
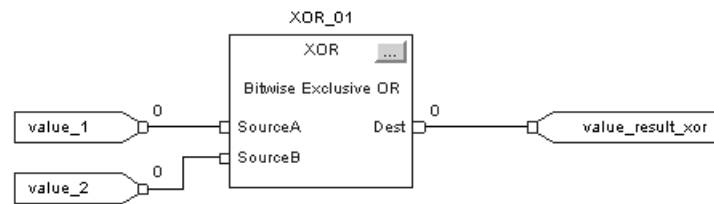
# Bitwise NOT (NOT)

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 326.

## Operands:
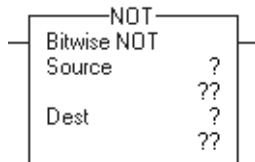
### Relay Ladder



| 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 | | |

### Structured Text

`dest := NOT source`

Use NOT as an operator within an expression. This expression evaluates NOT *source*.

See Appendix C, Structured Text Programming for information on the syntax of expressions within structured text.
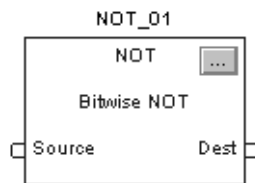
### Function Block



| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| NOT tag | FBD_LOGICAL | structure | NOT structure |

## FBD_LOGICAL Structure

| Input Parameter | Data Type | Description |
|---|---|---|
| EnableIn | BOOL | Enable input. If cleared, the instruction does not execute and outputs are not updated. |
| | | default is set |
| Source | DINT | Value to NOT. |
| | | valid = any integer |

| Output Parameter | Data Type | Description |
|---|---|---|
| EnableOut | BOOL | The instruction produced a valid result. |
| Dest | DINT | Result of the instruction. Arithmetic status flags are set for this output. |

**Description:**  When enabled, the instruction evaluates the NOT operation:

| If the Bit In the Source Is: | The Bit In the Destination Is: |
|---|---|
| 0 | 1 |
| 1 | 0 |

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

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

**Fault Conditions:**  none

**Execution:**

**Relay Ladder**

| Condition | Action |
|---|---|
| prescan | The rung-condition-out is set to false. |
| rung-condition-in is false | The rung-condition-out is set to false. |
| rung-condition-in is true | The instruction performs a bitwise NOT operation. |
| | The rung-condition-out is set to true. |
| postscan | The rung-condition-out is set to false. |

    **Function Block**

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

**Example:** When enabled, the NOT instruction performs a bitwise NOT operation on Source and places the result in Dest.

*value_1*   | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

*value_result_not*   | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Relay Ladder**
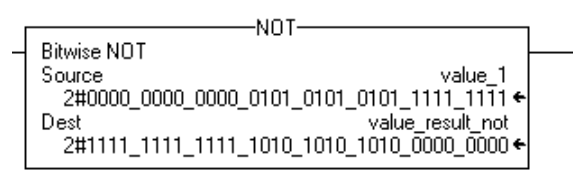


**Structured Text**

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

**Function Block**
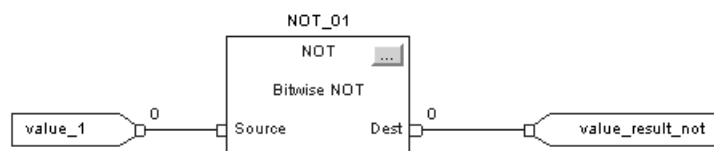
# Boolean AND (BAND)

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

To perform a bitwise AND, see <u>page 303</u>.

## 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.
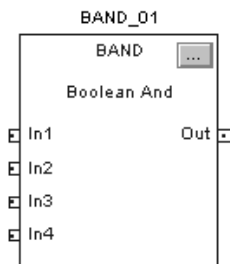
### Function Block



| 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.<br><br>Default is set. |
| In1 | BOOL | First boolean input.<br><br>Default is set. |
| In2 | BOOL | Second boolean input.<br><br>Default is set. |
| In3 | BOOL | Third boolean input.<br><br>Default is set. |
| In4 | BOOL | Fourth boolean input.<br><br>Default is set. |
| In5 | BOOL | Fifth boolean input.<br><br>default is set. |
| In6 | BOOL | Sixth boolean input.<br><br>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:**

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

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

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

**Structured Text**

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

### Function Block
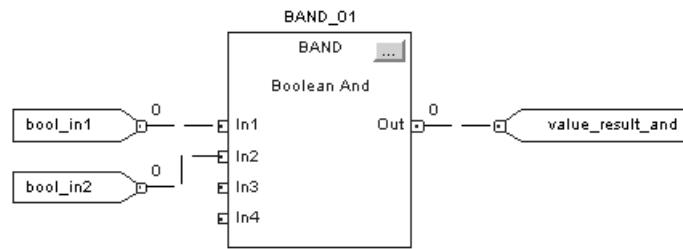


**Example 2:**    If both *bool_in1* and *bool_in2* are set (true), *light1* is set (on). Otherwise, *light1* is cleared (off).

### Structured Text

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

# Boolean OR (BOR)

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

To perform a bitwise OR, see .

### Operands:

## Structured Text

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

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

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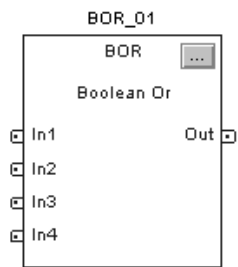
## Function Block

| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| BOR tag | FBD_BOOLEAN_OR | structure | BOR structure |

## FBD_BOOLEAN_OR Structure

| Input Parameter | Data Type | Description |
|-----------------|-----------|-------------|
| EnableIn | BOOL | Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set. |
| In1 | BOOL | First boolean input. Default is cleared. |
| In2 | BOOL | Second boolean input. Default is cleared. |
| In3 | BOOL | Third boolean input. Default is cleared. |
| In4 | BOOL | Fourth boolean input. Default is cleared. |
| In5 | BOOL | Fifth boolean input. Default is cleared. |
| In6 | BOOL | Sixth boolean input. Default is cleared. |
| In7 | BOOL | Seventh boolean input. Default is cleared. |

| Input Parameter | Data Type | Description |
|---|---|---|
| In8 | BOOL | Eighth boolean input. |
| | | Default is cleared. |

| Output Parameter | Data Type | Description |
|---|---|---|
| EnableOut | BOOL | Enable output. |
| Out | BOOL | The output of the instruction. |

**Description:** The BOR instruction ORs as many as eight boolean inputs. If an input is not used, it defaults to cleared (0).

Out = In1 OR In2 OR In3 OR In4 OR In5 OR In6 OR In7 OR In8

**Arithmetic Status Flags:** not affected

**Fault Conditions:** none

**Execution:**

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

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

| If BOOL_IN1 Is | If BOOL_IN2 Is: | Then VALUE_RESULT_OR Is: |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

**Structured Text**

```
value_result_or := bool_in1 OR bool_in2;
```

### Function Block
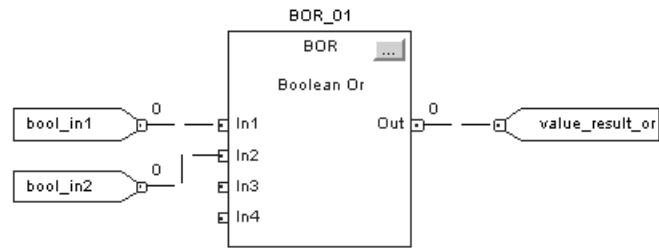


**Example 2:**    In this example, *light1* is set (on) if:

- only *bool_in1* is set (true).
- only *bool_in2* is set (true).
- both *bool_in1* and *bool_in2* are set (true).

Otherwise, *light1* is cleared (off).

### Structured Text

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

## Boolean Exclusive OR (BXOR)

The BXOR performs an exclusive OR on two boolean inputs.

To perform a bitwise XOR, see page 310.

### Operands:

#### Structured Text

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

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

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

#### Function Block

```
BXOR_01
    BXOR        [...]
Boolean Exclusive Or
In1          Out
In2
```
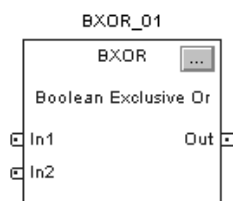
| Operand | Type | Format | Description |
|---|---|---|---|
| BXOR tag | FBD_BOOLEAN_XOR | structure | BXOR structure |

#### FBD_BOOLEAN_XOR Structure

| Input Parameter | Data Type | Description |
|---|---|---|
| EnableIn | BOOL | Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set. |
| In1 | BOOL | First boolean input. Default is cleared. |
| In2 | BOOL | Second boolean input. Default is cleared. |
| **Output Parameter** | **Data Type** | **Description** |
| EnableOut | BOOL | Enable output. |
| Out | BOOL | The output of the instruction. |

**Description:** The BXOR instruction performs an exclusive OR on two boolean inputs.

Out = In1 XOR In2

**Arithmetic Status Flags:** not affected

**Fault Conditions:** none

**Execution:**

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

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

| If BOOL_IN1 Is | If BOOL_IN2 Is | Then VALUE_RESULT_XOR Is |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

**Structured Text**

```
value_result_xor := bool_in1 XOR bool_in2;
```

**Function Block**
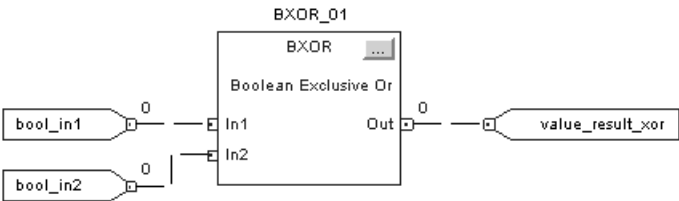
**Example 2:**   In this example, *light1* is set (on) if

- only *bool_in1* is set (true).
- only *bool_in2* is set (true).

Otherwise, *light1* is cleared (off).

### Structured Text

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

# Boolean NOT (BNOT)

The BNOT instruction complements a boolean input.

To perform a bitwise NOT, see page 314.

### Operands:

### Structured Text

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

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 Appendix C, Structured Text Programming for information on the syntax of expressions within structured text.
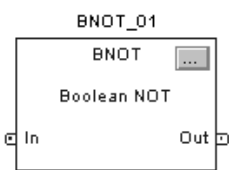
### Function Block

| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| BNOT tag | FBD_BOOLEAN_NOT | structure | BNOT structure |

### FBD_BOOLEAN_NOT Structure

| Input Parameter | Data Type | Description |
|-----------------|-----------|-------------|
| EnableIn | BOOL | Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set. |
| In | BOOL | Input to the instruction. Default is set. |

| Output Parameter | Data Type | Description: |
|------------------|-----------|-------------|
| EnableOut | BOOL | Enable output. |
| Out | BOOL | The output of the instruction. |

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

Out = NOT In

**Arithmetic Status Flags:**  not affected

**Fault Conditions:**  none

**Execution:**

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

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

| If BOOL_IN1 Is | Then VALUE_RESULT_NOT Is |
|---|---|
| 0 | 1 |
| 1 | 0 |

**Structured Text**

```
value_result_not := NOT bool_in1;
```

**Function Block**
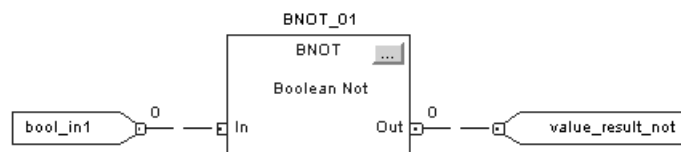


**Example 2:**    If *bool_in1* is cleared, *light1* is cleared (off). Otherwise, *light1* is set (on).

**Structured Text**

```
IF NOT bool_in1 THEN
    light1 := 0;
ELSE
    light1 := 1;
END_IF;
```

**Notes:**

# Array (File)/Misc. Instructions

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

## 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<br><br>structured text[1] | 335 |
| search for and compare values in arrays | FSC | relay ladder | 346 |
| copy the contents of one array into another array | COP | relay ladder<br><br>structured text | 355 |
| copy the contents of one array into another array without interruption | CPS | relay ladder<br><br>structured text | 355 |
| fill an array with specific data | FLL | relay ladder<br><br>structured text[1] | 361 |
| calculate the average of an array of values | AVE | relay ladder<br><br>structured text[1] | 365 |
| sort one dimension of array data into ascending order | SRT | relay ladder<br><br>structured text | 370 |
| calculate the standard deviation of an array of values | STD | relay ladder<br><br>structured text[1] | 375 |
| find the size of a dimension of an array | SIZE | relay ladder<br><br>structured text | 381 |

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

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

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

## Selecting Mode of Operation

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

| If You Want To | Select This Mode |
|---|---|
| operate on all of the specified elements in an array before continuing on to the next instruction | All |
| distribute array operation over a number of scans<br><br>enter the number of elements to operate on per scan (1-2147483647) | Numerical |
| manipulate one element of the array each time the rung-condition-in goes from false to true | Incremental |

### All mode

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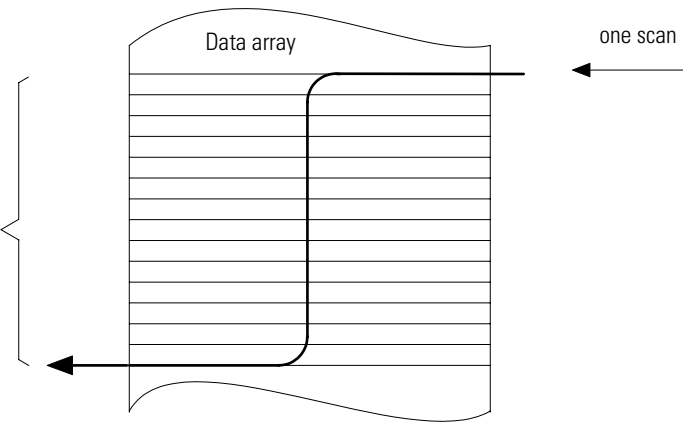


16639

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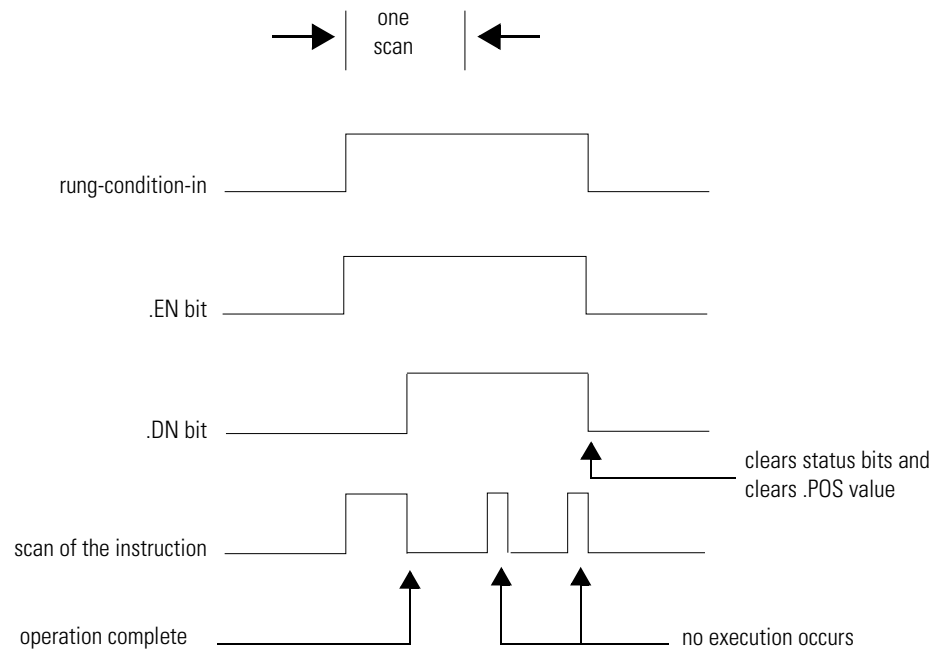


## Numerical mode

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.



one scan

second scan

next scan

16641
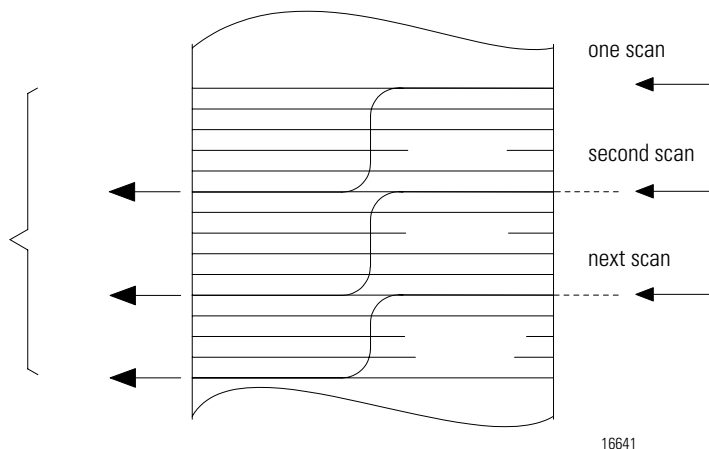
| IMPORTANT | Avoid using the results of a file instruction operating in numerical mode until the .DN bit is set. |
|---|---|

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



**rung is true at completion**

multiple scans

rung-condition-in

.EN bit

.DN bit

scan of the instruction

operation complete

clears status bits and clears .POS value

**rung is false at completion**

multiple scans

operation complete

40013

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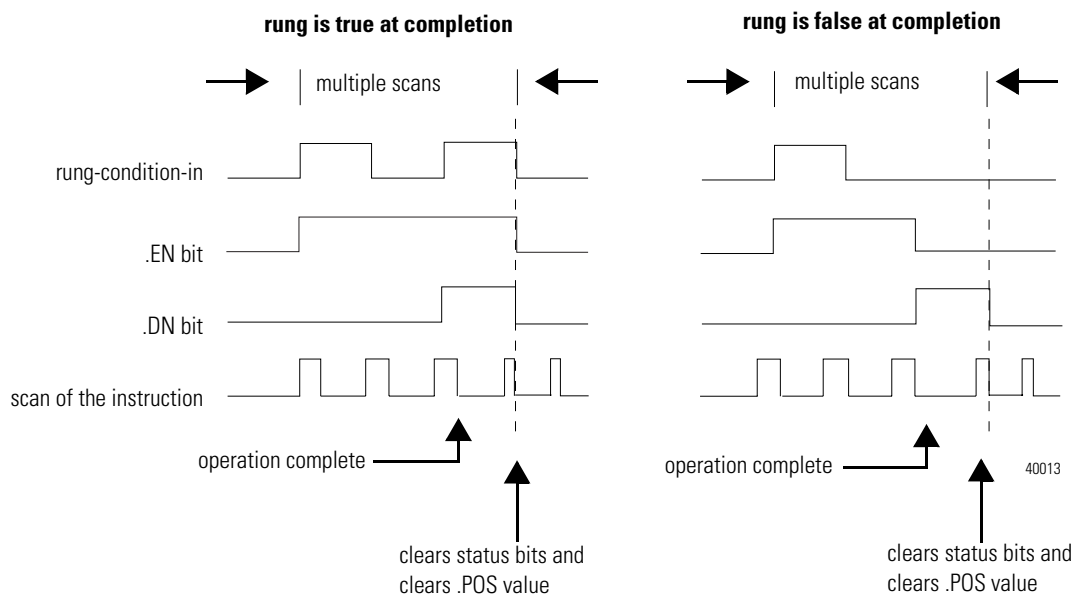
If the rung-condition-in is true at completion, the .EN and .DN bit are set until the rung-condition-in goes false. When the rung-condition-in goes false, these bits are cleared and the .POS value is cleared.

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.
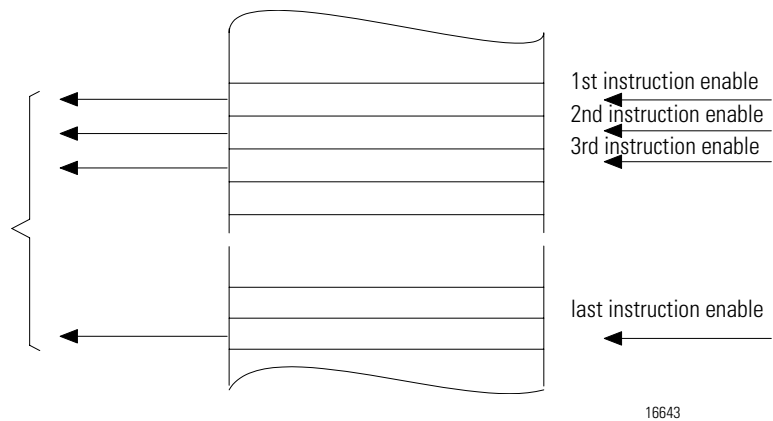


1st instruction enable
2nd instruction enable
3rd instruction enable

last instruction enable

16643

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.
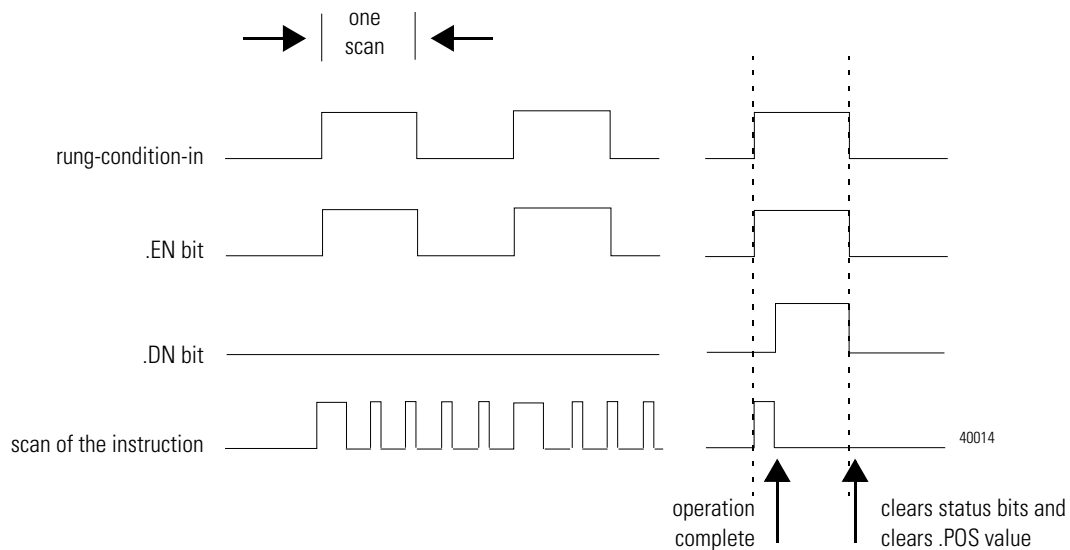


The .EN bit is set when rung-condition-in is true. The .DN bit is set when the last element in the array has been manipulated. When the last element has been manipulated and the rung-condition-in goes false, the .EN bit, the .DN bit, and the .POS value are cleared.

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:
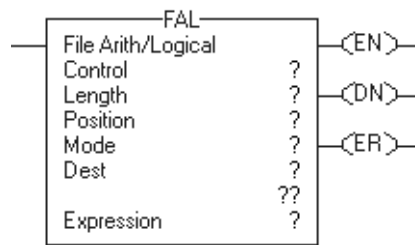
### Relay Ladder

| 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<br><br>initial value is typically 0 |
| Mode | DINT | immediate | how to distribute the operation<br><br>select INC, ALL, or enter a number |
| Destination | SINT | tag | tag to store the result |
|  | INT |  |  |
|  | **DINT** |  |  |
|  | **REAL** |  |  |
| Expression | SINT | immediate | an expression consisting of tags and/or immediate values separated by operators |
|  | INT | tag |  |
|  | **DINT** |  |  |
|  | **REAL** |  |  |
|  | A SINT or INT tag converts to a DINT value by sign-extension. | | |

### Structured Text

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

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

See Appendix C, Structured Text Programming for information on the syntax of constructs within structured text.

## CONTROL Structure

| Mnemonic | Data Type | Description |
|---|---|---|
| .EN | BOOL | The enable bit indicates that the FAL instruction is enabled. |
| .DN | BOOL | The done bit is set when the instruction has operated on the last element (.POS = .LEN). |
| .ER | BOOL | The error bit is set if the expression generates an overflow (S:V is set). The instruction stops executing until the program clears the .ER bit. The .POS value contains the position of the element that caused the overflow. |
| .LEN | DINT | The length specifies the number of elements in the array on which the FAL instruction operates. |
| .POS | DINT | The position contains the position of the current element that the instruction is accessing. |

**Description:** The FAL instruction performs the same operations on arrays as the CPT instruction performs on elements.

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

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

**Fault Conditions:**

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

**Execution:**

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

rung-condition-in is false



page 341

| Condition | Relay Ladder Action |
|---|---|

rung-condition-in is true

```
examine .ER bit  --.ER bit = 0-->  examine .DN bit  --.DN bit = 0-->  .LEN < 0 or .POS < 0  --yes--> major fault
     |                                    |                                   |
  .ER bit = 1                        .DN bit = 1                             no
     |                                    |                                   |
     v                                    v                            .LEN = 0  --no--> INC mode --no--> ALL mode --no--> numeric mode (page 341)
                                                                            |                |                |
                                                                           yes              yes              yes
                                                                            |                |                |
                                     .DN bit is set                         v              INC mode          ALL mode
                                                                                          (page 339)         (page 340)
```

common

loop_count = loop_count - 1

loop_count < 0  --no--> .POS = .POS + 1 --> evaluate expression --> examine S:V --no-->

yes / examine S:V --yes--> .ER bit is set

.POS = .POS + 1

.POS = .LEN  --no--> .POS = .POS + 1

yes

.DN bit is set

rung-condition-out is set to **true**

end

| Condition | Relay Ladder Action |
|---|---|

INC mode

examine .EN bit

.EN bit = 1

.EN bit = 0

examine *internal bit*

*bit* = 1

.POS = .POS + 1

*bit* = 0

*internal bit* is set

.POS ≥ .LEN

yes

.POS = 0

yes

no

no

.EN bit is set

*loop_count* = 1

.POS = .POS - 1

.DN bit is set

common

rung-condition-out is set to **true**

end

| Condition | Relay Ladder Action |
|-----------|---------------------|

ALL mode

examine .EN bit

.EN bit = 0

.EN bit = 1

examine *internal bit*

*bit* = 1

*bit* = 0

.POS = .POS + 1

.EN bit is set

.POS ≥ .LEN

yes

no

.POS = 0

yes

no

*loop_count* = .LEN - .POS

.POS = .POS - 1

common

[page 338](#)

.DN bit is set

rung-condition-out is set to **true**

end

| Condition | Relay Ladder Action |
|---|---|

**numeric mode**

examine .EN bit
- .EN bit = 1 → .POS = .POS + 1
- .EN bit = 0 → examine *internal bit*

examine *internal bit*
- *bit* = 1 → rung-condition-out is set to **true**
- *bit* = 0 → *internal bit* is set

*internal bit* is set → .POS ≥ .LEN
- yes → .POS = 0
  - yes → .DN bit is set
  - no → .POS = .POS - 1 → .DN bit is set
- no → .LEN ≥ mode
  - no → *mode* = .LEN
  - yes → .EN bit is set / *loop_count* = .LEN - .POS

.DN bit is set → rung-condition-out is set to **true** → **end**

.EN bit is set
*loop_count* = .LEN - .POS

mode ≥ *loop_count*
- yes → **common**
- no → .EN bit is set → common

| postscan | The rung-condition-out is set to false. |
|---|---|

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

**array-to-array copy**

```
------FAL------------------------
File Arith/Logical              ----(EN)---
Control              control_2
Length                    10 ←  ----(DN)---
Position                   0 ←
Mode                     ALL   ----(ER)---
Dest       array_1[control_2.pos]
                         0.0 ←
Expression  array_2[control_2.pos]
```

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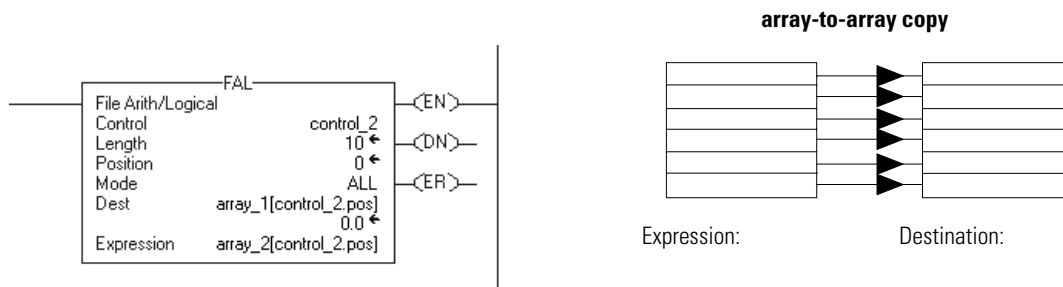
**Example 2:**   When enabled, the FAL instruction copies *value_1* into the first 10 positions of the second dimension of *array_2*.

**element-to-array copy**

```
------FAL------------------------
File Arith/Logical              ----(EN)---
Control              control_2
Length                    10 ←  ----(DN)---
Position                   0 ←
Mode                     ALL   ----(ER)---
Dest       array_2[0,control_2.pos]
                           0 ←
Expression              value_1
```

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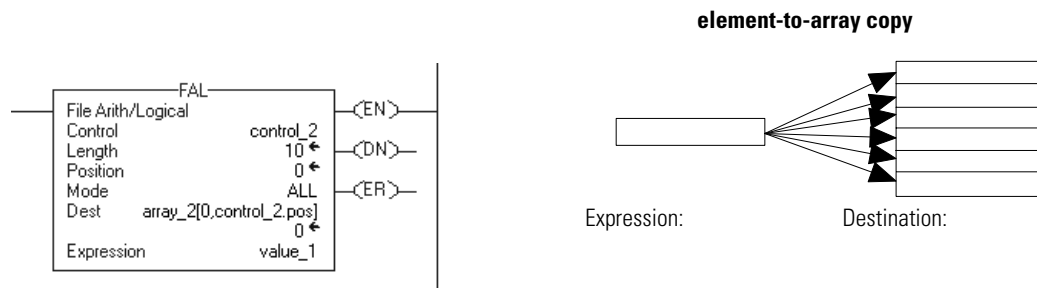
**Example 3:**   Each time the FAL instruction is enabled, it copies the current value of *array_1* to *value_1*. The FAL instruction uses incremental mode, so only one array value is copied each time the instruction is enabled. The next time the instruction is enabled, the instruction overwrites *value_1*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

**array-to-element copy**

```
------FAL------------------------
File Arith/Logical              ----(EN)---
Control              control_1
Length                    10 ←  ----(DN)---
Position                   0 ←
Mode                     inc   ----(ER)---
Dest                  value_1
                           0 ←
Expression  array_1[control_1.pos]
```

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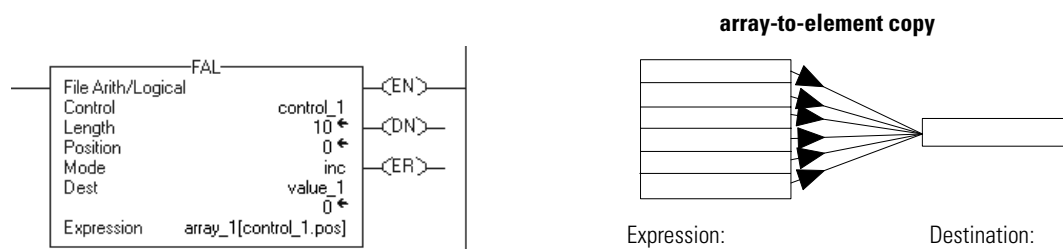
**Example 4:**   When enabled, the FAL instruction adds *value_1* and *value_2* and stores the result in the current position of *array_1*.

**arithmetic operation: (element + element) to array**

```
------FAL------------------------
File Arith/Logical              ----(EN)---
Control              control_1
Length                    10 ←  ----(DN)---
Position                   0 ←
Mode                     all   ----(ER)---
Dest       array_1[control_1.pos]
                           0 ←
Expression     value_1 + value_2
```

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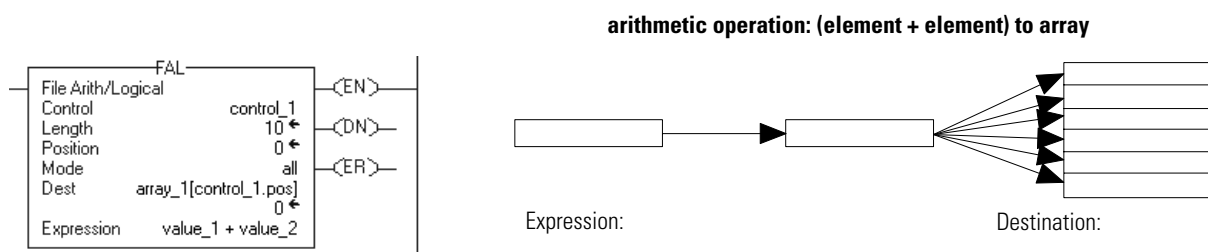
**Example 5:**   When enabled, the FAL instruction divides the value in the current position of *array_2* with the value in the current position of *array_3* and stores the result in the current position of *array_1*.

**arithmetic operation: (array / array) to array**

```
------------FAL------------
File Arith/Logical              ⟨EN⟩
Control                control_2
Length                      10 ←   ⟨DN⟩
Position                     0 ←
Mode                       all   ⟨ER⟩
Dest           array_1[control_2.pos]
                          0.0 ←
Expression  array_2[control_2.pos] / array_3[control_2.pos]
```

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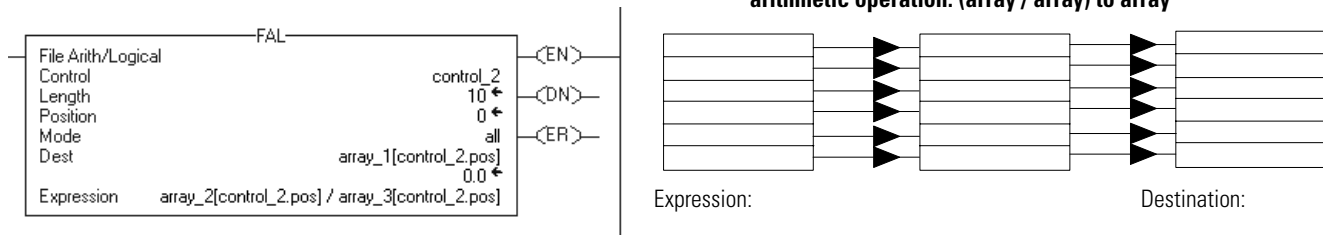
**Example 6:**   When enabled, the FAL instruction adds the value at the current position in *array_1* to *value_1* and stores the result in the current position in *array_3*. The instruction must execute 10 times for the entire *array_1* and *array_3* to be manipulated.

**arithmetic operation: (array + element) to array**

```
------------FAL------------
File Arith/Logical              ⟨EN⟩
Control                control_1
Length                      60 ←   ⟨DN⟩
Position                     0 ←
Mode                         6   ⟨ER⟩
Dest           array_3[control_1.pos]
                            0 ←
Expression  (array_1[control_1.pos]+value_1)
```

Expression:                          Destination:
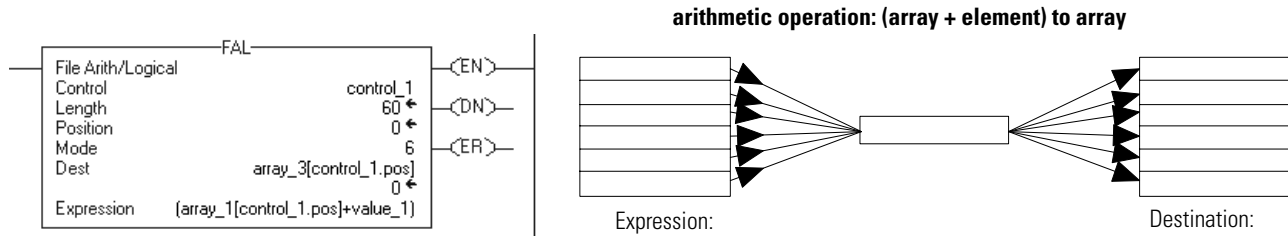
**Example 7:**   Each time the FAL instruction is enabled, it adds *value_1* to the current value of *array_1* and stores the result in *value_2*. The FAL instruction uses incremental mode, so only one array value is added to *value_1* each time the instruction is enabled. The next time the instruction is enabled, the instruction overwrites *value_2*.

**arithmetic operation: (element + array) to element**

```
------------FAL------------
File Arith/Logical              ⟨EN⟩
Control                control_1
Length                      10 ←   ⟨DN⟩
Position                     0 ←
Mode                       inc   ⟨ER⟩
Dest                     value_2
                            0 ←
Expression  value_1 + array_1[control_1.pos]
```

Expression:                          Destination:
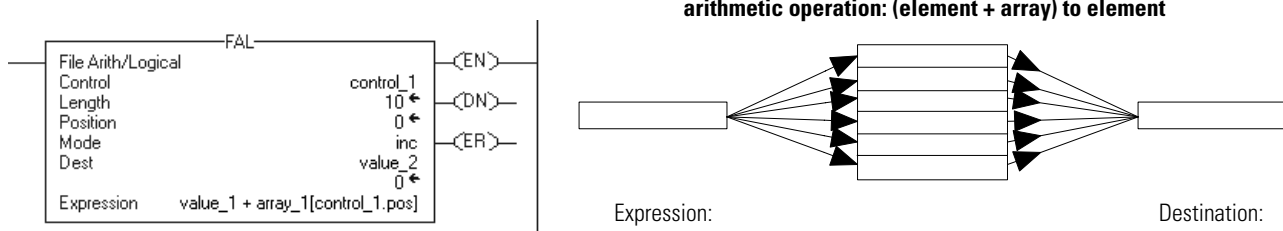
**Example 8:**    When enabled, the FAL instruction multiplies the current value of *array_1* by the current value of *array_3* and stores the result in *value_1*. The FAL instruction uses incremental mode, so only one pair of array values is multiplied each time the instruction is enabled. The next time the instruction is enabled, the instruction overwrites *value_1*.

**arithmetic operation: (array ∗ array) to element**

```
┌──────────────FAL───────────────────┐
│  File Arith/Logical                 │──(EN)──
│  Control                  control_1 │
│  Length                       10 ←  │──(DN)──
│  Position                      0 ←  │
│  Mode                        inc    │──(ER)──
│  Dest                     value_1   │
│                                0 ←  │
│  Expression   array_1[control_1.pos] * array_3[control_1.pos]  │
└─────────────────────────────────────┘
```

Expression:                                            Destination:
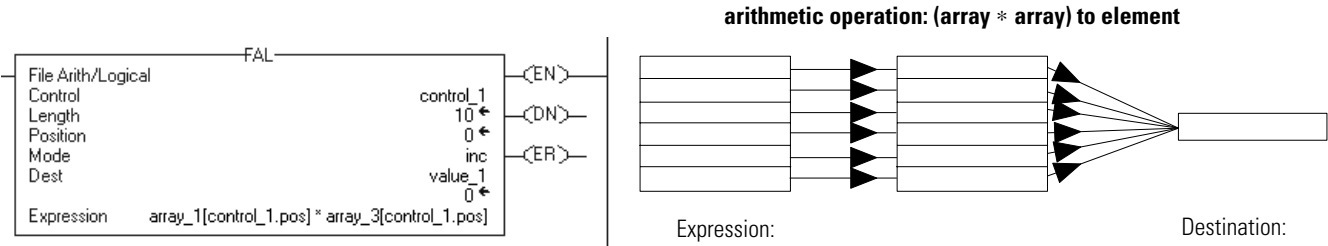
## FAL Expressions

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(*tag_a*) |
| two operands | operand_a operator operand_b | • *tag_b* + 5<br>• *tag_c* AND *tag_d*<br>• (*tag_e* ** 2) MOD (*tag_f* / *tag_g*) |

## Determine the order of operation

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

Operations of equal order are performed from left to right.

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

# File Search and Compare (FSC)

The FSC instruction compares values in an array, element by element.

## Operands:
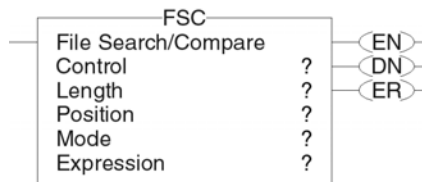
### Relay Ladder

```
        ┌─────FSC─────────────┐
        │ File Search/Compare │──(EN)─
────────│ Control          ?  │──(DN)─
        │ Length           ?  │──(ER)─
        │ Position         ?  │
        │ Mode             ?  │
        │ Expression       ?  │
        └─────────────────────┘
```

| 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<br><br>initial value is typically 0 |

### CONTROL Structure

| Mnemonic | Data Type | Description |
|----------|-----------|-------------|
| .EN | BOOL | The enable bit indicates that the FSC instruction is enabled. |
| .DN | BOOL | The done bit is set when the instruction has operated on the last element (.POS = .LEN). |
| .ER | BOOL | The error bit is not modified. |
| .IN | BOOL | The inhibit bit indicates that the FSC instruction detected a true comparison. You must clear this bit to continue the search operation. |
| .FD | BOOL | The found bit indicates that the FSC instruction detected a true comparison. |
| .LEN | DINT | The length specifies the number of elements in the array on which the instruction operates. |
| .POS | DINT | The position contains the position of the current element that the instruction is accessing. |

**Description:** When the FSC instruction is enabled and the comparison is true, the instruction sets the .FD bit and the .POS bit reflects the array position where the instruction found the true comparison. The instruction sets the .IN bit to prevent further searching.

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

**Fault Conditions:**

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

**Execution:**

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

rung-condition-in is false

examine .DN bit

.DN bit = 0

.DN bit = 1

INC mode

no

*internal bit* is cleared

yes

.EN bit is cleared

.EN bit is cleared
.ER bit is cleared

ALL mode

no

.LEN < 0 or .POS < 0

yes

major fault

yes

no

.POS = .POS + 1

.LEN = 0

yes

no

.POS = 0

yes

no

.POS < .LEN

no

yes

.POS = .POS - 1

.LEN > *mode*

no

yes

.DN bit is set

*mode* = .LEN

.DN bit is set

numeric mode

rung-condition-out is set to **false**

end

| Condition | Relay Ladder Action |
|---|---|
| rung-condition-in is true | |



| postscan | The rung-condition-out is set to false. |
|---|---|

**Example 1:**   Search for a match between two arrays. When enabled, the FSC instruction compares each of the first 10 elements in *array_1* to the corresponding elements in *array_2*.

```
                            ┌──────FSC──────────┐
                            │ File Search/Compare        │──(EN)──
                            │ Control        control_3   │──(DN)──
                            │ Length              10 ◆   │──(ER)──
                            │ Position             0 ◆   │
                            │ Mode               ALL     │
                            │ Expression  array_1[control_3.pos] <> array_2[control_3.pos] │
                            └────────────────────┘
```
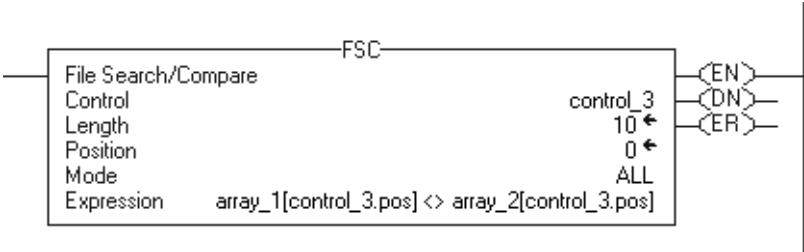
| *array_1* | *array_2* | *control_3.pos* |
|---|---|---|
| 00000000000000000000000000000000 | 00000000000000000000000000000000 | 0 |
| 00000000000000000000000000000000 | 00000000000000000000000000000000 | 1 |
| 00000000000000000000000000000000 | 00000000000000000000000000000000 | 2 |
| 00000000000000000000000000000000 | 00000000000000000000000000000000 | 3 |
| 00000000000000001111111111111111 | 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*.

```
                        ─────FSC─────
            File Search/Compare                          ─(EN)─
            Control                          control_1    ─(DN)──
            Length                                 10 ←   ─(ER)──
            Position                                0 ←
            Mode                                   all
            Expression   MySearchKey = reference[control_1.pos]
```

*MySearchKey*                   *reference*                 *control_3.pos*

|  |  |
|---|---|
| 0000000000000000000000000000000 | 0 |
| 0000000000000000000000000000000 | 1 |
| 0000000000000000000000000000000 | 2 |
| 0000000000000000000000000000000 | 3 |
| 1111111111111110000000000000000 | 4 |
| 1111111111111111111111111111111 | 5 |
| 1111111111111111111111111111111 | 6 |
| 1111111111111111111111111111111 | 7 |
| 1111111111111111111111111111111 | 8 |
| 1111111111111111111111111111111 | 9 |

1111111111111110000000000000000

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

**Example 3:**   Search for a string in an array of strings. When enabled, the FSC instruction compares the characters in *code* to 10 elements in *code_table*.

```
┌─────────────────FSC──────────────────┐
│ File Search/Compare                    │──⟨EN⟩──
│ Control                  code_table_search│──⟨DN⟩──
│ Length                         10 ←     │──⟨ER⟩──
│ Position                        0 ←     │
│ Mode                           ALL      │
│ Expression    code = code_table[code_table_search.POS] │
└────────────────────────────────────────┘
```

*code*

| SAM |
| --- |

*code_table*

| | *code_table_search.POS* |
| --- | --- |
| AFG | 0 |
| BEH | 1 |
| HUO | 2 |
| SAK | 3 |
| SAM | 4 |
| FQG | 5 |
| CLE | 6 |
| CAK | 7 |
| DET | 8 |
| BWG | 9 |

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

## FSC expressions

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

# Valid Operators

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

| 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><li>*tag_b* + 5</li><li>*tag_c* AND *tag_d*</li><li>(*tag_e* ** 2) MOD (*tag_f* / *tag_g*)</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 |

lesser ↑    greater ↓

AB < B

a > B

# Copy File (COP) Synchronous Copy File (CPS)

The COP and CPS instructions copy the value(s) in the Source to the Destination. The Source remains unchanged.
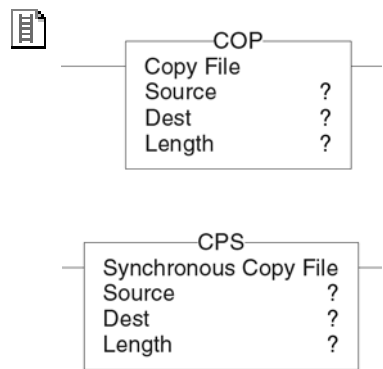
**Operands:**



## Relay Ladder

| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| Source | **SINT** | tag | initial element to copy |
| | **INT** | | **Important:** the Source and Destination operands should be the same data type, or unexpected results may occur |
| | **DINT** | | |
| | **REAL** | | |
| | **string** | | |
| | structure | | |
| Destination | **SINT** | tag | initial element to be overwritten by the Source |
| | **INT** | | |
| | **DINT** | | **Important:** the Source and Destination operands should be the same data type, or unexpected results may occur |
| | **REAL** | | |
| | **string** | | |
| | structure | | |
| Length | DINT | immediate | number of Destination elements to copy |
| | | tag | |

## Structured Text



```
COP(Source,Dest,Length);

CPS(Source,Dest,Length);
```

The operands are the same as those for the relay ladder COP and CPS instructions.

**Description:**   During execution of the COP and CPS instructions, other controller actions may try to interrupt the copy operation and change the source or destination data:

| If the Source Or Destination Is | And You Want To | Then Select | Notes |
|---|---|---|---|
| • produced tag<br>• consumed tag<br>• I/O data<br>• data that another task can overwrite | prevent the data from changing during the copy operation | CPS | • Tasks that attempt to interrupt a CPS instruction are delayed until the instruction is done.<br>• To estimate the execution time of the CPS instruction, see *ControlLogix System User Manual*, publication 1756-UM001. |
| | allow the data to change during the copy operation | COP | |
| none of the above | →————————————————→ | COP | |

The number of bytes copied is:

Byte Count = Length ∗ (number of bytes in the Destination data type)

| **ATTENTION** | If the byte count is greater than the length of the Source, unpredictable data is copied for the remaining elements. |
|---|---|

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

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

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

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

**Arithmetic Status Flags:**   not affected

**Fault Conditions:**   none

**Execution:**

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



| 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

```
              ┌──COP──────────────┐
              │Copy File          │
              │Source   array_4[0]│
              │Dest     array_5[0]│
              │Length          10 │
              └───────────────────┘
```
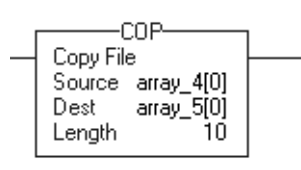
### Structured Text

```
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

```
              ┌──COP──────────────┐
              │Copy File          │
              │Source      timer_1│
              │Dest   array_timer[5]│
              │Length           1 │
              └───────────────────┘
```
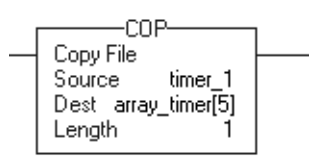
### Structured Text

```
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

```
           CPS
  Synchronous Copy File
  Source    project_data[0]
  Dest   produced_array[0]
  Length                 100
```
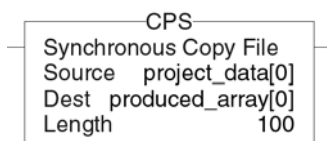
### Structured Text

```
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

```
           CPS
  Synchronous Copy File
  Source   Local:0:I.Data[0]
  Dest        input_buffer[0]
  Length                   20
```
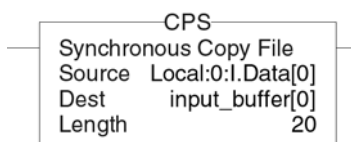
### Structured Text

```
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.

| | |
|---|---|
| array_timer[0] | First the instruction copies timer[0] values to timer[1] |
| array_timer[1] | Then the instruction copies timer[1] values to timer[2] |
| array_timer[2] | Then the instruction copies timer[2] values to timer[3] |
| array_timer[3] | Then the instruction copies timer[3] values to timer[4] |
| array_timer[4] | |
| • | |
| • | |
| • | |
| array_timer[9] | Finally, the instruction copies timer[9] values to timer[10] |
| array_timer[10] | |
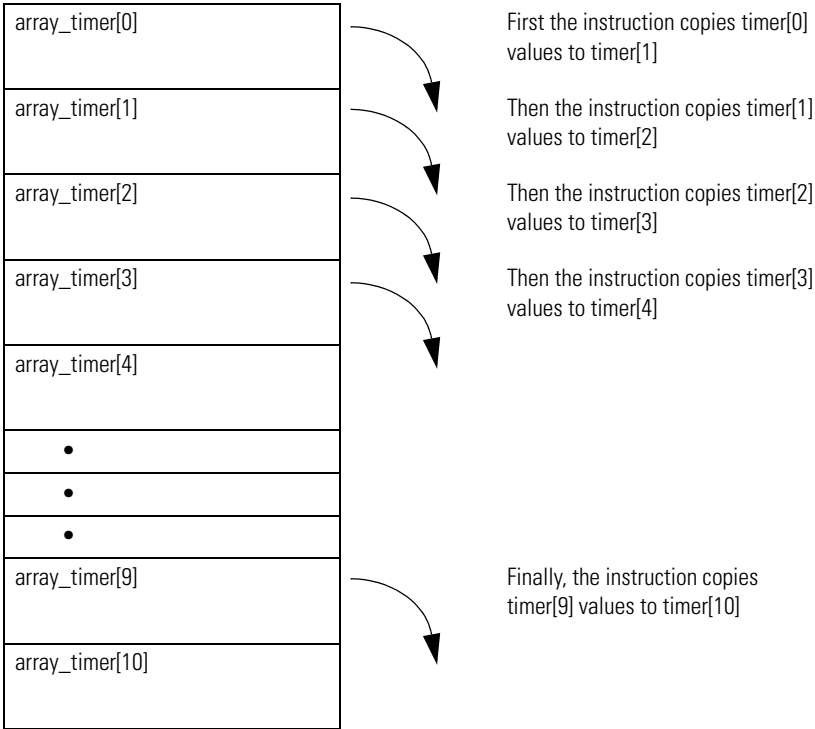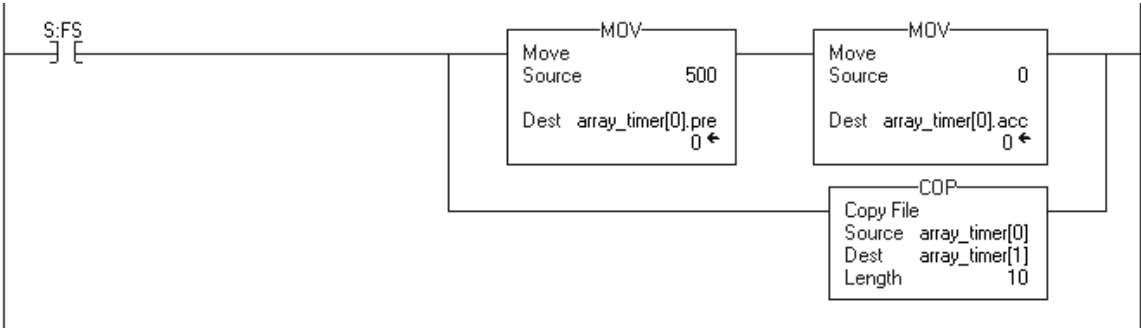
**Relay Ladder**



**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:



```
         ┌─FLL────────┐
    ─────┤ Fill File  ├─────
         │ Source   ? │
         │ Dest     ? │
         │ Length   ? │
         └────────────┘
```
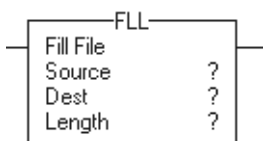
### Relay Ladder

| Operand | Type | Format: | Description |
|---|---|---|---|
| Source | **SINT** | immediate | element to copy |
| | **INT** | tag | **Important:** the Source and Destination operands should be the same data type, or unexpected results may occur |
| | **DINT** | | |
| | **REAL** | | |
| Destination | **SINT** | tag | initial element to be overwritten by the Source |
| | **INT** | | |
| | **DINT** | | **Important:** the Source and Destination operands should be the same data type, or unexpected results may occur |
| | **REAL** | | |
| | 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 <u>Appendix C, Structured Text Programming</u> for information on the syntax of constructs within structured text.

**Description:** The number of bytes filled is:

Byte count = Length * (number of bytes in the Destination data type)

---

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

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

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

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

---

For best results, the Source and Destination should be the same type. If you want to fill a structure, use the COP instruction (see example 3 on page 359). If you mix data types for the Source and Destination, the Destination elements are filled with converted Source values.

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

**Arithmetic Status Flags:**  not affected

**Fault Conditions:**  none

**Execution:**

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

rung-condition-in is true

$end\_address = start\_address + (Length *$ number of bytes in a destination element)

$end\_address >$ end of destination array

yes

$end\_address =$ end of destination array

no

$source\_address =$ Source

$destination\_address = end\_address$

yes

no

copy data in $source\_address$ to $destination\_address$

$destination\_address = destination\_address + 1$

rung-condition-out is set to **true**

end

| postscan | The rung-condition-out is set to false. |

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

**Relay Ladder**
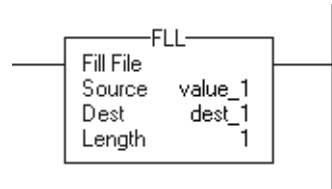


| Source (*value_1*) Data Type | Source (*value_1*) Value | Destination (*dest_1*) Data Type | Destination (*dest_1*) Value After FLL |
|---|---|---|---|
| SINT | 16#80 (-128) | DINT | 16#FFFF FF80 (-128) |
| DINT | 16#1234 5678 | SINT | 16#78 |
| SINT | 16#01 | REAL | 1.0 |
| REAL | 2.0 | INT | 16#0002 |
| SINT | 16#01 | TIMER | 16#0101 0101 16#0101 0101 16#0101 0101 |
| INT | 16#0001 | TIMER | 16#0001 0001 16#0001 0001 16#0001 0001 |
| DINT | 16#0000 0001 | TIMER | 16#0000 0001 16#0000 0001 16#0000 0001 |

**Structured Text**

```
dest_1 := value_1;
```

# File Average (AVE)

The AVE instruction calculates the average of a set of values.

## Operands:
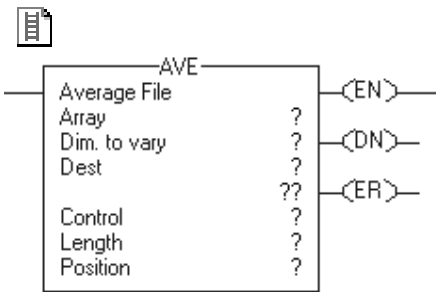


### Relay Ladder

| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| Array | SINT | array tag | find the average of the values in this array |
| | INT | | specify the first element of the group of elements to average |
| | DINT | | **do not** use CONTROL.POS in the subscript |
| | **REAL** | | |
| 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 | | |
| | **REAL** | | |
| 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 <u>Appendix C, Structured Text Programming</u> 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 337. |
| postscan | The rung-condition-out is set to false. |

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

dimension 1

| 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 |

dimension 0

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

*dint_ave* = 12

**Relay Ladder**

```
              ┌─────────AVE─────────┐
              │ Average File        │──(EN)
──────────────┤ Array    array_dint[0,1] │
              │ Dim. to vary      0 │──(DN)──
              │ Dest        dint_ave │
              │                  0 ← │──(ER)──
              │ Control      control_1 │
              │ Length           4 ← │
              │ Position         0 ← │
              └─────────────────────┘
```
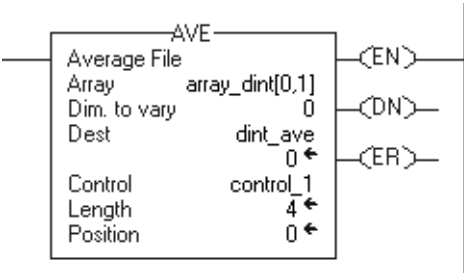
**Structured Text**

```
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

| 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 |

dimension 0

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

*dint_ave* = 3

**Relay Ladder**

```
                    ─AVE─
                  Average File
 ──────────── Array        array_dint[3,0]      ─(EN)─
              Dim. to vary              1
              Dest             dint_ave          ─(DN)─
                                      0 ◄
                                                 ─(ER)─
              Control          control_1
              Length                  5 ◄
              Position                0 ◄
```

**Structured Text**

```
SIZE(array_dint,1,length);
sum := 0;
FOR position = 0 TO (length-1) DO
    sum := sum + array_dint[position];
END_FOR;
dint_ave := sum / length;
```

# File Sort (SRT)

The SRT instruction sorts a set of values in one dimension (Dim to vary) of the Array into ascending order.

## Operands:
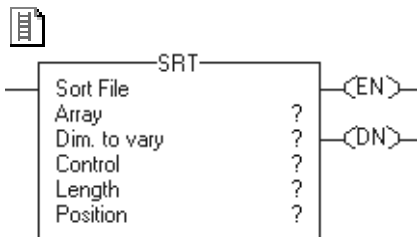
### Relay Ladder

| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| Array | **SINT** | array tag | array to sort |
| | **INT** | | specify the first element of the group of elements to sort |
| | **DINT** | | |
| | **REAL** | | **do not** use CONTROL.POS in the subscript |
| 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] |
| Control | CONTROL | tag | control structure for the operation |
| Length | DINT | immediate | number of elements of the array to sort |
| Position | DINT | immediate | current element in the array |
| | | | initial value is typically 0 |

### Structured Text

```
SRT(Array,Dimtovary,
    Control);
```

The operands are the same as those for the relay ladder SRT instruction. However, you specify the Length and Position values by accessing the .LEN and .POS members of the CONTROL structure, rather than by including values in the operand list.

### CONTROL Structure

| Mnemonic | Data Type | Description |
|----------|-----------|-------------|
| .EN | BOOL | The enable bit indicates that the SRT instruction is enabled. |
| .DN | BOOL | The done bit is set when the specified elements have been sorted. |
| .ER | BOOL | The error bit is set when either .LEN < 0 or .POS < 0. Either of these conditions also generates a major fault. |
| .LEN | DINT | The length specifies the number of elements in the array on which the instruction |
| .POS | DINT | The position contains the position of the current element that the instruction is accessing. |

**Description:**  The SRT instruction sorts a set of values in one dimension (Dim to vary) of the Array into ascending order.

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

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

**IMPORTANT**  Make sure the Length does not cause the instruction to exceed the specified Dimension to vary. If this happens, unexpected results will occur.

This is a transitional instruction:

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

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

**Fault Conditions:**

| A Major Fault Will Occur If | Fault Type | Fault Code |
|---|---|---|
| .POS < 0 or .LEN < 0 | 4 | 21 |
| Dimension to vary does not exist for the specified array | 4 | 20 |
| Instruction tries to access data outside of the array boundaries | 4 | 20 |

### Execution:

| Condition | Relay Ladder Action | Structured Text Action |
|---|---|---|
| prescan | The .EN bit is cleared. | The .EN bit is cleared. |
| | The .DN bit is cleared. | The .DN bit is cleared. |
| | The .ER bit is cleared. | The .ER bit is cleared. |
| | The rung-condition-out is set to false. | |
| rung-condition-in is false | | na |



| Condition | Relay Ladder Action | Structured Text Action |
|---|---|---|
| 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 sorts the specified elements of the array into ascending order. | The instruction sorts the specified elements of the array into ascending order. |
| postscan | The rung-condition-out is set to false. | No action taken. |

**Example 1:**  Sort *int* _array, which is DINT[4,5].

**Before**

dimension 1

| 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 |

dimension 0

**After**

dimension 1

| subscripts | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | 20 | 19 | 3 | 17 | 16 |
| 1 | 15 | 14 | 8 | 12 | 11 |
| 2 | 10 | 9 | 13 | 7 | 6 |
| 3 | 5 | 4 | 18 | 2 | 1 |

dimension 0

**Relay Ladder**

```
            ┌─────SRT─────────┐
            │ Sort File        │──(EN)
            │ Array  int_array[0,2]
            │ Dim. to vary    0 │──(DN)
            │ Control  control_1│
            │ Length         4 ←│
            │ Position       0 ←│
            └──────────────────┘
```
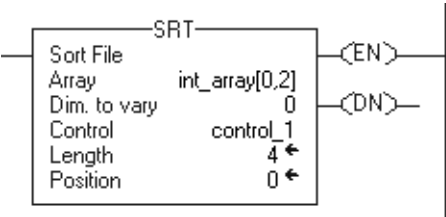
**Structured Text**

```
control_1.LEN := 4;

control_1.POS := 0;

SRT(int_array[0,2],0,control_1);
```

<div align="center">

**Example 2:**    Sort *int* _array, which is DINT[4,5].

</div>

**Before**                                                                 **After**

dimension 1                                                          dimension 1

| 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 |

dimension 0

| subscripts | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | 20 | 19 | 18 | 17 | 16 |
| 1 | 15 | 14 | 13 | 12 | 11 |
| 2 | 6 | 7 | 8 | 9 | 10 |
| 3 | 5 | 4 | 3 | 2 | 1 |

dimension 0

**Relay Ladder**

```
                    ┌─SRT────────────────┐
                    │ Sort File          │──(EN)──
                    │ Array    int_array[2,0]│
                    │ Dim. to vary        1│──(DN)──
                    │ Control     control_1│
                    │ Length            5 ←│
                    │ Position          0 ←│
                    └────────────────────┘
```
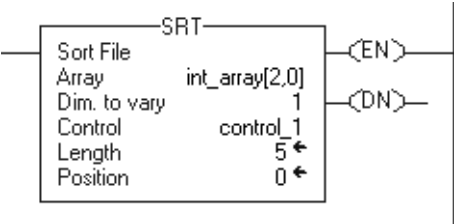
**Structured Text**

```
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.
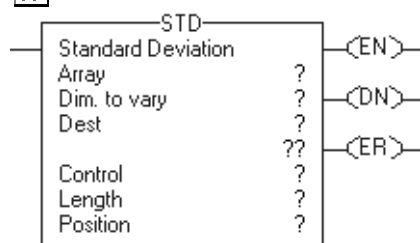
**Operands:**



## Relay Ladder

| Operand | Type | Format | Description |
|---|---|---|---|
| Array | SINT | array tag | find the standard deviation of the values in this array |
| | INT | | |
| | **DINT** | | specify the first element of the group of elements to use in calculating the standard deviation |
| | **REAL** | | |
| | | | **do not** use CONTROL.POS in the subscript |
| | | A SINT or INT tag converts to a DINT value by sign-extension. | |
| Dimension to vary | DINT | immediate | 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 for information on the syntax of constructs within structured text.

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

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

Where:

- start = dimension-to-vary subscript of the array operand
- $x_i$ = variable element in the array
- N = number of specified elements in the array
- AVE = 

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

**IMPORTANT**    Make sure the Length does not cause the instruction to exceed the specified Dimension to vary. If this happens, the Destination will be incorrect.

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

**Fault Conditions:**

| A Major Fault Will Occur If | Fault Type | Fault Code |
|---|---|---|
| .POS < 0 or .LEN < 0 | 4 | 21 |
| Dimension to vary does not exist for the specified array | 4 | 20 |

### Execution:

| Condition | Relay Ladder Action |
|---|---|
| prescan | The .EN bit is cleared. |
| | The .DN bit is cleared. |
| | The .ER bit is cleared. |
| | The rung-condition-out is set to false. |
| rung-condition-in is false | |



| | |
|---|---|
| rung-condition-in is true | The STD instruction calculates the standard deviation of the specified elements. |
| | Internally, the instruction uses a FAL instruction to calculate the average: |
| | Expression = standard deviation calculation |
| | Mode = ALL |
| | For details on how the FAL instruction executes, see page 337. |
| postscan | The rung-condition-out is set to false. |

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

dimension 1

| 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 |

dimension 0

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

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

*real_std* = 6.454972

**Relay Ladder**

```
            ┌─────STD──────────┐
            │ Standard Deviation│──(EN)──
            │ Array    dint_array[0,4]│
            │ Dim. to vary      0│──(DN)──
            │ Dest        real_std│
            │             0.0 ←│──(ER)──
            │ Control    control_1│
            │ Length          4 ←│
            │ Position        0 ←│
            └──────────────────┘
```
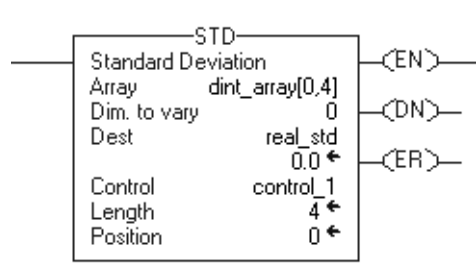
**Structured Text**

```
SIZE(dint_array,0,length);
sum := 0;
FOR position = 0 TO (length-1) DO
    sum := sum + dint_array[position];
END_FOR;
average := sum / length;
sum := 0;
FOR position = 0 TO (length-1) DO
    sum := sum + ((dint_array[position] - average)**2);
END_FOR;
real_std := SQRT(sum / (length-1));
```

**Example 2:**    Calculate the standard deviation of d*int_array*, which is DINT[4,5].

|  | dimension 1 | | | | |
|---|---|---|---|---|---|
| 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 |

dimension 0

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

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

*real_std* = 1.581139

### Relay Ladder

```
                    ┌──────STD──────────┐
                    │ Standard Deviation │──(EN)──
              ──────│ Array    dint_array[0,0] │
                    │ Dim. to vary      1 │──(DN)──
                    │ Dest         real_std │
                    │              0.0 ◄ │──(ER)──
                    │ Control    control_1 │
                    │ Length          5 ◄ │
                    │ Position        0 ◄ │
                    └───────────────────┘
```
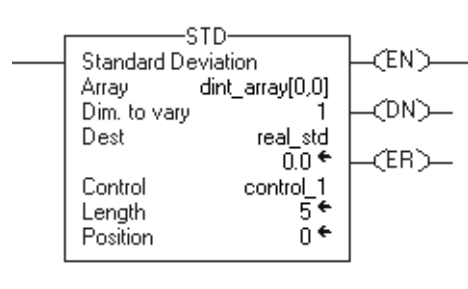
### Structured Text

```
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));
```
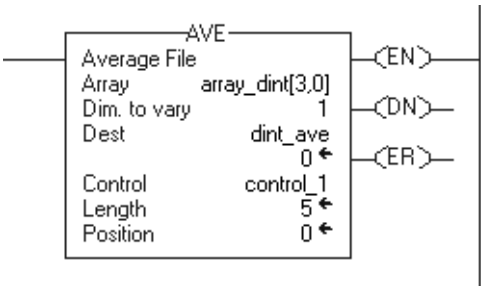
# Size In Elements (SIZE)

The SIZE instruction finds the size of a dimension of an array.

## Operands:

### Relay Ladder



```
        ─SIZE─
 Size in Elements
 Source            ?
                  ??
 Dim. To Vary      ?
 Size              ?
                  ??
```

| 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: |

| For The Size Of | Enter |
|---|---|
| first dimension | 0 |
| second dimension | 1 |
| third dimension | 2 |

| Operand | Type | Format | Description |
|---|---|---|---|
| Size | SINT | tag | tag to store the number of elements in the specified dimension of the array |
| | INT | | |
| | DINT | | |
| | REAL | | |

### Structured Text



`SIZE(Source,Dimtovary,Size);`

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

**Description:** The SIZE instruction finds the number of elements (size) in the designated dimension of the Source array and places the result in the Size operand.

- The instruction finds the size of one dimension of an array.
- The instruction operates on an:
  - array
  - array in a structure
  - array that is part of a larger array

**Arithmetic Status Flags:** not affected

**Fault Conditions:** none.

**Execution:**

| Condition | Relay Ladder Action | Structured Text Action |
|---|---|---|
| prescan | The rung-condition-out is set to false. | No action taken. |
| rung-condition-in is false | The rung-condition-out is set to false. | na |
| rung-condition-in is true | The instruction executes.<br><br>The rung-condition-out is set to true. | na |
| EnableIn is set | na | EnableIn is always set.<br><br>The instruction executes. |
| instruction execution | The instruction finds the size of a dimension. | The instruction finds the size of a dimension. |
| postscan | The rung-condition-out is set to false. | No action taken. |

**Example 1:** Find the number of elements in dimension 0 (first dimension) of *array_a*. Store the size in *array_a_size*. In this example, dimension 0 of *array_a* has 10 elements.

**Relay Ladder**

```
            ─────SIZE─────
           Size in Elements
           Source       array_a[0]
                              255
           Dim. To Vary          0
           Size        array_a_size
                               10
```
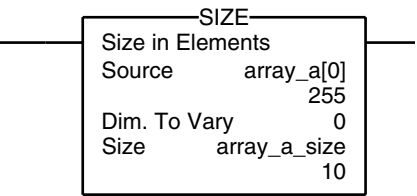
**Structured Text**

```
SIZE(array_a,0,array_a_size);
```

**Example 2:** Find the number of elements in the DATA member of *string_1*, which is a string. Store the size in *string_1_size*. In this example, the DATA member of *string_1* has 82 elements. (The string uses the default STRING data type.) Since each element holds one character, *string_1* can contain up to 82 characters.

**Relay Ladder**

```
            ─────SIZE─────
           Size in Elements
           Source   string_1.DATA[0]
                              '$00'
           Dim. To Vary          0
           Size        string_1_size
                               82
```
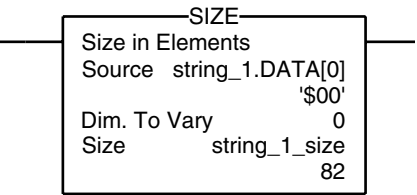
**Structured Text**

```
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

```
                 ─────SIZE──────────
        │       Size in Elements                │
        ┤       Source   strings_a[0].DATA[0]    ├
        │                          '$00'         │
        │       Dim. To Vary              0      │
        │       Size            data_size_a      │
        │                               24       │
```
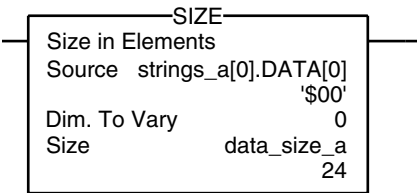
### Structured Text

```
SIZE(strings_a[0].DATA[0],0,data_size_a);
```

**Notes:**

# Array (File)/Shift Instructions

## (BSL, BSR, FFL, FFU, LFL, LFU)

## Introduction

Use the array (file)/shift instructions to modify the location of data within arrays.

| If You Want To | Use This Instruction | Available In These Languages | See Page |
|---|---|---|---|
| Load bits into, shift bits through, and unload bits from a bit array one bit at a time. | BSL | relay ladder | 386 |
| | BSR | relay ladder | 390 |
| Load and unload values in the same order. | FFL | relay ladder | 394 |
| | FFU | relay ladder | 400 |
| Load and unload values in reverse order. | LFL | relay ladder | 406 |
| | LFU | relay ladder | 412 |

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:
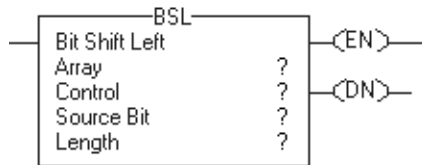
### Relay Ladder

```
          ┌─BSL──────────────┐
──────────┤ Bit Shift Left   ├──(EN)──
          │ Array        ?   │
          │ Control      ?   ├──(DN)──
          │ Source Bit   ?   │
          │ Length       ?   │
          └──────────────────┘
```

| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| Array | DINT | array tag | array to modify |
| | | | specify the first element of the group of elements |
| | | | **do not** use CONTROL.POS in the subscript |
| Control | CONTROL | tag | control structure for the operation |
| Source bit | BOOL | tag | bit to shift |
| Length | DINT | immediate | number of bits in the array to shift |

### CONTROL Structure

| Mnemonic | Data Type | Description |
|----------|-----------|-------------|
| .EN | BOOL | The enable bit indicates that the BSL instruction is enabled. |
| .DN | BOOL | The done bit is set to indicate that bits shifted one position to the left. |
| .UL | BOOL | The unload bit is the instruction's output. The .UL bit stores the status of the bit that was shifted out of the range of bits. |
| .ER | BOOL | The error bit is set when .LEN < 0. |
| .LEN | DINT | The length specifies the number of array bits to shift. |

**Description:** When enabled, the instruction unloads the uppermost bit of the specified bits to the .UL bit, shifts the remaining bits one position left, and loads Source bit into bit 0 of Array.

| 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:**

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

| Condition: | Relay Ladder Action |
|---|---|

rung-condition-in is true



| postscan | The rung-condition-out is set to false. |
|---|---|

**Example 1:**   When enabled, the BSL instruction starts at bit 0 in *array_dint[0]*. The instruction unloads *array_dint[0].9* into the .UL bit, shifts the remaining bits, and loads *input_1* into *array_dint[0].0*.  The values in the remaining bits (10-31) are invalid.

```
            ┌──BSL────────────────────┐
         ───┤ Bit Shift Left          ├──⟨EN⟩──
            │ Array         array_dint[0]⟩  
            │ Control         control_1 ├──⟨DN⟩──
            │ Source Bit        input_1 │
            │ Length                 10 │
            └─────────────────────────┘
```

                                                    9 8 7 6 5 4 3 2 1 0

*array_dint[0]*  1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0
before shift

                       0                     these bits shift left                input_1  1
                     .UL bit

                                                    9 8 7 6 5 4 3 2 1 0

*array_dint[0]*                                      0 1 1 1 1 0 0 0 0 1
after shift
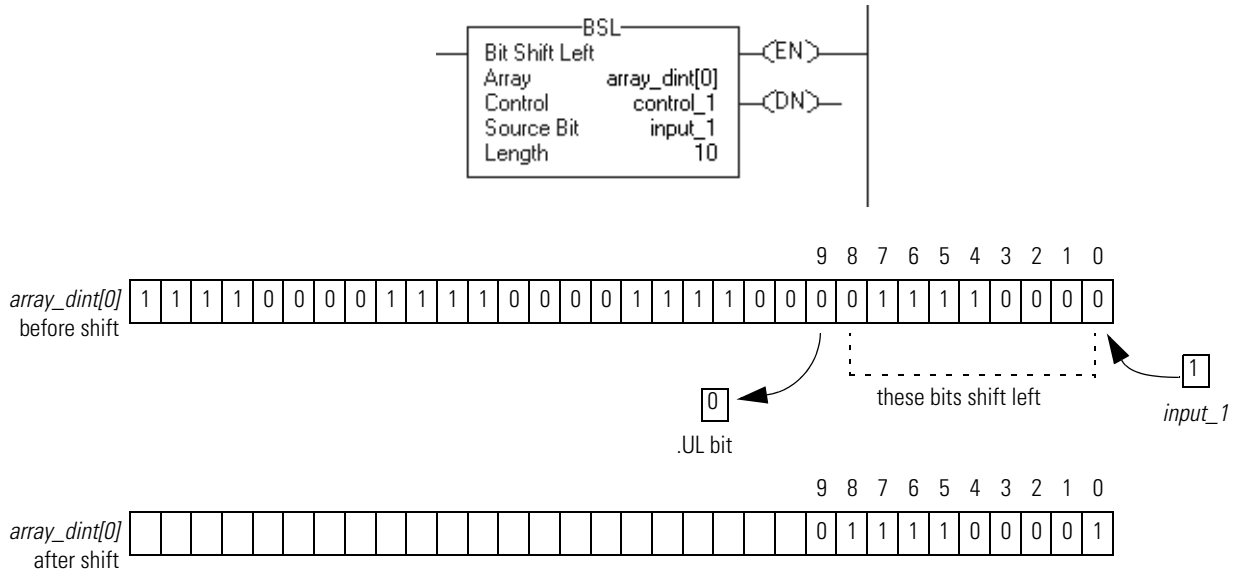
**Example 2:**   When enabled, the BSL instruction starts at bit 0 in *array_dint[0]*. The instruction unloads *array_dint[1].25* into the .UL bit, shifts the remaining bits, and loads *input_1* into *array_dint[0].0.* The values in the remaining bits (31-26 in *array_dint[1]*) are invalid. Note how *array_dint[0].31* shifts across words to *array_dint[1].0.*

```
            ┌──BSL────────────────────┐
         ───┤ Bit Shift Left          ├──⟨EN⟩──
            │ Array         array_dint[0]│
            │ Control         control_1 ├──⟨DN⟩──
            │ Source Bit        input_1 │
            │ Length               58 ← │
            └─────────────────────────┘
```

31                                                                              0

*array_dint[0]*  1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0

                          these bits shift left                           input_1  1

31                                                                              0

*array_dint[1]*          0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 1 1 1 1 0 0 0 0

       0                    these bits shift left
     .UL bit
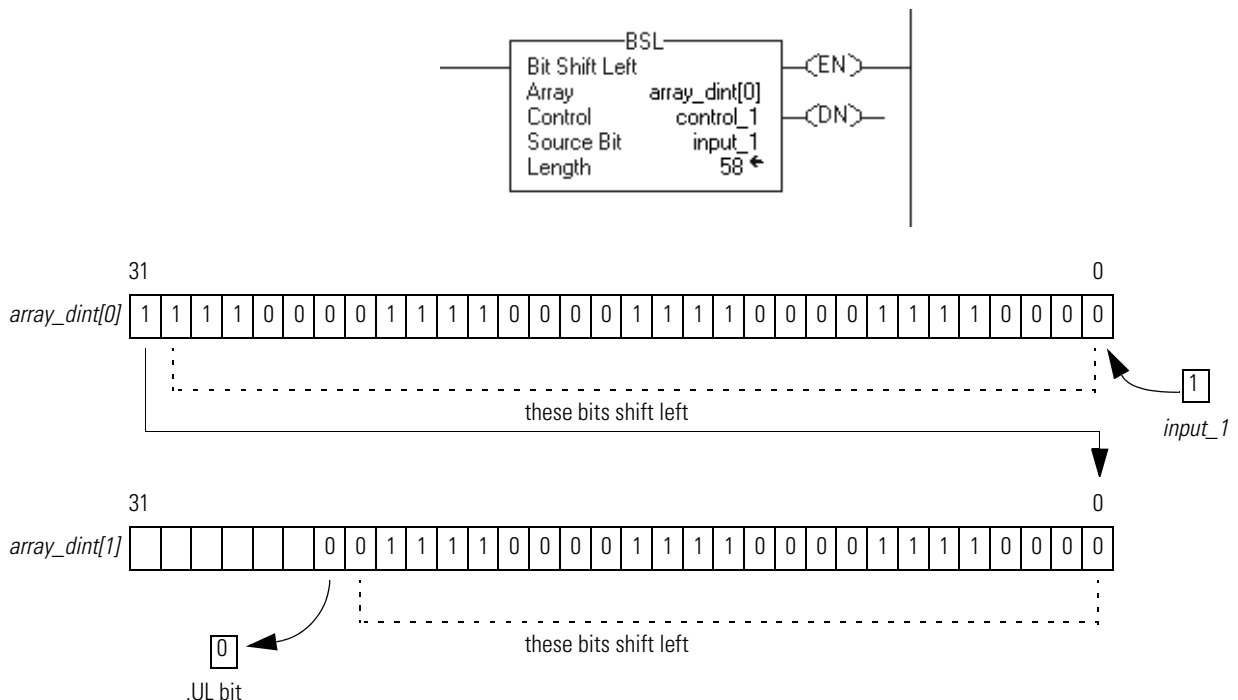
# Bit Shift Right (BSR)

The BSR instruction shifts the specified bits within the Array one position right.

## Operands:
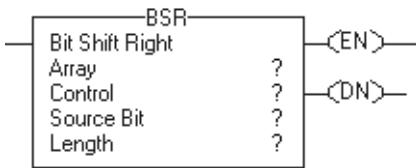
### Relay Ladder

```
----------BSR----------
   Bit Shift Right           (EN)
   Array              ?
   Control            ?       (DN)
   Source Bit         ?
   Length             ?
```

| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| Array | DINT | array tag | array to modify |
| | | | specify the element where to begin the shift |
| | | | **do not** use CONTROL.POS in the subscript |
| Control | CONTROL | tag | control structure for the operation |
| Source bit | BOOL | tag | bit to shift |
| Length | DINT | immediate | number of bits in the array to shift |

### CONTROL Structure

| Mnemonic | Data Type | Description |
|----------|-----------|-------------|
| .EN | BOOL | The enable bit indicates that the BSR instruction is enabled. |
| .DN | BOOL | The done bit is set to indicate that bits shifted one position to the right. |
| .UL | BOOL | The unload bit is the instruction's output. The .UL bit stores the status of the bit that was shifted out of the range of bits. |
| .ER | BOOL | The error bit is set when .LEN < 0. |
| .LEN | DINT | The length specifies the number of array bits to shift. |

**Description:**     When enabled, the instruction unloads the value at bit 0 of Array to the .UL bit, shifts the remaining bits one position right, and loads Source bit into the uppermost bit of the specified bits.

| 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. |

| Condition | Relay Ladder Action |
|---|---|

rung-condition-in is true

examine .EN bit

.EN bit = 1

.EN bit = 0

.EN bit is set

.LEN = 0

yes

.DN bit is set

no

examine source bit

.source bit = 1

.UL bit remains set

.LEN < 0

yes

.ER bit is set

.source bit = 0

.UL bit is set

no

shift array left one position left

source bit → array → .UL bit

.DN bit is set

rung-condition-out is set to **true**

end

| postscan | The rung-condition-out is set to false. |
|---|---|

**Example 1:**    When enabled, the BSR instruction starts at bit 9 in *array_dint[0]*. The instruction unloads *array_dint[0].0* into the .UL bit, shifts the remaining bits right, and loads *input_1* into *array_dint[0].9*. The values in the remaining bits (10-31) are invalid.

```
        ┌──────BSR──────────┐
        │ Bit Shift Right   │──(EN)
        │ Array  array_dint[0]│
        │ Control    control_1│──(DN)──
        │ Source Bit   input_1│
        │ Length          10 ←│
        └────────────────────┘
```

```
                                                          9 8 7 6 5 4 3 2 1 0
array_dint[0]  |1|1|1|1|0|0|0|0|1|1|1|1|0|0|0|0|1|1|1|1|0|0|0|0|1|1|1|1|0|0|0|0|
before shift
```

these bits shift right

[1]

*input_1*

```
                                                          9 8 7 6 5 4 3 2 1 0
array_dint[0]  | | | | | | | | | | | | | | | | | | | | | |1|0|0|1|1|1|1|0|0|0|
after shift
```

[0]
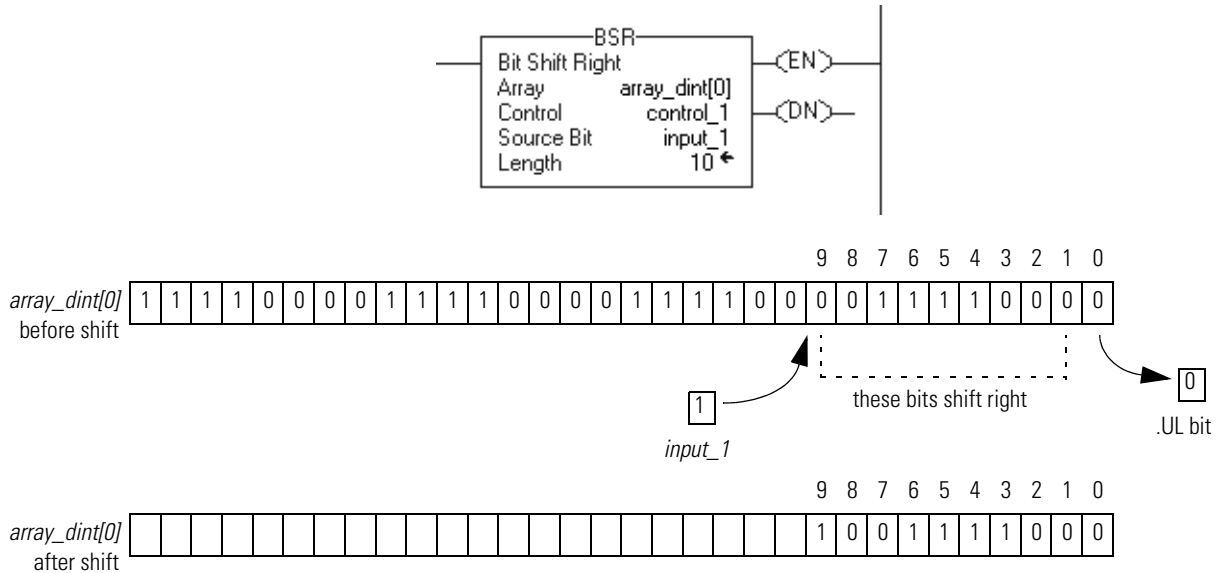
.UL bit

**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*.

```
        ┌──────BSR──────────┐
        │ Bit Shift Right   │──(EN)
        │ Array  array_dint[0]│
        │ Control    control_1│──(DN)──
        │ Source Bit   input_1│
        │ Length          58 │
        └────────────────────┘
```

```
31                                                                          0
array_dint[0]  |1|1|1|1|0|0|0|0|1|1|1|1|0|0|0|0|1|1|1|1|0|0|0|0|1|1|1|1|0|0|0|0|
```

these bits shift right

[0]

.UL bit

```
31                                                                          0
array_dint[1]  | | | | | | |0|0|1|1|1|1|0|0|0|0|1|1|1|1|0|0|0|0|1|1|1|1|0|0|0|0|
```

[1]

*input_1*

these bits shift right
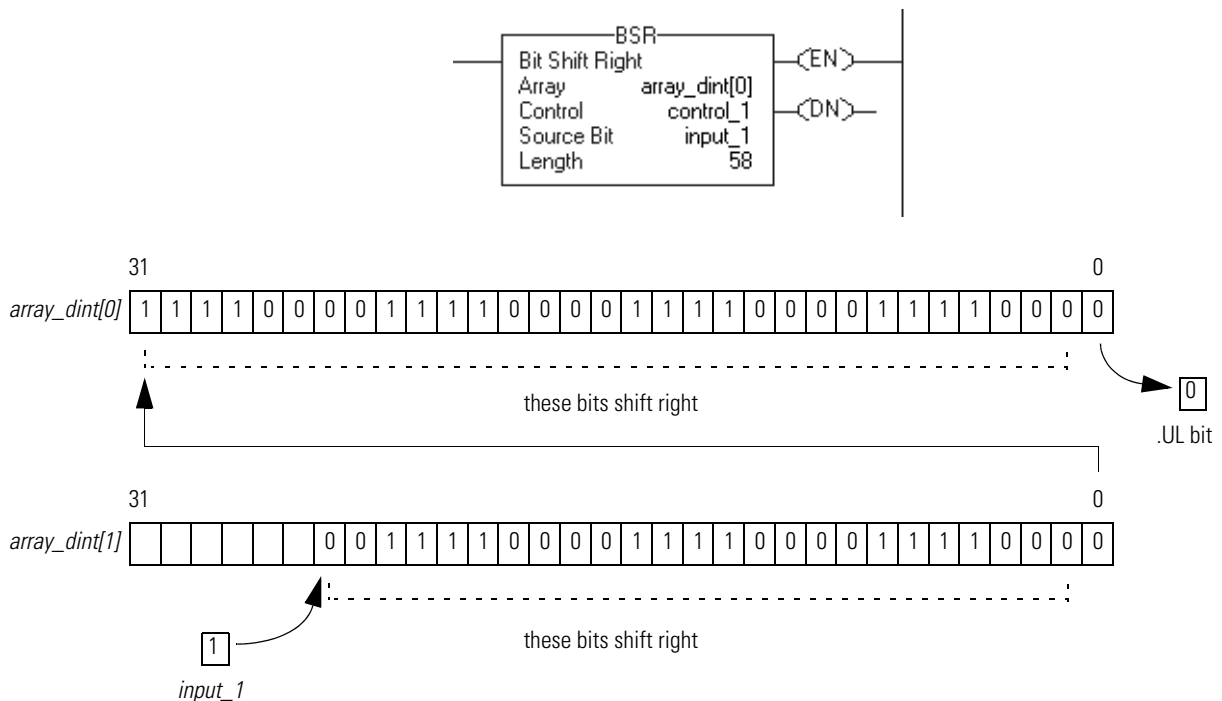
# FIFO Load (FFL)

The FFL instruction copies the Source value to the FIFO.

## Operands:
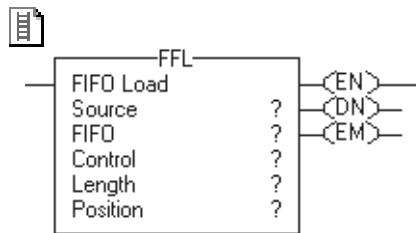


### Relay Ladder

| 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 | | **do not** 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 ≤0 or .POS < 0, both the .EM bit and .DN bit are set. |
| .LEN | DINT | The length specifies the maximum number of elements the FIFO can hold at one time. |
| .POS | DINT | The position identifies the location in the FIFO where the instruction will load the next value. |

**Description:** Use the FFL instruction with the FFU instruction to store and retrieve data in a first-in/first-out order. When used in pairs, the FFL and FFU instructions establish an asynchronous shift register.

Typically, the Source and the FIFO are the same data type.

When enabled, the FFL instruction loads the Source value into the position in the FIFO identified by the .POS value. The instruction loads one value each time the instruction is enabled, until the FIFO is full.

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

**Arithmetic Status Flags:**  not affected

**Fault Conditions:**

| A Major Fault Will Occur If | Fault Type | Fault Code |
|---|---|---|
| (starting element + .POS) > FIFO array size | 4 | 20 |

### Execution:

| Condition | Relay Ladder Action |
|---|---|

prescan

```
                  ↓
      ┌─────────────────────────┐
      │ .EN bit is set to prevent a
      │ false load when scan begins
      └─────────────────────────┘
                  ↓
              ◇ .LEN < 0 ◇ ──yes──────────────────┐
                  │ no                             │
              ◇ .POS < 0 ◇ ──yes──────────────────┤
                  │ no                             │
      ┌──────────────────┐          ┌──────────────────┐
      │ .EM is cleared   │          │ .EM is set       │
      └──────────────────┘          └──────────────────┘
                  ↓                           │
              ◇ .POS = 0 ◇ ──yes──→ ┌──────────────┐
                  │ no              │ .EM is set   │
                  ↓←───────────────└──────────────┘
              ◇ .POS ≥ .LEN ◇ ──yes──→ ┌──────────────┐
                  │ no                 │ .DN is set   │
                  ↓←──────────────────└──────────────┘
      ┌──────────────────────────┐
      │ rung-condition-out is set to
      │ **false**
      └──────────────────────────┘
                  ↓
               ( end )
```

| Condition | Relay Ladder Action |
|---|---|

rung-condition-in is false

```
                        │
                        ▼
              ┌───────────────────┐
              │ .EN bit is cleared │
              └───────────────────┘
                        │
                        ▼
                    ╱────────╲          yes
                   ╱ .LEN < 0 ╲──────────────────────┐
                   ╲          ╱                       │
                    ╲────────╱                        │
                        │ no                          │
                        ▼                             │
                    ╱────────╲          yes           │
                   ╱ .POS < 0 ╲─────────────────────┐ │
                   ╲          ╱                      │ │
                    ╲────────╱                       │ │
                        │ no                         ▼ ▼
              ┌───────────────────┐          ┌───────────────┐
              │  .EM is cleared   │          │  .EM is set   │
              └───────────────────┘          └───────────────┘
                        │                           │
                        ▼                           │
                    ╱────────╲    yes   ┌─────────────┐
                   ╱ .POS = 0 ╲────────▶│ .EM is set  │
                   ╲          ╱         └─────────────┘
                    ╲────────╱                │
                        │ no ◀───────────────┘
                        ▼
                    ╱──────────╲    yes   ┌─────────────┐
                   ╱ .POS ≥ .LEN ╲───────▶│  .DN is set  │
                   ╲            ╱         └─────────────┘
                    ╲──────────╱               │
                        │ no ◀─────────────────┴──────────┘
                        ▼
              ┌──────────────────────────┐
              │ rung-condition-out is set to │
              │ **false**                │
              └──────────────────────────┘
                        │
                        ▼
                     ( end )
```

| Condition | Relay Ladder Action |
|---|---|

rung-condition-in is true

examine .EN bit

.EN = 0 → .EN bit is set →

.LEN < 0

yes

no

.EN = 1

.LEN < 0

yes

no

.POS < 0

yes

no

.EM bit is cleared
.DN is cleared

.EM bit is set

.POS < 0

yes

no

.EM bit is cleared

.EM bit is set

.POS ≥ .LEN

yes → .DN bit is set

no

.POS = 0

yes → .EM bit is set

no

.POS or .LEN > size of array

yes → major fault

no

.POS ≥ .LEN

yes → .DN bit is set

no

.POS > .LEN

yes → .POS = .POS - 1

no

FIFO[.POS - 1] = source

rung-condition-out is set to **true**

end

| 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.

```
        ┌──FFL──────────────────┐
────────┤ FIFO Load              ├──(EN)──
        │ Source        value_1  │──(DN)──
        │ FIFO      array_dint[0] │──(EM)──
        │ Control       control_1 │
        │ Length            10   │
        │ Position           5   │
        └────────────────────────┘
```

before FIFO load

| | |
|---|---|
| *array_dint[0]* | 00000 |
| | 11111 |
| | 22222 |
| | 33333 |
| | 44444 |
| *array_dint[5]* | 00000 |
| | 00000 |
| | 00000 |
| | 00000 |
| | 00000 |

*control_1.pos* = 5

*value_1* = 55555

after FIFO load

| |
|---|
| 00000 |
| 11111 |
| 22222 |
| 33333 |
| 44444 |
| 55555 |
| 00000 |
| 00000 |
| 00000 |
| 00000 |

*control_1.pos* = 6
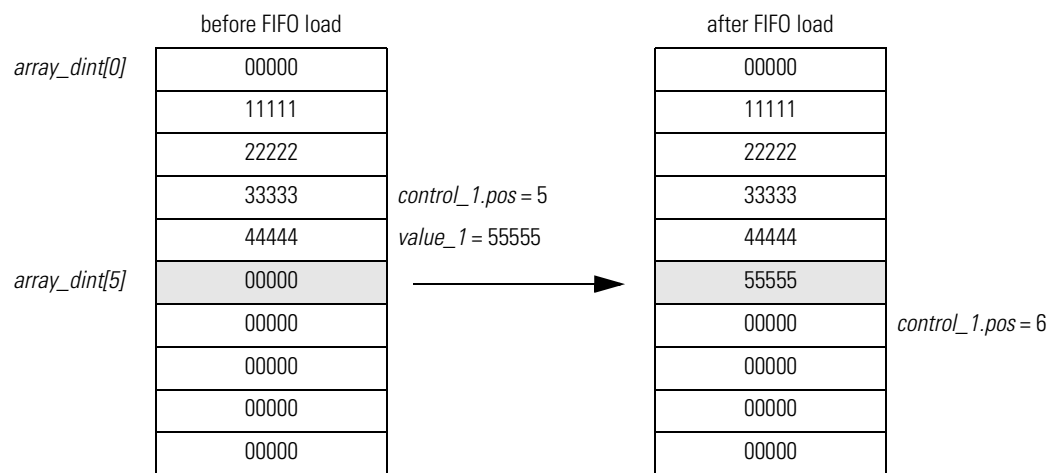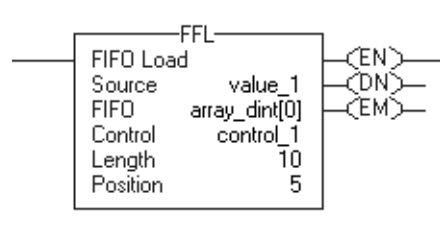
# FIFO Unload (FFU)

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:**
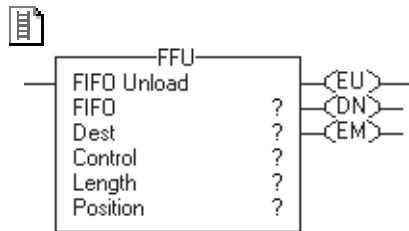
### Relay Ladder

| Operand | Type | Format | Description |
|---|---|---|---|
| FIFO | SINT | array tag | FIFO to modify |
|  | INT |  | specify the first element of the FIFO |
|  | DINT |  | **do not** 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:

| Condition | Relay Ladder Action |
| --- | --- |

prescan

| Condition | Relay Ladder Action |
|---|---|

rung-condition-in is false

```
          │
          ▼
┌──────────────────┐
│ .EU bit is cleared │
└──────────────────┘
          │
          ▼
       ╱╲
      ╱   ╲         yes
     ╱ .LEN ╲ ──────────────────────────────┐
     ╲  < 0 ╱                                │
      ╲   ╱                                  │
       ╲╱                                    │
        │ no                                 │
        ▼                                    │
       ╱╲                                    │
      ╱   ╲         yes                      │
     ╱ .POS ╲ ──────────────────────────────┤
     ╲  < 0 ╱                                │
      ╲   ╱                                  │
       ╲╱                                    │
        │ no                                 ▼
        ▼                          ┌──────────────────┐
┌──────────────────┐               │   .EM is set      │
│  .EM is cleared   │               └──────────────────┘
└──────────────────┘                         │
        │                                     │
        ▼                                     │
       ╱╲                                     │
      ╱   ╲    yes   ┌──────────────┐         │
     ╱ .POS ╲ ─────▶ │  .EM is set   │        │
     ╲  = 0 ╱        └──────────────┘         │
      ╲   ╱               │                   │
       ╲╱                 │                   │
        │ no ◀────────────┘                   │
        ▼                                     │
       ╱╲                                     │
      ╱   ╲    yes   ┌──────────────┐         │
     ╱ .POS ╲ ─────▶ │  .DN is set   │        │
     ╲ ≥.LEN╱        └──────────────┘         │
      ╲   ╱               │                   │
       ╲╱                 │                   │
        │ no ◀────────────┴───────────────────┘
        ▼
┌──────────────────────────┐
│ rung-condition-out is set to │
│ **false**                 │
└──────────────────────────┘
        │
        ▼
      (  end  )
```

| Condition | Relay Ladder Action |
|---|---|

rung-condition-in is true

**Example:**  When enabled, the FFU instruction unloads *array_dint[0]* into *value_2* and shifts the remaining elements in *array_dint*.

```
                ┌─FFU*──────────┐
       ──────── │ FIFO Unload    │──(EU)
                │ FIFO  array_dint[0] │──(DN)──
                │ Dest       value_2  │──(EM)──
                │ Control    cotrol_1 │
                │ Length          10  │
                │ Position         6  │
                └────────────────────┘
```

|  | before FIFO unload |  | after FIFO unload |  |
|---|---|---|---|---|
| *array_dint[0]* | 00000 | → | 11111 | |
| | 11111 | | 22222 | |
| | 22222 | | 33333 | |
| | 33333 | | 44444 | |
| | 44444 | | 55555 | |
| *array_dint[5]* | 55555 | | 00000 | *control_1.pos* = 5 |
| | 00000 | *control_1.pos* = 6 | 00000 | *value_2* = 00000 |
| | 00000 | | 00000 | |
| | 00000 | | 00000 | |
| | 00000 | | 00000 | |
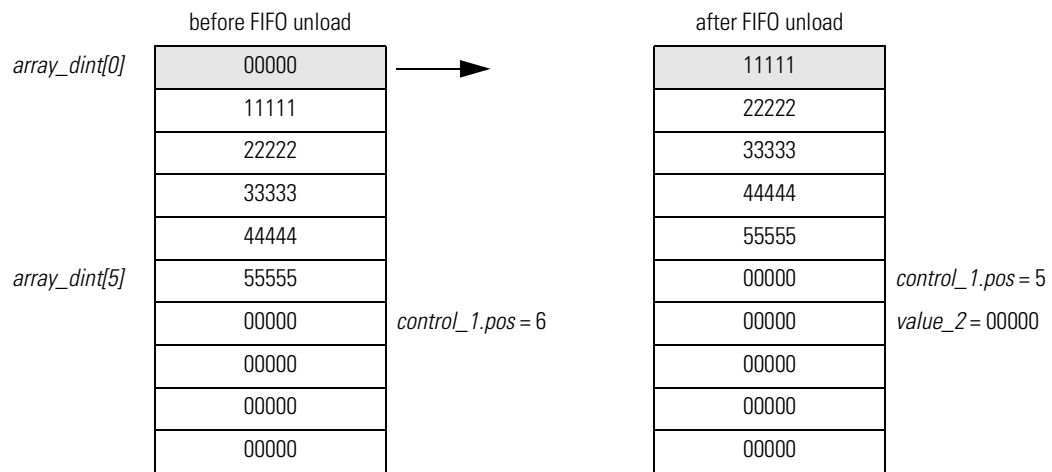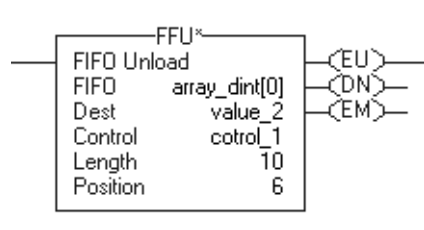
# LIFO Load (LFL)

The LFL instruction copies the Source value to the LIFO.

## Operands:
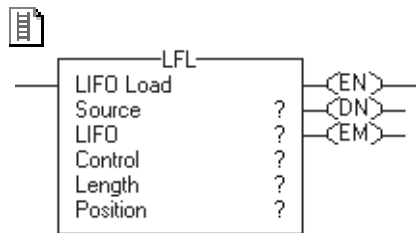
### Relay Ladder

| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| Source | **SINT** | immediate | data to be stored in the LIFO |
|  | **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. |  |  |
| LIFO | **SINT** | array tag | LIFO to modify |
|  | **INT** |  | specify the first element of the LIFO |
|  | **DINT** |  | **do not** 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 LFU |
| Length | DINT | immediate | maximum number of elements the LIFO can hold at one time |
| Position | DINT | immediate | next location in the LIFO where the instruction loads data |
|  |  |  | initial value is typically 0 |

If you use a user-defined structure as the data type for the Source or LIFO operand, use the same structure for both operands.

## CONTROL Structure

| Mnemonic | Data Type | Description: |
|---|---|---|
| .EN | BOOL | The enable bit indicates that the LFL instruction is enabled. |
| .DN | BOOL | The done bit is set to indicate that the LIFO is full (.POS = .LEN). The .DN bit inhibits loading the LIFO until .POS < .LEN. |
| .EM | BOOL | The empty bit indicates that the LIFO is empty. If .LEN ≤0 or .POS < 0, both the .EM bit and .DN bit are set. |
| .LEN | DINT | The length specifies the maximum number of elements the LIFO can hold at one time. |
| .POS | DINT | The position identifies the location in the LIFO where the instruction will load the next value. |

**Description:** Use the LFL instruction with the LFU instruction to store and retrieve data in a last-in/first-out order. When used in pairs, the LFL and LFU instructions establish an asynchronous shift register.

Typically, the Source and the LIFO are the same data type.

When enabled, the LFL instruction loads the Source value into the position in the LIFO identified by the .POS value. The instruction loads one value each time the instruction is enabled, until the LIFO is full.

---

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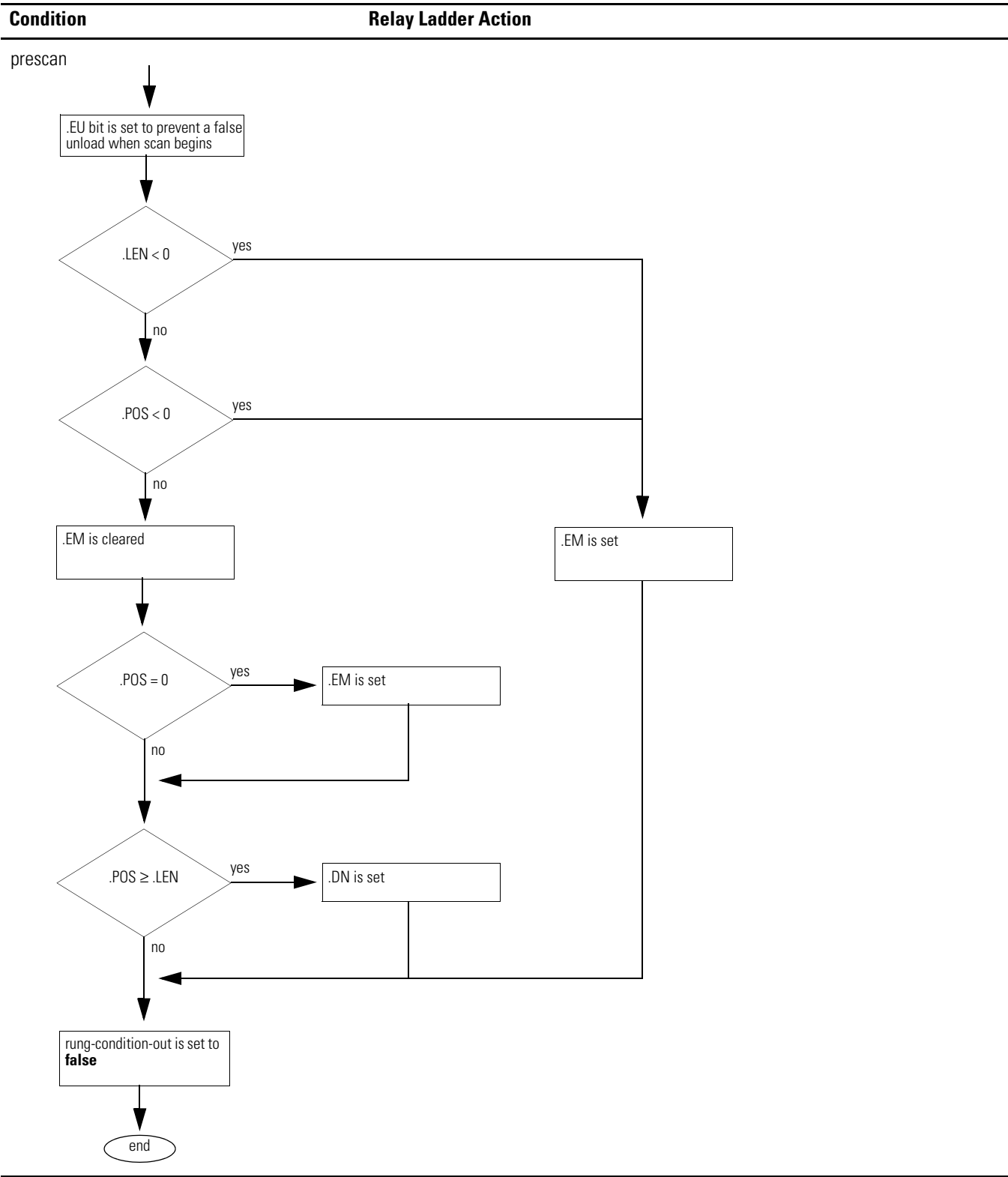
The LFL instruction operates on contiguous memory. In some cases, the instruction loads data past the array into other members of the tag. This happens if the length is too big and the tag is a user-defined data type.
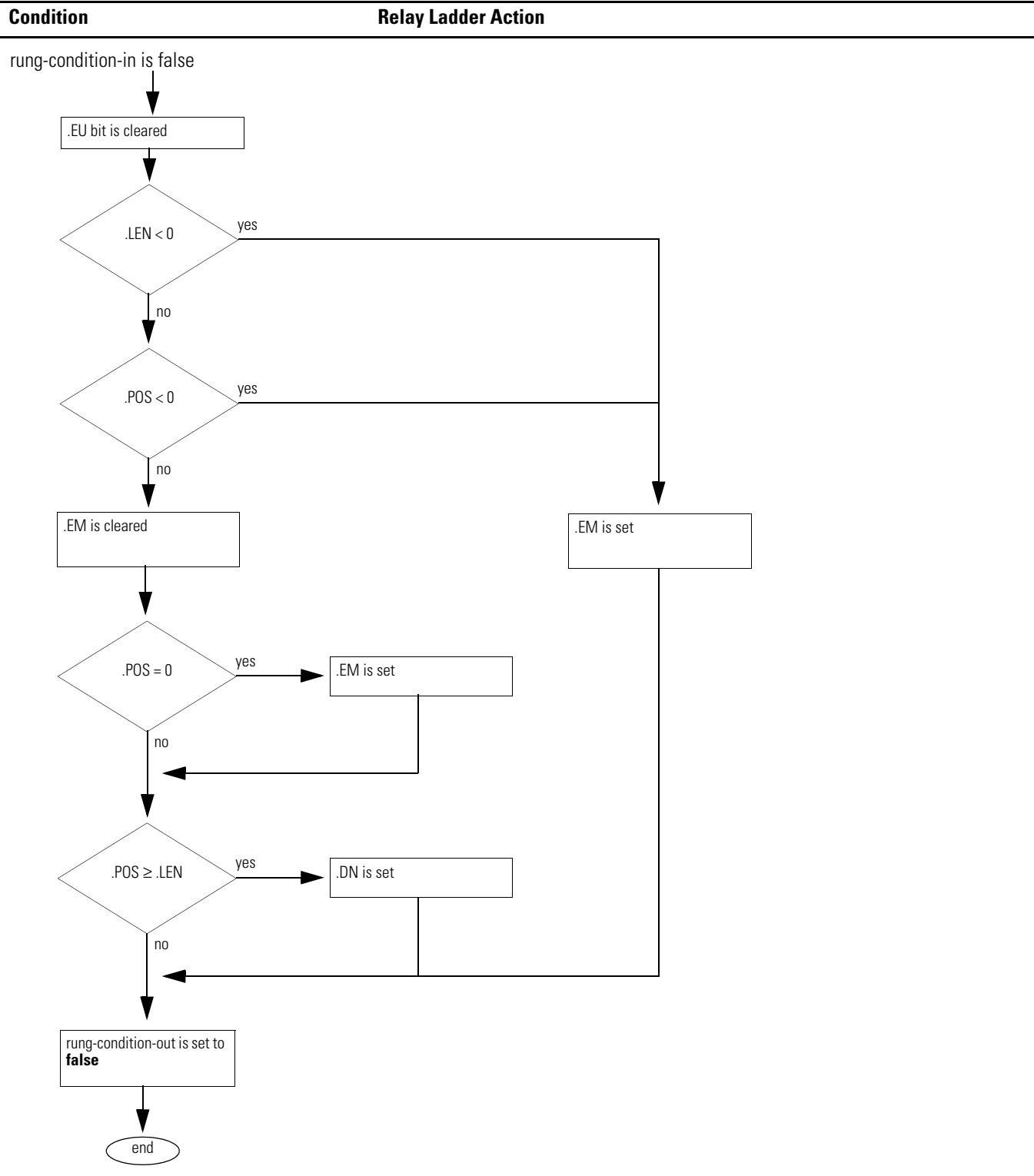
---

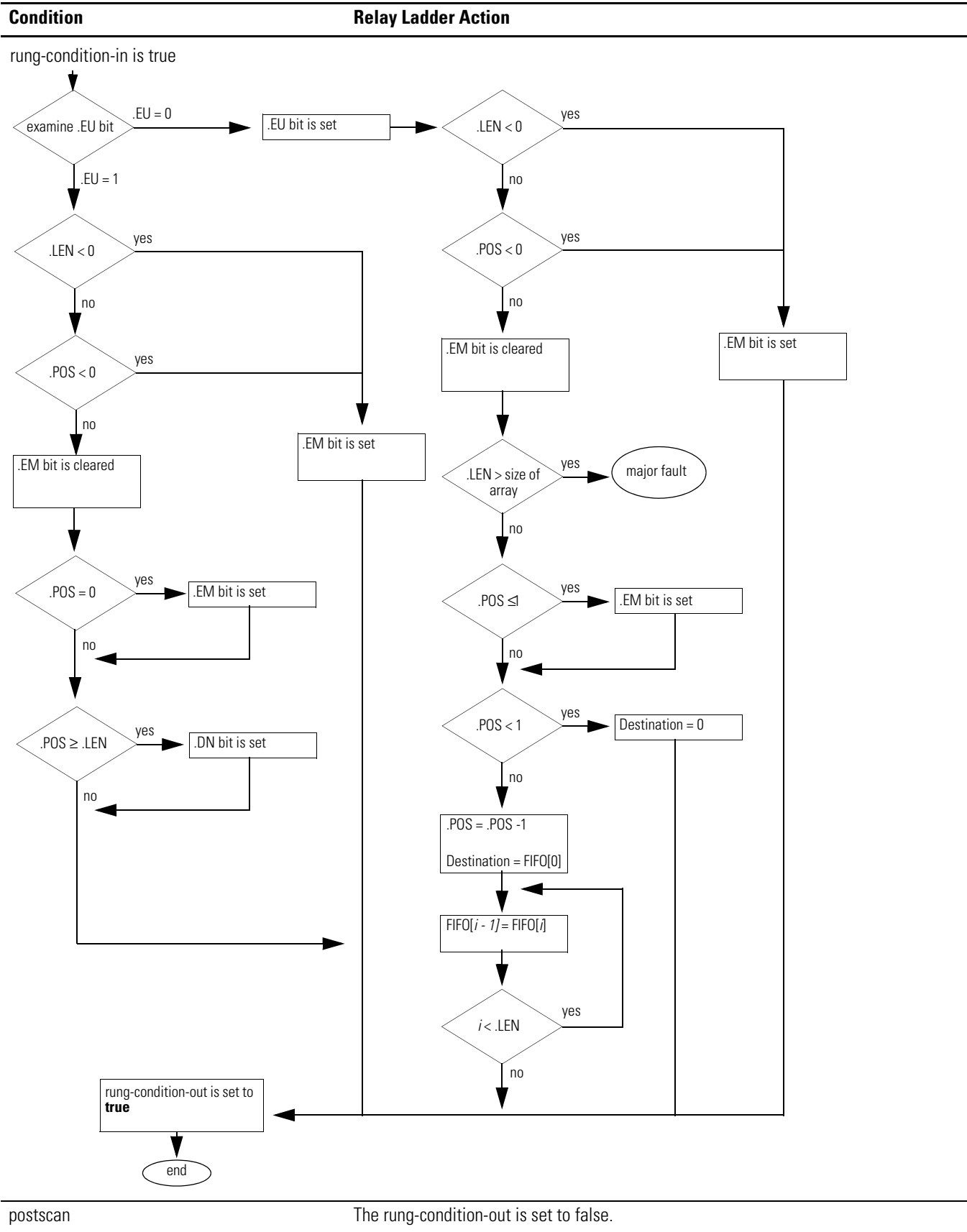**Arithmetic Status Flags:**   not affected

**Fault Conditions:**

| A Major Fault Will Occur If | Fault Type | Fault Code |
|---|---|---|
| (starting element + .POS) > LIFO array size | 4 | 20 |
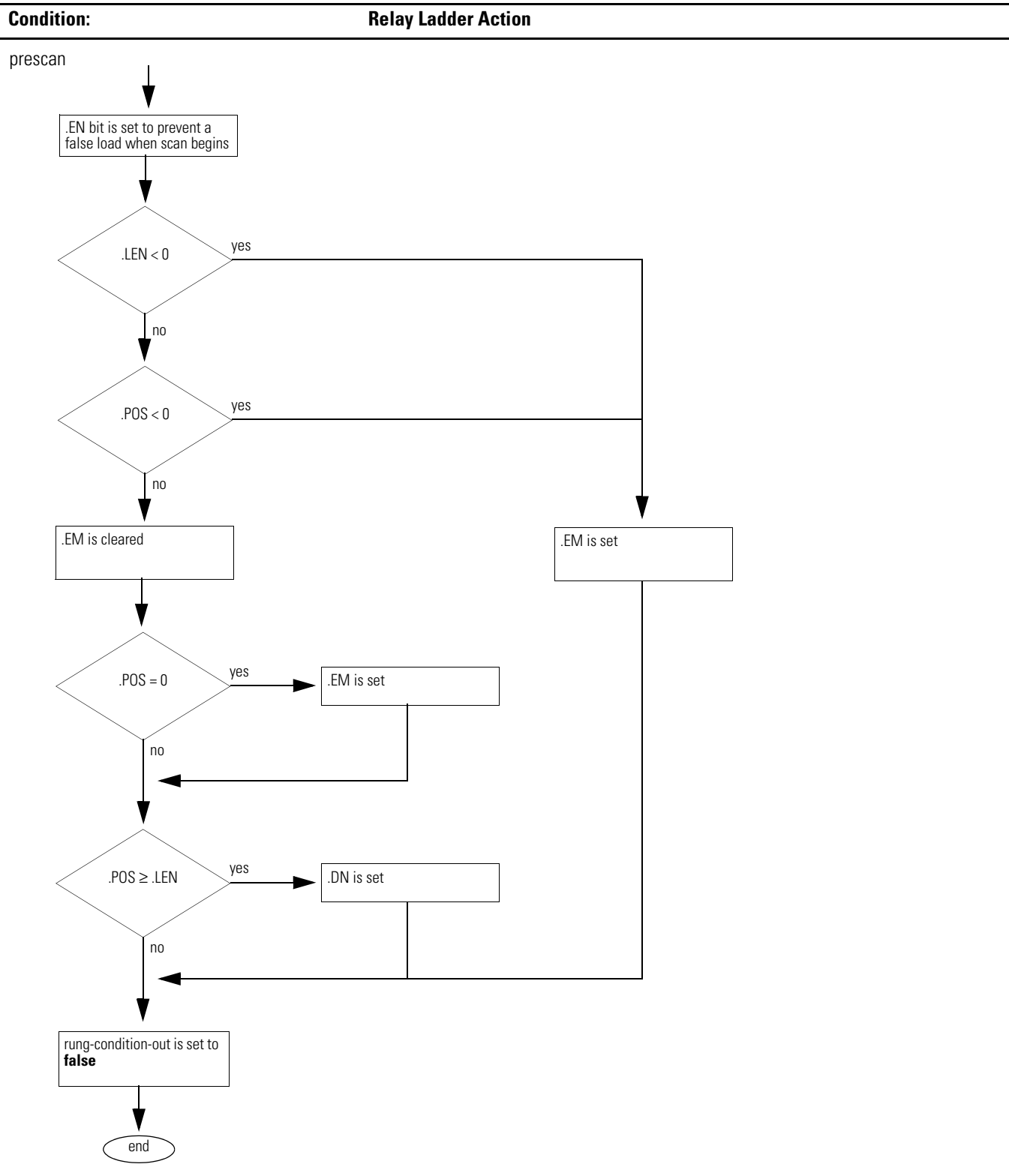
### Execution:

| Condition: | Relay Ladder Action |
|---|---|

prescan

```
                    ┌──────────────────────────┐
                    │ .EN bit is set to prevent a │
                    │ false load when scan begins │
                    └──────────────────────────┘
                                │
                            ╱ .LEN < 0 ╲ ──── yes ──────────────┐
                            ╲          ╱                         │
                                │ no                             │
                            ╱ .POS < 0 ╲ ──── yes ──────┐        │
                            ╲          ╱                 │       │
                                │ no                     │       │
                    ┌──────────────────┐      ┌──────────────────┐
                    │ .EM is cleared   │      │ .EM is set       │
                    └──────────────────┘      └──────────────────┘
                                │
                            ╱ .POS = 0 ╲ ── yes ──→ ┌────────────┐
                            ╲          ╱            │ .EM is set │
                                │ no               └────────────┘
                                │←─────────────────────┘
                            ╱ .POS ≥ .LEN ╲ ── yes ──→ ┌────────────┐
                            ╲             ╱            │ .DN is set │
                                │ no                   └────────────┘
                                │←───────────────────────┘
                    ┌──────────────────────────┐
                    │ rung-condition-out is set to │
                    │ **false**                │
                    └──────────────────────────┘
                                │
                             ( end )
```

| Condition: | Relay Ladder Action |
|---|---|

rung-condition-in is false

.EN bit is cleared

.LEN < 0 — yes

no

.POS < 0 — yes

no

.EM is cleared

.EM is set

.POS = 0 — yes → .EM is set

no

.POS ≥ .LEN — yes → .DN is set

no

rung-condition-out is set to **false**

end

| Condition: | Relay Ladder Action |
|---|---|

rung-condition-in is true



| postscan | The rung-condition-out is set to false. |
|---|---|
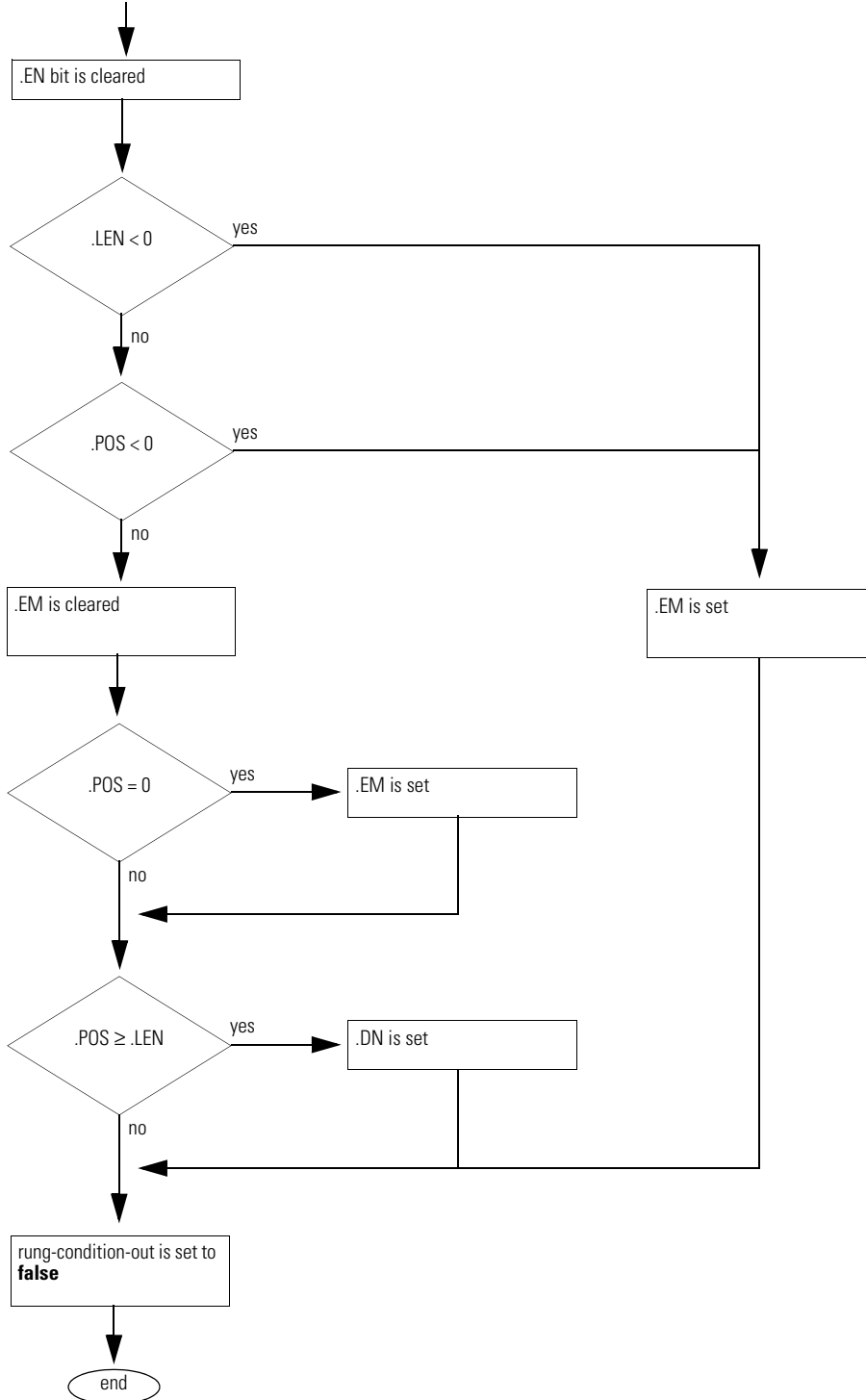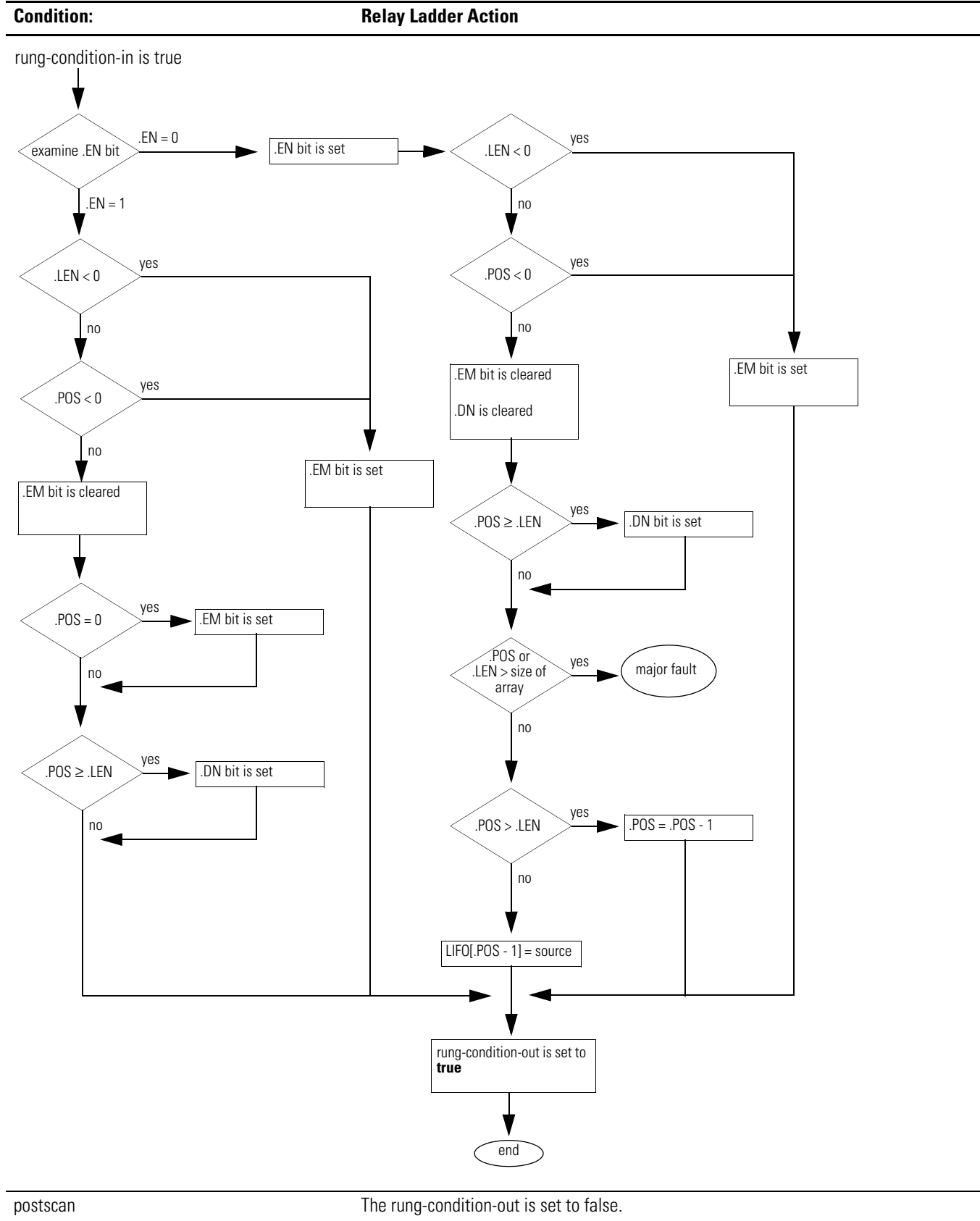
**Example:**    When enabled, the LFL instruction loads *value_1* into the next position in the LIFO, which is *array_dint[5]* in this example.

```
        ┌─────LFL──────────┐
        │ LIFO Load        ├──(EN)──
        │ Source    value_1│──(DN)──
        │ LIFO  array_dint[0]│──(EM)──
        │ Control   control_1│
        │ Length         10│
        │ Position        5│
        └──────────────────┘
```

before LIFO load

| | |
|---|---|
| *array_dint[0]* | 00000 |
| | 11111 |
| | 22222 |
| | 33333 |
| | 44444 |
| *array_dint[5]* | 00000 |
| | 00000 |
| | 00000 |
| | 00000 |
| | 00000 |

*control_1.pos* = 5

*value_1* = 55555

after LIFO load

| |
|---|
| 00000 |
| 11111 |
| 22222 |
| 33333 |
| 44444 |
| 55555 |
| 00000 |
| 00000 |
| 00000 |
| 00000 |

*control_1.pos* = 6
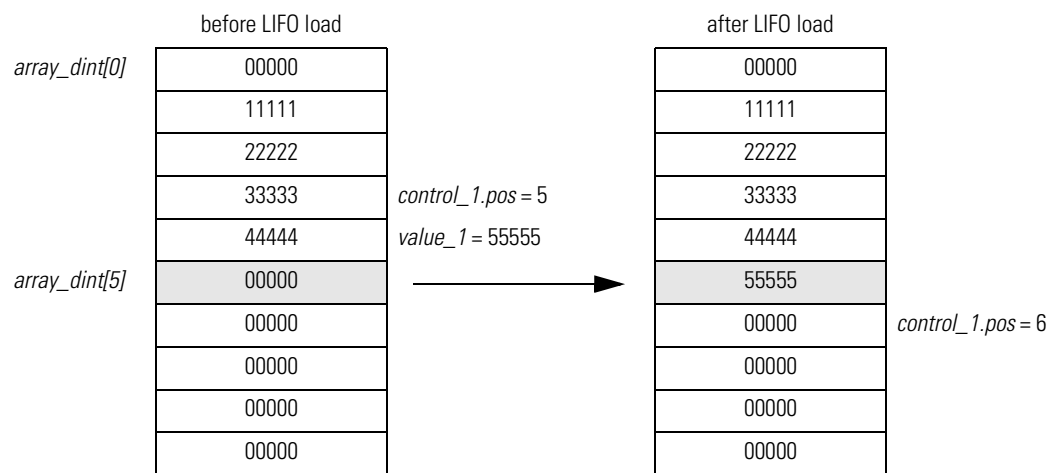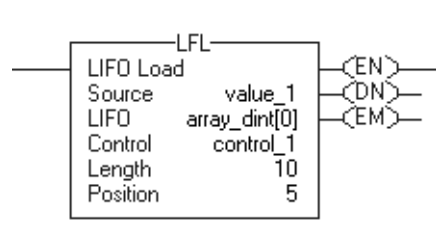
# LIFO Unload (LFU)

The LFU instruction unloads the value at .POS of the LIFO and stores 0 in that location.

## Operands:

### Relay Ladder

```
        ┌───LFU───────────┐
        │ LIFO Unload     │──(EU)──
        │ LIFO        ?   │──(DN)──
      ──│ Dest        ?   │──(EM)──
        │ Control     ?   │
        │ Length      ?   │
        │ Position    ?   │
        └─────────────────┘
```
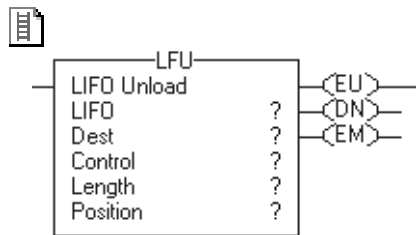
| Operand | Type | Format | Description |
|---|---|---|---|
| LIFO | SINT | array tag | LIFO to modify |
| | INT | | specify the first element of the LIFO |
| | DINT | | **do not** use CONTROL.POS in the subscript |
| | REAL | | |
| | string | | |
| | structure | | |
| Destination | SINT | tag | value 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:**

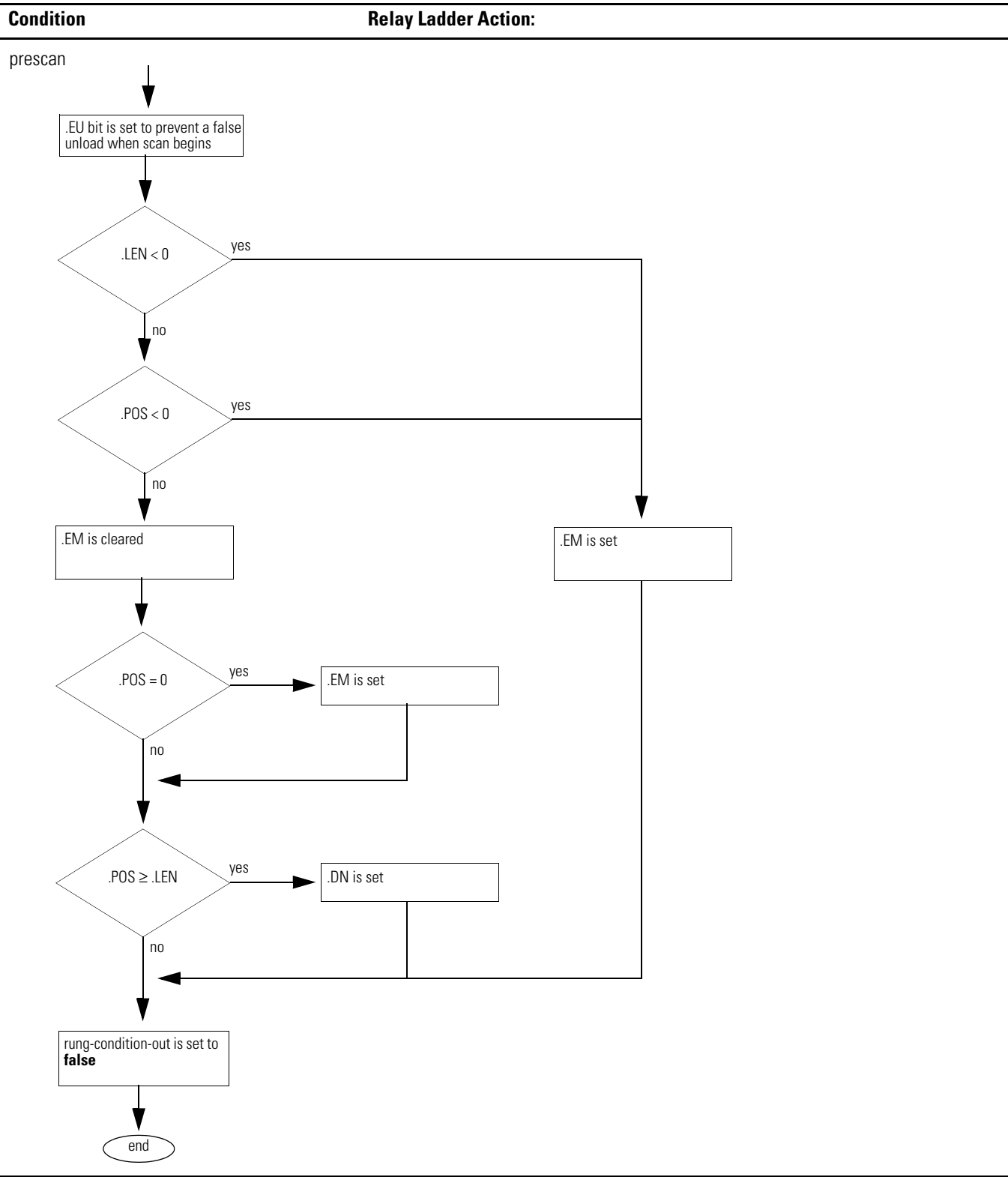| A Major Fault Will Occur If | Fault Type | Fault Code |
|---|---|---|
| Length > LIFO array size | 4 | 20 |

### Execution:

| Condition | Relay Ladder Action: |
|---|---|

prescan

```
         |
         v
  +---------------------+
  | .EU bit is set to   |
  | prevent a false     |
  | unload when scan    |
  | begins              |
  +---------------------+
         |
         v
      /  .LEN < 0  \  --- yes ------------------------+
      \           /                                   |
         | no                                         |
         v                                            |
      /  .POS < 0  \  --- yes ------------------------+
      \           /                                   |
         | no                                         |
         v                                            v
  +----------------+                        +----------------+
  | .EM is cleared |                        | .EM is set     |
  +----------------+                        +----------------+
         |                                           |
         v                                           |
      /  .POS = 0  \  --- yes ---> +----------+      |
      \           /                | .EM is set|      |
         | no <----------------+   +----------+      |
         v                                           |
      / .POS ≥ .LEN \ --- yes ---> +----------+      |
      \             /              | .DN is set|      |
         | no <----------------+   +----------+      |
         v <------------------------------------------+
  +---------------------+
  | rung-condition-out  |
  | is set to **false** |
  +---------------------+
         |
         v
      ( end )
```

| Condition | Relay Ladder Action: |
|---|---|

rung-condition-in is false

```
                    │
                    ▼
        ┌───────────────────────┐
        │ .EU bit is cleared     │
        └───────────────────────┘
                    │
                    ▼
              ╱─────────╲              yes
             ╱  .LEN < 0  ╲──────────────────────────┐
             ╲            ╱                            │
              ╲─────────╱                              │
                    │ no                               │
                    ▼                                  │
              ╱─────────╲              yes             │
             ╱  .POS < 0  ╲──────────────────────────┤
             ╲            ╱                            │
              ╲─────────╱                              │
                    │ no                               ▼
                    ▼                         ┌───────────────────┐
        ┌───────────────────────┐             │ .EM is set         │
        │ .EM is cleared         │             └───────────────────┘
        └───────────────────────┘                      │
                    │                                   │
                    ▼                                   │
              ╱─────────╲      yes    ┌──────────────┐  │
             ╱  .POS = 0  ╲──────────▶│ .EM is set    │  │
             ╲            ╱            └──────────────┘  │
              ╲─────────╱                    │          │
                    │ no ◀──────────────────┘          │
                    ▼                                   │
              ╱───────────╲    yes    ┌──────────────┐  │
             ╱ .POS ≥ .LEN ╲─────────▶│ .DN is set    │  │
             ╲             ╱          └──────────────┘  │
              ╲───────────╱                 │           │
                    │ no ◀──────────────────┴───────────┘
                    ▼
        ┌───────────────────────┐
        │ rung-condition-out is  │
        │ set to **false**       │
        └───────────────────────┘
                    │
                    ▼
                ( end )
```

| Condition | Relay Ladder Action: |
|---|---|

rung-condition-in is true

examine .EU bit

.EU = 0 → .EU bit is set → .LEN < 0

yes

.EU = 1

.LEN < 0
yes

no

.POS < 0
yes

no

.EM bit is cleared

.POS = 0 → yes → .EM bit is set

no

.POS ≥ .LEN → yes → .DN bit is set

no

.EM bit is set

.LEN < 0
yes

no

.POS < 0
yes

no

.EM bit is cleared

.EM bit is set

.POS ≤1 → yes → .EM bit is set

no

.POS < 1 → yes → Destination = 0

no

.POS > .LEN → yes → .POS = .LEN

no

.POS = .POS -1

.LEN > size of array → yes → major fault

no

Destination = LIFO[control.POS]

rung-condition-out is set to **true**

end

| postscan | The rung-condition-out is set to false. |
|---|---|

**Example:**  When enabled, the LFU instruction unloads *array_dint[5]* into *value_2*.

```
--------LFU--------
LIFO Unload            (EU)
LIFO      array_dint[0] (DN)
Dest          value_2   (EM)
Control      control_1
Length            10
Position           6
```

before LIFO unload | after LIFO unload

*array_dint[0]*

| before LIFO unload | after LIFO unload |
|---|---|
| 00000 | 00000 |
| 11111 | 11111 |
| 22222 | 22222 |
| 33333 | 33333 |
| 44444 | 44444 |
| 55555 | 00000 |
| 00000 | 00000 |
| 00000 | 00000 |
| 00000 | 00000 |
| 00000 | 00000 |

*array_dint[5]* → 55555

*control_1.pos* = 6

*control_1.pos* = 5
*value_2* = 55555
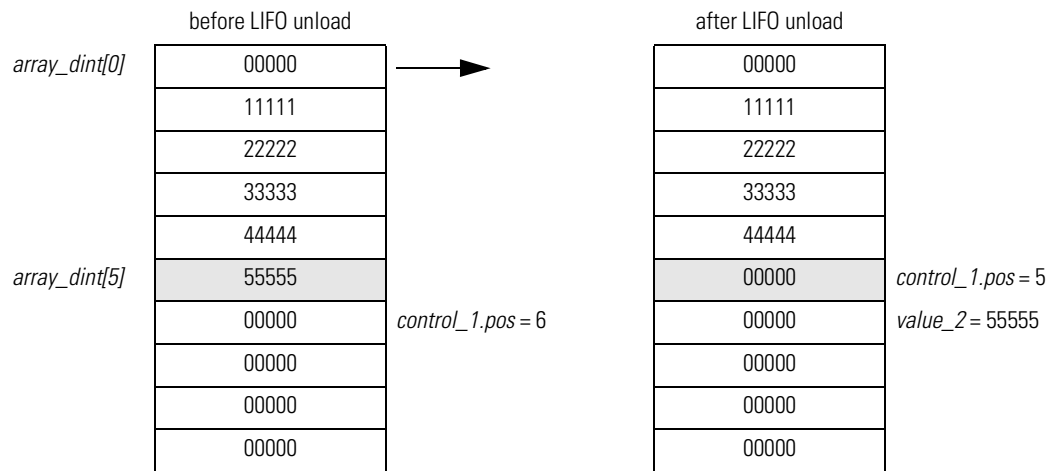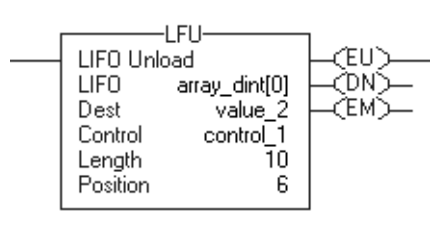
**Notes:**

# Sequencer Instructions

## (SQI, SQO, SQL)

## Introduction

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

| If You Want To | Use This Instruction | Available In These Languages | See Page |
|---|---|---|---|
| Detect when a step is complete. | SQI | relay ladder | 420 |
| Set output conditions for the next step. | SQO | relay ladder | 424 |
| Load reference conditions into sequencer arrays | SQL | relay ladder | 428 |

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

# Sequencer Input (SQI)

The SQI instruction detects when a step is complete in a sequence pair of SQO/SQI instructions.

## Operands:



### Relay Ladder

| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| Array | DINT | array tag | sequencer array |
| | | | specify the first element of the sequencer array |
| | | | **do not** use CONTROL.POS in the subscript |
| Mask | SINT | tag | which bits to block or pass |
| | INT | immediate | |
| | **DINT** | | |
| | 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 ≤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.

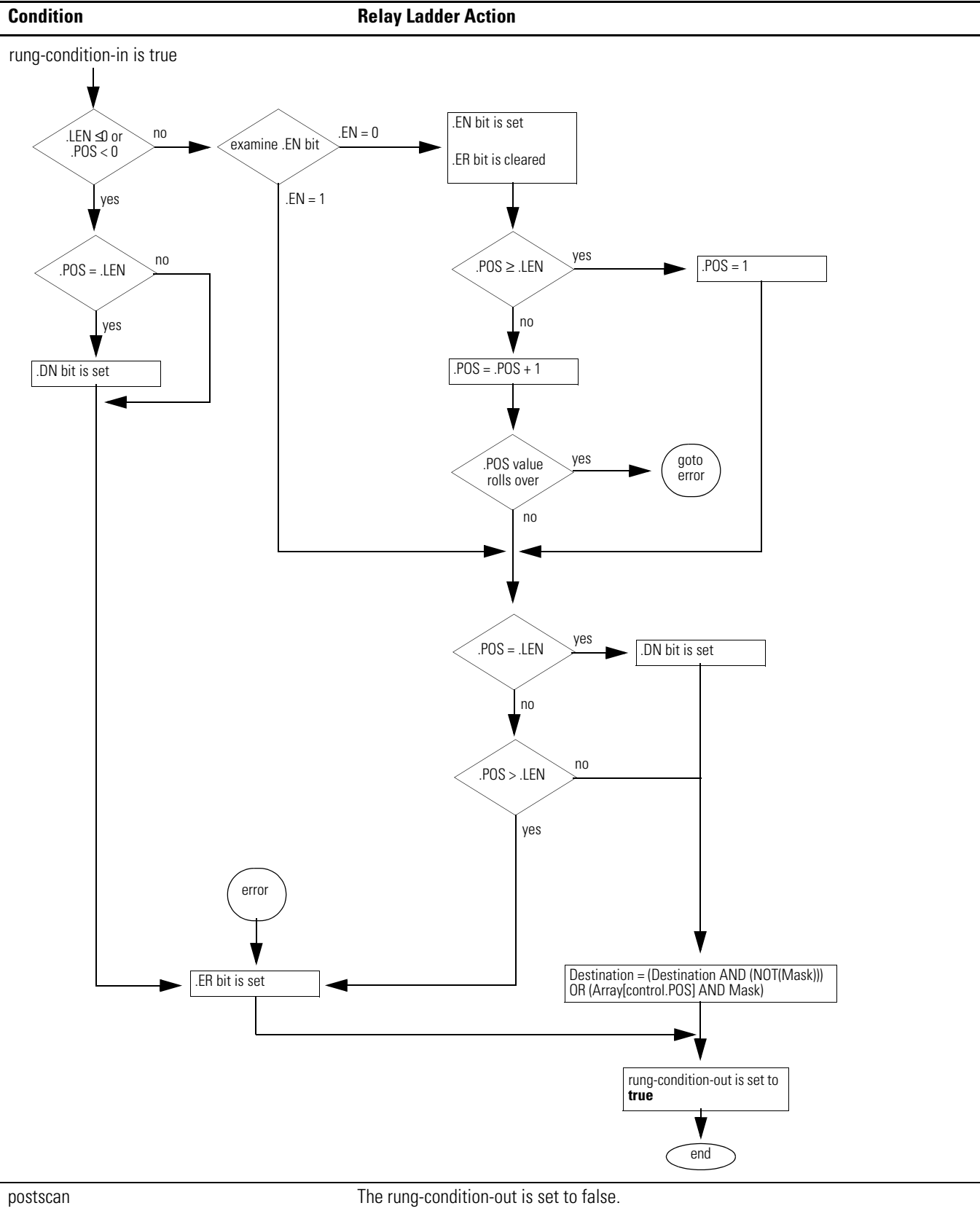| Prefix: | Description |
|---------|-------------|
| 16# | hexadecimal |
|     | for example; 16#0F0F |
| 8# | octal |
|    | for example; 8#16 |
| 2# | binary |
|    | for example; 2#00110011 |

**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

```
                        │
                        ▼
                  ╱──────────╲
                 ╱ .LEN ≤0     ╲        no
                ╱  .POS < 0     ╲────────────▶  ┌──────────────────┐
                ╲    or         ╱               │ .ER bit is cleared│
                 ╲ .POS > .LEN ╱                └──────────────────┘
                  ╲──────────╱                           │
                       │                                 │
                       │ yes                             ▼
                       ▼                           ╱──────────────╲
                ┌────────────┐                    ╱  masked Source  ╲
                │ .ER bit is set│      no        ╱        =          ╲
                └────────────┘  ◀──────────────  ╲ masked Array[.POS] ╱
                       │                           ╲────────────────╱
                       │                                  │
                       │                                  │ yes
                       ▼                                  ▼
         ┌────────────────────┐            ┌────────────────────┐
         │ rung-condition-out │            │ rung-condition-out │
         │ is set to          │            │ is set to          │
         │ **false**          │            │ **true**           │
         └────────────────────┘            └────────────────────┘
                       │                                  │
                       └──────────────┬───────────────────┘
                                      ▼
                                   ( end )
```

| | |
|---|---|
| postscan | The rung-condition-out is set to false. |

**Example:**   When enabled, the SQI instruction passes *value_2* through the mask to determine whether the result is equal to the current element in *array_dint*. The masked comparison is true, so the rung-condition-out goes true.

```
            ┌───SQI──────────────────┐
────────────┤ Sequencer Input        ├──────────
            │ Array      array_dint[0]│
            │ Mask          16#0F0F   │
            │ Source        value_2   │
            │ Control       control_1 │
            │ Length          10 ←    │
            │ Position         0 ←    │
            └────────────────────────┘
```
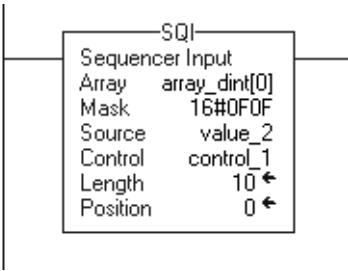
| SQI Operand | Example Values (DINTs Displayed In Binary) |
|---|---|
| Source | **xxxxxxxx xxxxxxxx xxxx0101 xxxx1010** |
| Mask | **00000000 00000000 00001111 00001111** |
| Array | **xxxxxxxx xxxxxxxx xxxx0101 xxxx1010** |

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.

```
    ┌───SQI────────────────┐                          ┌───ADD──────────────────┐
────┤ Sequencer Input      ├──────────────────────────┤ Add                     ├───
    │ Array      array_1[0]│                          │ Source A  control_1.pos │
    │ Mask    16#0000f0ff   │                          │                 0 ←     │
    │ Source      value_1   │                          │ Source B          1     │
    │ Control     control_1 │                          │                         │
    │ Length         10 ←   │                          │ Dest      control_1.pos │
    │ Position        0 ←   │                          │                 0 ←     │
    └──────────────────────┘                          └────────────────────────┘

    ┌───GRT────────────────┐                          ┌───MOV──────────────────┐
────┤ Greater Than (A>B)   ├──────────────────────────┤ Move                    ├───
    │ Source A  control_1.pos│                         │ Source            1     │
    │               0 ←      │                         │                         │
    │ Source B  control_1.len│                         │ Dest  control_1.pos     │
    │              10 ←      │                         │                 0 ←     │
    └──────────────────────┘                          └────────────────────────┘
```
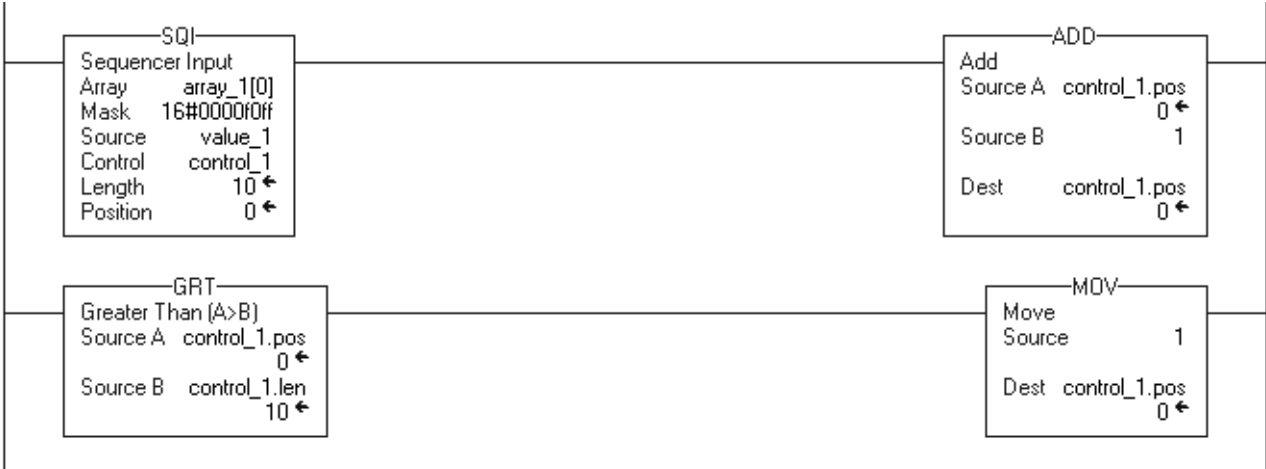
# Sequencer Output (SQO)

The SQO instruction sets output conditions for the next step of a sequence pair of SQO/SQI instructions.

## Operands:

```
              -SQO-
         Sequencer Output        -(EN)-
         Array           ?
         Mask            ?        -(DN)-
         Dest            ?
         Control         ?
         Length          ?
         Position        ?
```

### Relay Ladder

| Operand | Type | Format | Description |
|---|---|---|---|
| Array | DINT | array tag | sequencer array |
| | | | specify the first element of the sequencer array |
| | | | **do not** use CONTROL.POS in the subscript |
| Mask | SINT | 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. |

| Condition | Relay Ladder Action |
|---|---|
| rung-condition-in is true | |

rung-condition-in is true

```
              ┌─────────────┐
              │ .LEN ≤0 or  │  no      ┌──────────────┐  .EN = 0   ┌─────────────────┐
              │ .POS < 0    │─────────▶│ examine .EN   │──────────▶│ .EN bit is set  │
              └─────────────┘          │ bit           │           │                 │
                    │ yes              └──────────────┘           │ .ER bit is      │
                    │                        │ .EN = 1            │ cleared         │
                    ▼                        │                    └─────────────────┘
              ┌─────────────┐  no            │                           │
              │ .POS = .LEN │──────┐         │                           ▼
              └─────────────┘      │         │                    ┌──────────────┐  yes   ┌──────────┐
                    │ yes          │         │                    │ .POS ≥ .LEN  │───────▶│ .POS = 1 │
                    ▼              │         │                    └──────────────┘        └──────────┘
              ┌──────────────┐     │         │                           │ no                  │
              │ .DN bit is   │     │         │                           ▼                     │
              │ set          │     │         │                    ┌──────────────┐             │
              └──────────────┘     │         │                    │ .POS=.POS+1  │             │
                    │          ◀───┘         │                    └──────────────┘             │
                    │                        │                           │                     │
                    │                        │                           ▼                     │
                    │                        │                    ┌──────────────┐ yes ┌──────┐│
                    │                        │                    │ .POS value   │────▶│ goto ││
                    │                        │                    │ rolls over   │     │ error││
                    │                        │                    └──────────────┘     └──────┘│
                    │                        │                           │ no                  │
                    │                        └───────────────────────────┤◀────────────────────┘
                    │                                                    ▼
                    │                                             ┌──────────────┐ yes ┌──────────────┐
                    │                                             │ .POS = .LEN  │────▶│ .DN bit is   │
                    │                                             └──────────────┘     │ set          │
                    │                                                    │ no           └──────────────┘
                    │                                                    ▼                    │
                    │                                             ┌──────────────┐ no         │
                    │                                             │ .POS > .LEN  │────┐       │
                    │                                             └──────────────┘    │       │
                    │                                                    │ yes        │       │
                    │     ┌───────┐                                      │            │       │
                    │     │ error │                                      │            │       │
                    │     └───────┘                                      │            │       │
                    │         │                                          │            │       │
                    │         ▼                                          ▼            ▼       │
                    │   ┌──────────────┐◀──────────────────────  ┌─────────────────────────┐ │
                    └──▶│ .ER bit is   │                         │ Destination =            │ │
                        │ set          │                         │ (Destination AND         │ │
                        └──────────────┘                         │ (NOT(Mask))) OR          │ │
                              │                                   │ (Array[control.POS]      │ │
                              │                                   │ AND Mask)                │ │
                              │                                   └─────────────────────────┘ │
                              └──────────────────────────┐              │◀───────────────────┘
                                                         ▼              ▼
                                                  ┌─────────────────────────┐
                                                  │ rung-condition-out is    │
                                                  │ set to **true**          │
                                                  └─────────────────────────┘
                                                            │
                                                            ▼
                                                        ┌───────┐
                                                        │  end  │
                                                        └───────┘
```

| | |
|---|---|
| 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────┐
          │Sequencer Output      ─(EN)─
          │Array      array_dint[0]
          │Mask        16#0F0F    ─(DN)─
          │Dest         value_1
          │Control     control_1
          │Length          10 ←
          │Position         0 ←
          └───────────┘
```
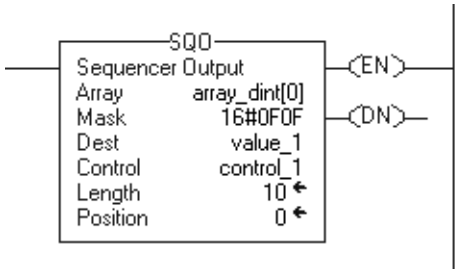
| SQO Operand | Example Values (Using INTS Displayed In Binary) |
|---|---|
| Array | **xxxxxxxx xxxxxxxx xxxx0101 xxxx1010** |
| Mask | **00000000 00000000 00001111 00001111** |
| Destination | **xxxxxxxx xxxxxxxx xxxx0101 xxxx1010** |

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

## Using SQI with SQO

If you pair an SQI instruction with an SQO instruction, make sure that both instructions use the same Control, Length, and Position values,.

```
┌────SQI────┐                                      ┌────SQO────┐
│Sequencer Input                                   │Sequencer Output      ─(EN)─
│Array      array_dint[0]                          │Array      array_dint[0]
│Mask        16#0F0F                               │Mask        16#0F0F    ─(DN)─
│Source       value_2                              │Dest         value_1
│Control     control_1                             │Control     control_1
│Length          10 ←                              │Length          10 ←
│Position         0 ←                              │Position         0 ←
└───────────┘                                      └───────────┘
```
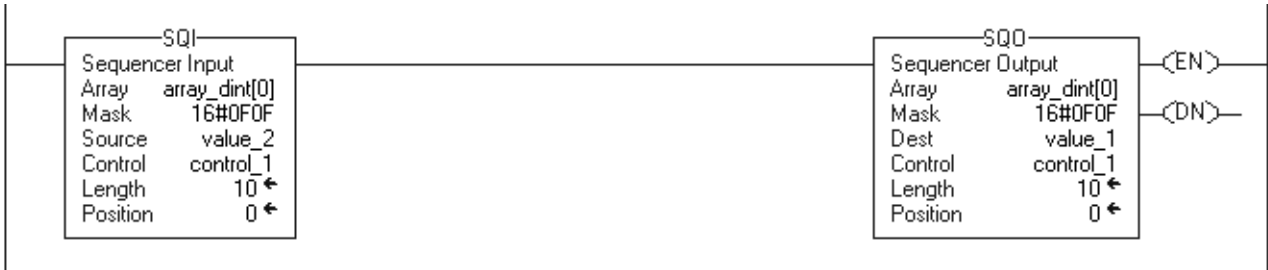
## Resetting the position of SQO

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.

```
  S:FS                                                       control_1
──┤ ├───────────────────────────────────────────────────────(RES)──
```

# Sequencer Load (SQL)

The SQL instruction loads reference conditions into a sequencer array.

## Operands:
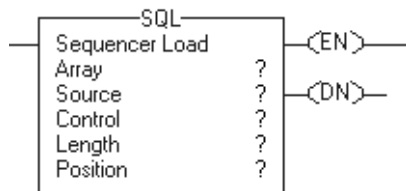


### Relay Ladder

| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| Array | DINT | array tag | sequencer array |
| | | | specify the first element of the sequencer array |
| | | | **do not** use CONTROL.POS in the subscript |
| Source | SINT | tag | input data to load into the sequencer array |
| | INT | immediate | |
| | **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 SQI and SQO instructions |
| Length | DINT | immediate | number of elements in the Array (sequencer table) to load |
| Position | DINT | immediate | current position in the array |
| | | | initial value is typically 0 |

### CONTROL Structure

| Mnemonic | Data Type | Description |
|----------|-----------|-------------|
| .EN | BOOL | The enable bit indicates that the SQL instruction is enabled. |
| .DN | BOOL | The done bit is set when all the specified elements have been loaded into Array. |
| .ER | BOOL | The error bit is set when .LEN  ≤0, .POS < 0, or .POS > .LEN. |
| .LEN | DINT | The length specifies the number of steps in the sequencer array. |
| .POS | DINT | The position identifies the element that the controller is currently manipulating. |

**Description:** When enabled, the SQL instruction increments to the next position in the sequencer array and loads the Source value into that position. If the .DN bit is set or if .POS ≥ .LEN, the instruction sets .POS=1.

Typically use the same CONTROL structure as the SQI and SQO instructions.

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

**Arithmetic Status Flags:** not affected

**Fault Conditions:**

| A Major Fault Will Occur If | Fault Type | Fault Code |
|---|---|---|
| Length > size of Array | 4 | 20 |

**Execution:**

| Condition | Relay Ladder Action |
|---|---|
| prescan | The .EN bit is set to prevent a false load when the program scan begins. |
| | The rung-condition-out is set to false. |
| rung-condition-in is false | The .EN bit is cleared. |
| | The rung-condition-out is set to false. |

| Condition | Relay Ladder Action |
|---|---|

rung-condition-in is true

.LEN ≤ 0 or .POS < 0

no

examine .EN bit

.EN = 0

.EN bit is set

.ER bit is cleared

yes

.EN = 1

.POS = .LEN

no

yes

.DN bit is set

.POS ≥ .LEN

yes

.POS = 1

no

.POS = .POS + 1

.POS value rolls over

yes

goto error

no

.POS = .LEN

yes

.DN bit is set

no

.POS > .LEN

no

yes

.LEN > size of array

yes

error

major fault

.ER bit is set

no

Array[control.POS] = Source

rung-condition-out is set to **true**

end

| postscan | The rung-condition-out is set to false. |
|---|---|
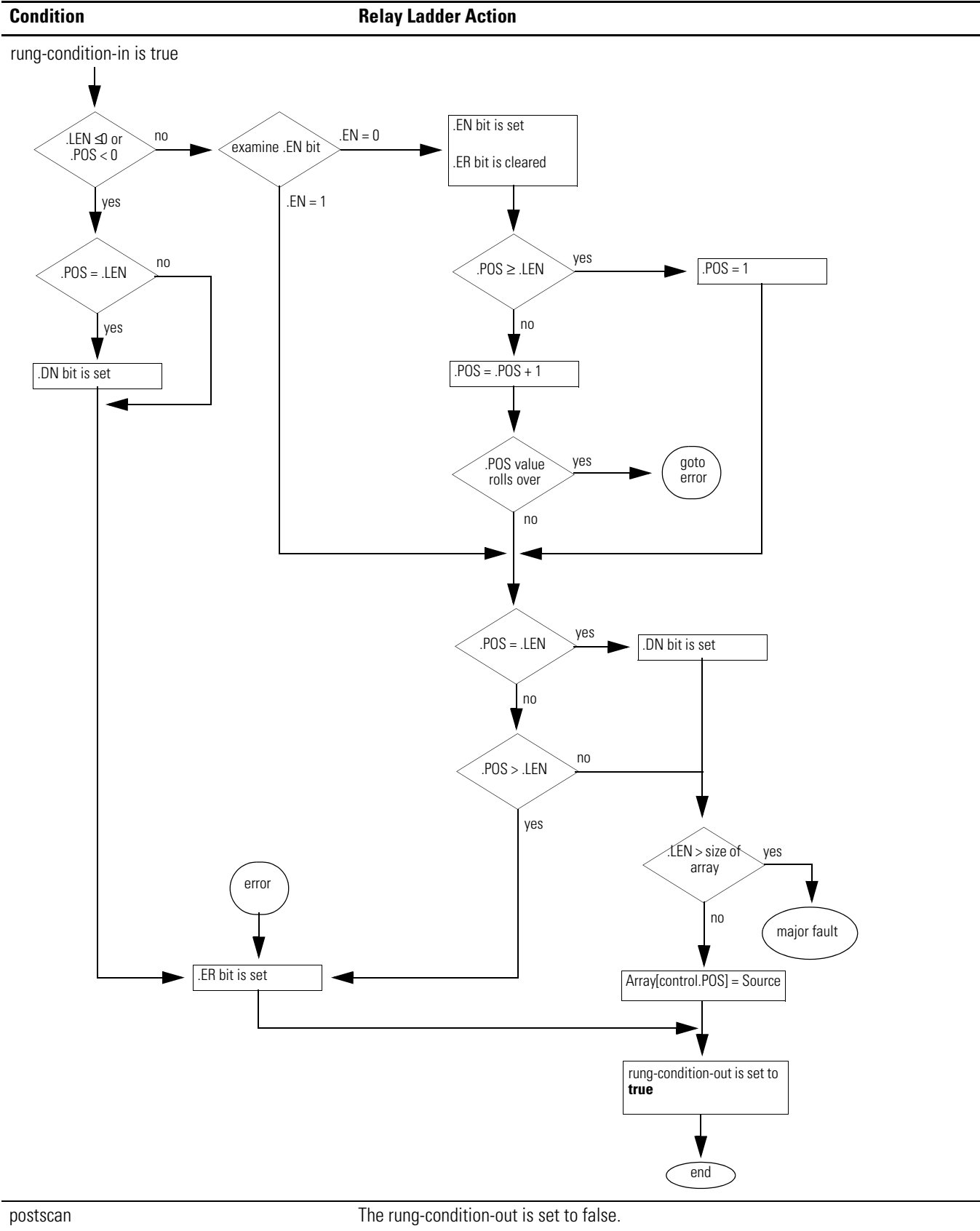
**Example:**     When enabled, the SQL instruction loads *value_3* into the next position in the sequencer array, which is *array_dint[5]* in this example.

```
-----SQL-----
Sequencer Load          --(EN)--
Array       array_dint[0]
Source            value_3   --(DN)--
Control          control_1
Length                  10
Position                 5
```



before load

| array_dint[0] | 00000 |
| | 11111 |
| | 22222 |
| | 33333 |
| | 44444 |
| array_dint[5] | 00000 |
| | 00000 |
| | 00000 |
| | 00000 |
| | 00000 |

*control_1.pos* = 5

*value_3* = 55555

after load

| 00000 |
| 11111 |
| 22222 |
| 33333 |
| 44444 |
| 55555 |
| 00000 |
| 00000 |
| 00000 |
| 00000 |

*control_1.pos* = 6
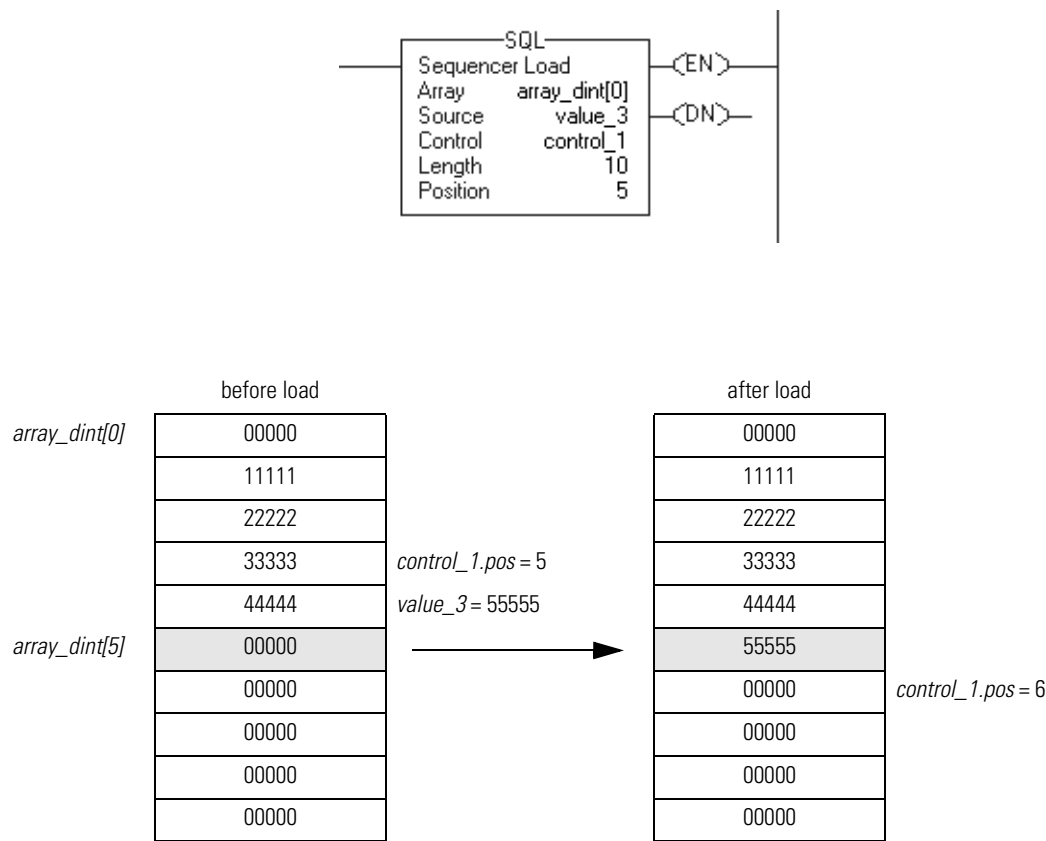
**Notes:**

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

## Introduction

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

| If You Want To | Use This Instruction | Available In These Languages | See Page |
|---|---|---|---|
| Jump over a section of logic that does not always need to be executed. | JMP LBL | relay ladder | 434 |
| Jump to a separate routine, pass data to the routine, execute the routine, and return results. | JSR SBR RET | relay ladder function block structured text | 436 |
| Jump to an external routine (SoftLogix5800 controller only) | JXR | relay ladder | 447 |
| Mark a temporary end that halts routine execution. | TND | relay ladder structured text | 450 |
| Disable all the rungs in a section of logic. | MCR | relay ladder | 452 |
| Disable user tasks. | UID | relay ladder structured text | 454 |
| Enable user tasks. | UIE | relay ladder structured text | 454 |
| Disable a rung. | AFI | relay ladder | 456 |
| Insert a placeholder in the logic. | NOP | relay ladder | 457 |
| End a transition for a sequential function chart. | EOT | relay ladder structured text | 458 |
| Pause a sequential function chart. | SFP | relay ladder structured text | 460 |
| Reset a sequential function chart. | SFR | relay ladder structured text | 462 |
| Trigger the execution of an event task | EVENT | relay ladder structured text | 464 |

# Jump to Label (JMP) Label (LBL)

The JMP and LBL instructions skip portions of ladder logic.

### Operands:

⬚

—(JMP)—

—[ LBL ]—
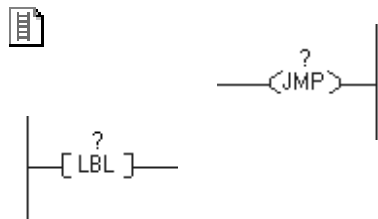
#### Relay Ladder

| 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.<br><br>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*.

label_20
(JMP)

*[other rungs of code]*

label_20
[ LBL ]
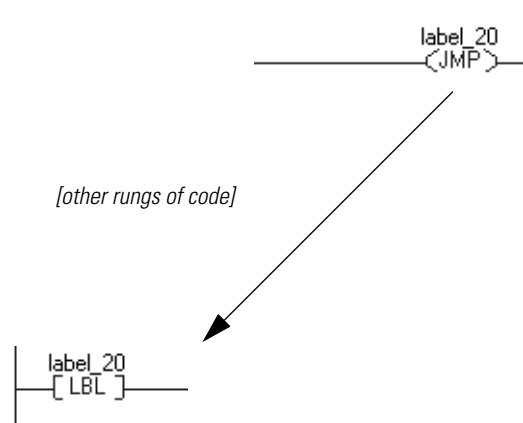
# Jump to Subroutine (JSR) Subroutine (SBR) Return (RET)

The JSR instruction jumps execution to a different routine. The SBR and RET instructions are optional instructions that exchange data with the JSR instruction.

### JSR Operands:

```
          -JSR-
  Jump to Subroutine
  Routine name          ?
  Input par             ?
  Return par            ?
```

### Relay Ladder

| Operand | Type | Format | Description |
|---|---|---|---|
| Routine name | ROUTINE | name | routine to execute (that is, subroutine) |
| Input parameter | BOOL<br><br>SINT<br><br>INT<br><br>DINT<br><br>REAL<br><br>structure | immediate<br><br>tag<br><br>array tag | data from this routine that you want to copy to a tag in the subroutine<br><br>• Input parameters are optional.<br>• Enter multiple input parameters, if needed. |
| Return parameter | BOOL<br><br>SINT<br><br>INT<br><br>DINT<br><br>REAL<br><br>structure | tag<br><br>array tag | tag in this routine to which you want to copy a result of the subroutine<br><br>• Return parameters are optional.<br>• Enter multiple return parameters, if needed. |

### JSR Operands Continued:

```
JSR(RoutineName,InputCount,
InputPar,ReturnPar);
```

**Structured Text**

| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| Routine name | ROUTINE | name | routine to execute (that is, subroutine) |
| Input count | SINT<br>INT<br>DINT<br>REAL | immediate | number of input parameters |
| Input parameter | BOOL<br>SINT<br>INT<br>DINT<br>REAL<br>structure | immediate<br>tag<br>array tag | data from this routine that you want to copy to a tag in the subroutine<br><br>• Input parameters are optional.<br>• Enter multiple input parameters, if needed. |
| Return parameter | BOOL<br>SINT<br>INT<br>DINT<br>REAL<br>structure | tag<br>array tag | tag in this routine to which you want to copy a result of the subroutine<br><br>• Return parameters are optional.<br>• Enter multiple return parameters, if needed. |

**Function Block**



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

| | |
|---|---|
| **ATTENTION** | For each parameter in a SBR or RET instruction, use the same data type (including any array dimensions) as the corresponding parameter in the JSR instruction. Using different data types may produce unexpected results. |

**SBR Operands:** The SBR instruction must be the first instruction in a relay ladder or structured text routine.
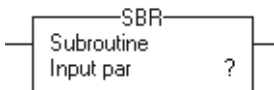
### Relay Ladder

| Operand | Type | Format | Description |
|---|---|---|---|
| Input parameter | BOOL | 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.
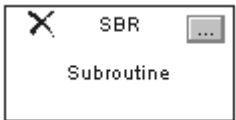
### Function Block
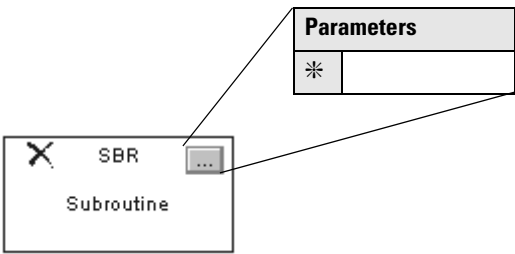
The operands are the same as those for the relay ladder SBR instruction.

### RET Operands:

#### Relay Ladder

RET
Return
Return par                ?
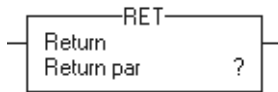
| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| Return parameter | BOOL | immediate | data from this routine that you want to copy to the corresponding return parameter in the JSR instruction |
| | SINT | tag | |
| | INT | array tag | |
| | DINT | | |
| | REAL | | |
| | structure | | |

#### Structured Text

`RET(ReturnPar);`

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

#### Function Block

RET    ...
Return

| Parameters |
|------------|
| ✳ |

RET    ...
Return

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.

Calling Routine                    Subroutine

**JSR**

1. If you want to copy data to a tag in the subroutine, enter an input parameter.

2. If you want to copy a result of the subroutine to a tag in this routine, enter a return parameter.

3. Enter as many input and return parameters as you need.
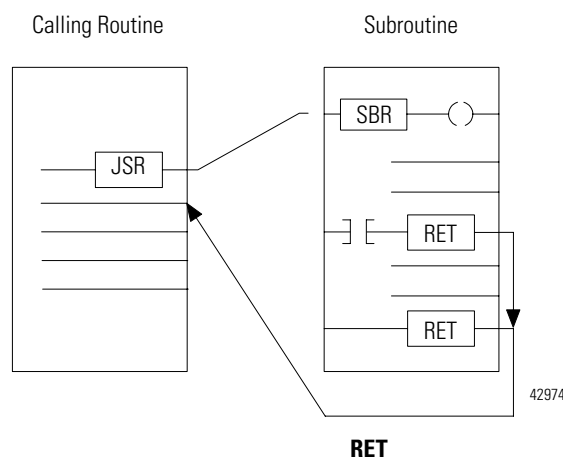
**SBR**

1. If the JSR instruction has an input parameter, enter an SBR instruction.

2. Place the SBR instruction as the first instruction in the routine.

3. For each input parameter in the JSR instruction, enter the tag into which

42974

**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.

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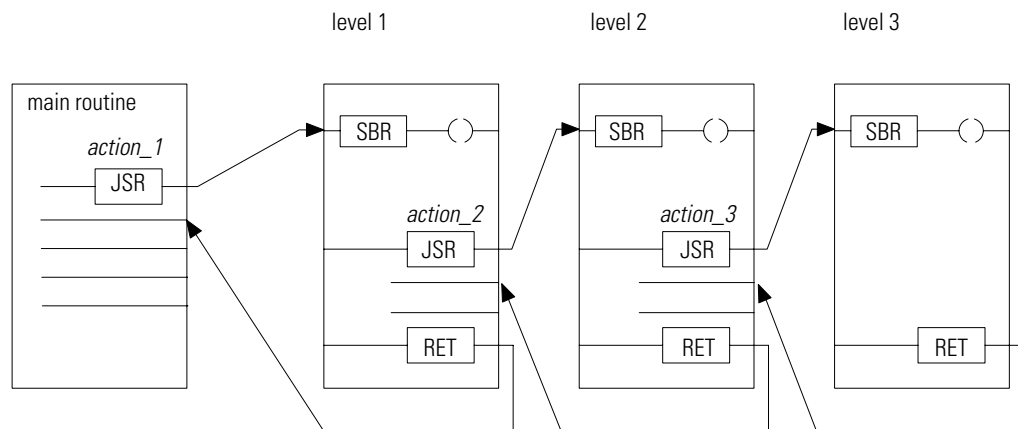
**Arithmetic Status Flags:** Arithmetic status flags are affected.

**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.)<br><br>• Release 6.x and earlier, input and return parameters are passed.<br>• Release 7.x and later, input and return parameters are not passed.<br><br>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.<br><br>The rung-condition-out is set to false (relay ladder only). | |
| rung-condition-in is false to the JSR instruction | The subroutine *does not* execute.<br><br>Outputs in the subroutine remain in their last state.<br><br>The rung-condition-out is set to false. | na |
| rung-condition-in is true | The instruction executes.<br><br>The rung-condition-out is set to true. | na |
| EnableIn is set | na | EnableIn is always set.<br><br>The instruction executes. |

| Condition | Relay Ladder Action | Structured Text Action |
|---|---|---|

instruction execution

| postscan | Same action as prescan described above. | Same action as prescan described above. |
|---|---|---|

## Function Block

| Condition: | Action |
| --- | --- |
| prescan | No action taken. |
| instruction first scan | No action taken. |
| instruction first run | No action taken. |
| normal execution | 1. If the routine contains an SBR instruction, the controller first executes the SBR instruction.<br><br>2. The controller latches all data values in IREFs.<br><br>3. The controller executes the other function blocks in the order that is determined by their wiring. This includes other JSR instructions.<br><br>4. The controller writes outputs in OREFs.<br><br>5. If the routine contains an RET instruction, the controller executes the RET instruction last. |
| postscan | The subroutine is called.<br><br>If the routine is an SFC routine, the routine in initialized the same as it is during prescan. |

**Example 1:**    The JSR instruction passes *value_1* and *value_2* to *routine_1*.

The SBR instruction receives *value_1* and *value_2* from the JSR instruction and copies those values to *value_a* and *value_b*, respectively. Logic execution continues in this routine.

The RET instruction sends *float_a* to the JSR instruction. The JSR instruction receives *float_a* and copies the value to *float_value_1*. Logic execution continues with the next instruction following the JSR instruction.

### Relay Ladder

| Routine: | Program |
|---|---|
| Main routine | JSR — Jump to Subroutine; Routine name routine_1; Input par value_1; Input par value_2; Return par float_value_1 |
| Subroutine | SBR — Subroutine; Input par value_a; Input par value_b
*[other rungs of code]*
RET — Return; Return par float_a |
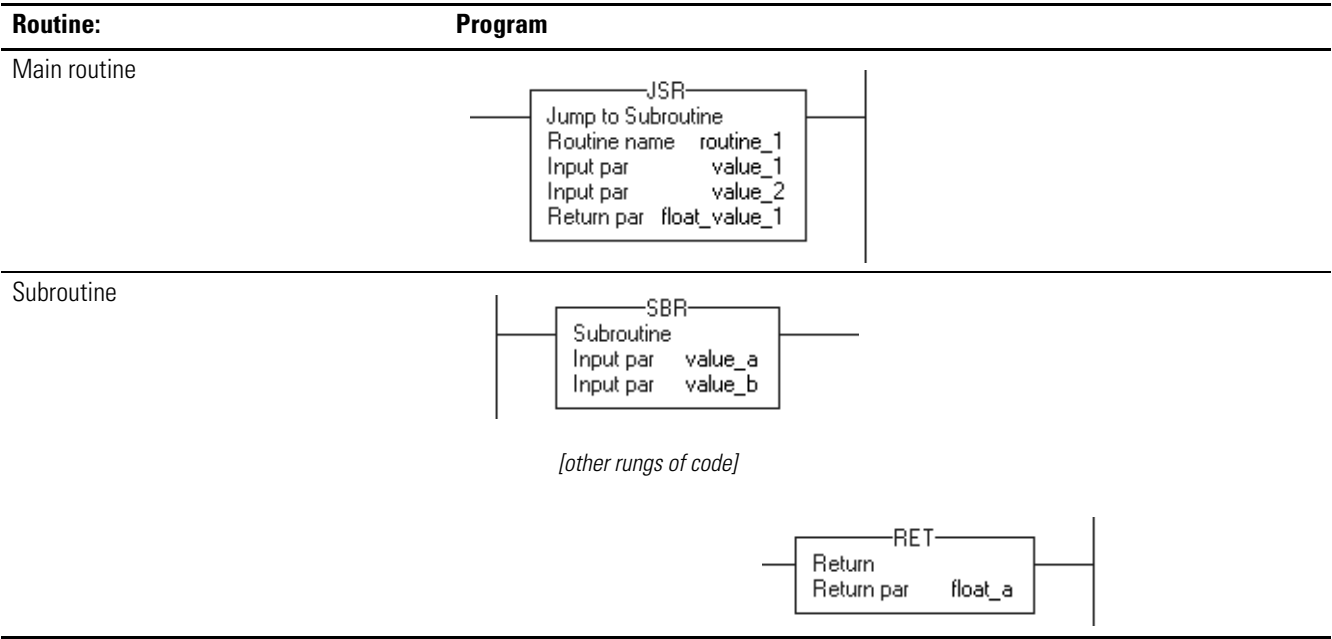
### Structured Text

| 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.*

```
   abc                                                    ┌──────────JSR──────────┐
───┤ ├──────────────────────────────────────────────────┤ Jump to Subroutine     ├───
                                                          │ Routine name  subroutine_1
                                                          │ Return par        cookies_1 │
                                                          └────────────────────────┘
```
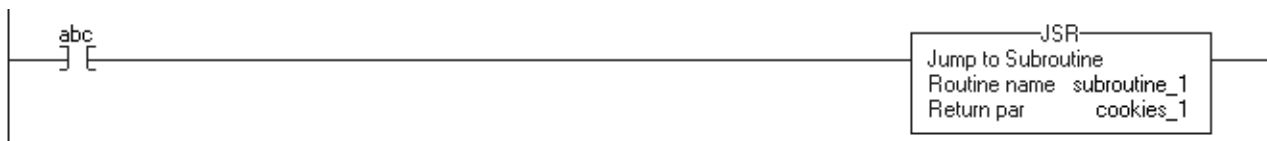
Adds the value in *cookies_1* to *cookies_2* and stores the result in *total_cookies*.

```
                                                          ┌──────────ADD──────────┐
──────────────────────────────────────────────────────────┤ Add                    ├───
                                                          │ Source A  cookies_2    │
                                                          │                   0 ←   │
                                                          │ Source B  cookies_1    │
                                                          │                   0 ←   │
                                                          │ Dest    total_cookies  │
                                                          │                   0 ←   │
                                                          └────────────────────────┘
```

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.

```
   def                                                    ┌──────────RET──────────┐
───┤ ├──────────────────────────────────────────────────┤ Return                 ├───
                                                          │ Return par    value_1  │
                                                          └────────────────────────┘
```
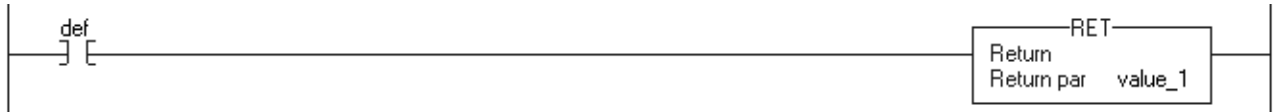
When *def* is off (previous rung) and *ghi* is on, the RET instruction returns *value_2* to the JSR *cookies_1* parameter and the rest of the subroutine is not scanned.

```
   ghi                                                    ┌──────────RET──────────┐
───┤ ├──────────────────────────────────────────────────┤ Return                 ├───
                                                          │ Return par    value_2  │
                                                          └────────────────────────┘
```
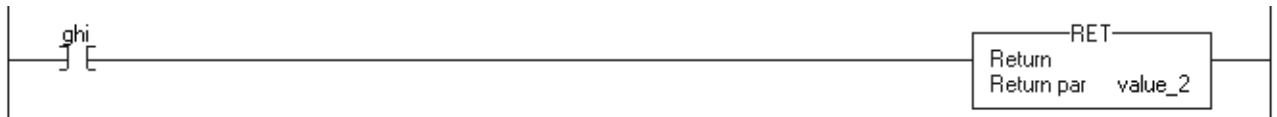
When both *def* and *ghi* are off (previous rungs), the RET instruction returns *value_3* to the JSR *cookies_1* parameter.

```
                                                          ┌──────────RET──────────┐
──────────────────────────────────────────────────────────┤ Return                 ├───
                                                          │ Return par    value_3  │
                                                          └────────────────────────┘
```
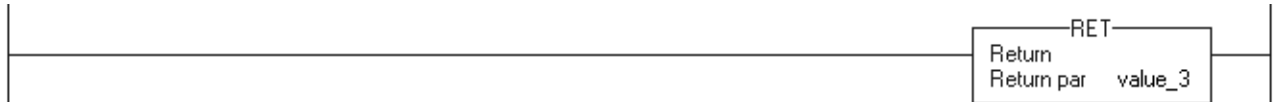
**Example 3:**

## Function Block

JSR instruction in *Routine_A*



42972

**1.** The values in *Add_Input_1*, *Add_Input_2*, and *Add_Input_3* are copied to *Input_A*, *Input_B*, and *Input_C*, respectively.

**3.** The value of *Sum_A_B_C* is copied to *Add_Three_Result*.

Function blocks of the *Add_Three_Inputs* routine



42973

**2.** The ADD instructions add *Input_A*, *Input_B*, and *Input_C* and place the result in *Sum_A_B_C*.
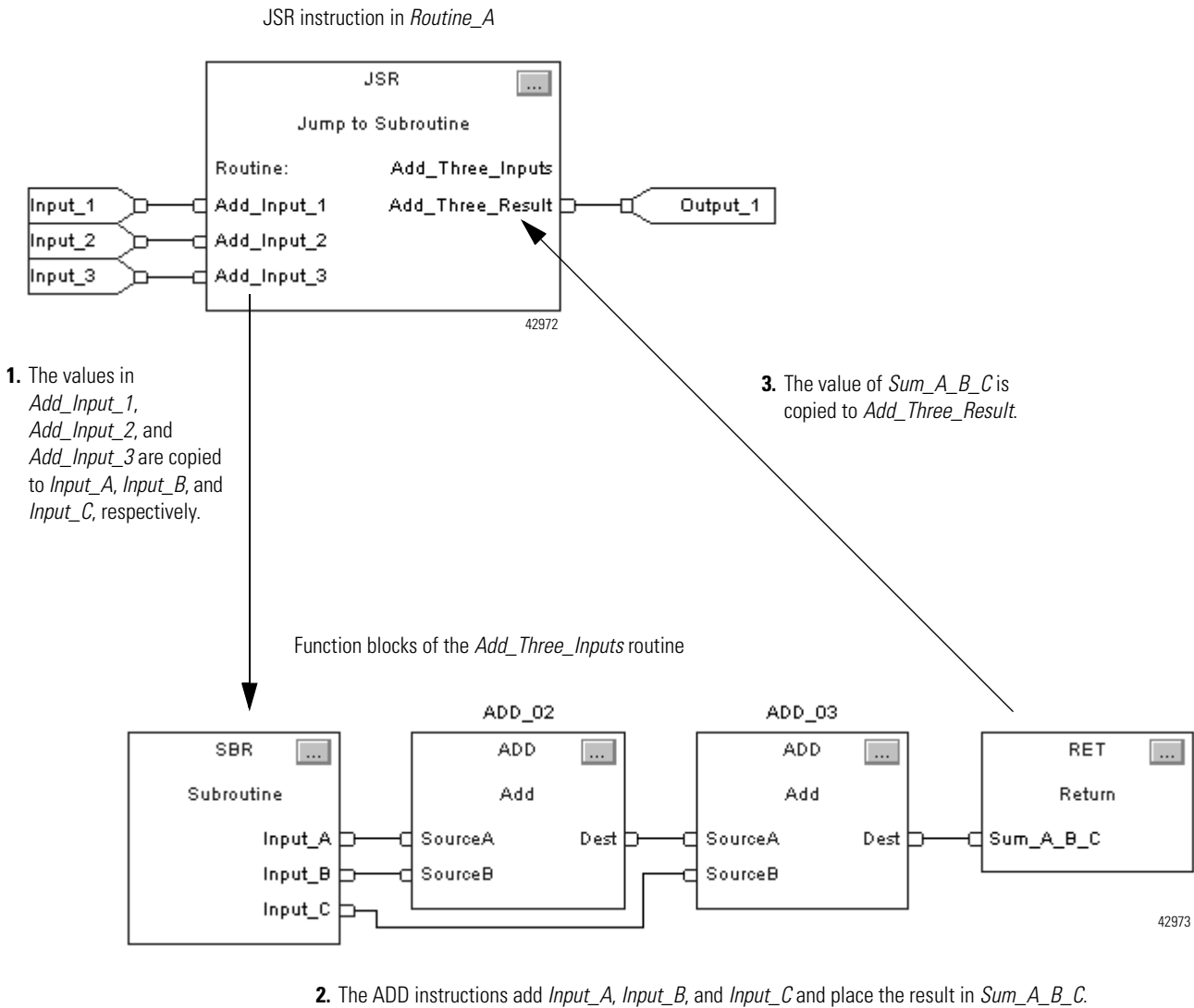
# Jump to External Routine (JXR)

The JXR instruction executes an external routine. This instruction is only supported by the SoftLogix5800 controllers.

## Operands:

### Relay Ladder



.
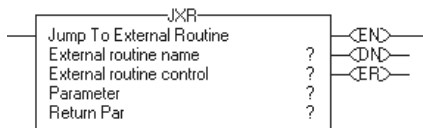
| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| External routine name | ROUTINE | name | external routine to execute |
| External routine control | EXT_ROUTINE_ CONTROL | tag | control structure (see the next page) |
| Parameter | BOOL | immediate | data from this routine that you want to copy to a variable in the external routine |
|  | SINT | tag | • Parameters are optional. |
|  | INT | array tag | • Enter multiple parameters, if needed. |
|  |  |  | • You can have as many as 10 parameters. |
|  | DINT |  |  |
|  | REAL |  |  |
|  | structure |  |  |
| Return parameter | BOOL | tag | tag in this routine to which you want to copy a result of the external routine |
|  | SINT |  | • The return parameter is optional. |
|  | INT |  | • You can have only one return parameter |
|  | 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_ROUTIN_ PARAMETERS | This value contains definitions of the return parameter from the external routine. There is only one return parameter. | Display only - this information is derived from the instruction entry. |
| EN | BOOL | When set, the enable bit indicates that the JXR instruction is enabled. | The external routine sets this bit. |
| ReturnsValue | BOOL | If set, this bit indicates that a return parameter was entered for the instruction. If cleared, this bit indicates that no return parameter was entered for the instruction. | Display only - this information is derived from the instruction entry. |
| DN | BOOL | The done bit is set when the external routine has executed once to completion. | The external routine sets this bit. |
| ER | BOOL | The error bit is set if an error occurs. The instruction stops executing until the program clears the error bit. | The external routine sets this bit. |
| FirstScan | BOOL | This bit identifies whether this is the first scan after switching the controller to Run mode. Use FirstScan to initialize the external routine, if needed. | The controller sets this bit to reflect scan status. |
| EnableOut | BOOL | Enable output. | The external routine sets this bit. |
| EnableIn | BOOL | Enable input. | The controller sets this bit to reflect rung-condition-in. The instruction executes regardless of rung condition. The developer of the external routine should monitor this status and act accordingly. |
| User1 | BOOL | These bits are available for the user. The controller does not initialize these bits. | Either the external routine or the user program can set these bits. |
| User0 | BOOL | | |
| ScanType1 | BOOL | These bits identify the current scan type: | The controller sets these bits to reflect scan status. |
| ScanType0 | BOOL | **Bit Values:**   **Scan Type:**<br><br>00      Normal<br><br>01      Pre Scan<br><br>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: |
|---|---|---|
| •an exception occurs in the external routine DLL | 4 | 88 |
| •the DLL could not be loaded | | |
| •the entry point was not found in the DLL | | |

**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**

📄

TND();

**Structured Text**

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.<br><br>The rung-condition-out is set to true. | na |
| EnableIn is set | na | EnableIn is always set.<br><br>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

```
    ───(TND)───┤
```
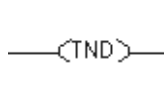
## Structured Text

```
TND();
```

# Master Control Reset (MCR)

The MCR instruction, used in pairs, creates a program zone that can disable all rungs within the MCR instructions.

## Operands:

### Relay Ladder

—(MCR)—

## 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:

## Execution:

| Condition | Relay Ladder Action |
|---|---|
| prescan | The rung-condition-out is set to false. |
| rung-condition-in is false | The rung-condition-out is set to false. |
| | The instructions in the zone are scanned, but the rung-condition-in is false and non-retentive outputs in the zone are disabled. |
| rung-condition-in is true | The rung-condition-out is set to true. |
| | The instructions in the zone are scanned normally. |
| postscan | The rung-condition-out is set to false. |

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

When the first MCR instruction is disabled (*input_1*, *input_2*, and *input_3* are not all set), the controller executes the rungs in the MCR zone (between the two MCR instructions) and the rung-condition-in goes false for all the rungs in the MCR zone, regardless of input conditions.

# User Interrupt Disable (UID) User Interrupt Enable (UIE)

The UID instruction and the UIE instruction work together to prevent a small number of critical rungs from being interrupted by other tasks.

## Operands:

### Relay Ladder

—<UID>—  —<UIE>—

### Structured Text

```
UID();
UIE();
```

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. | na |
|  | The rung-condition-out is set to true. |  |
| 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
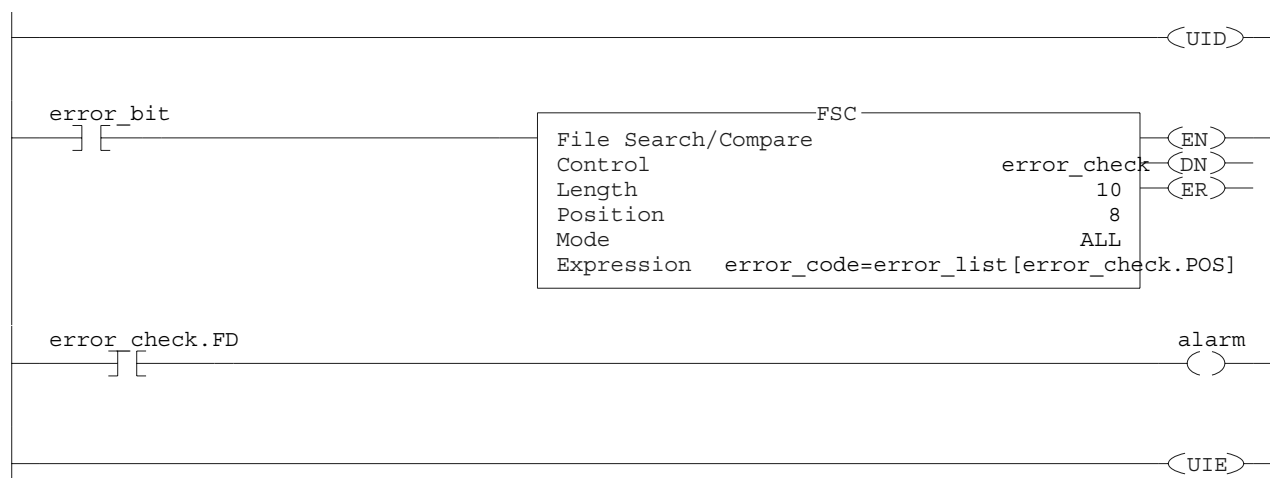


### Structured Text

```
UID();

    <statements>

UIE();
```

# Always False Instruction (AFI)

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

**Operands:**

📄     —[ AFI ]—

**Relay Ladder**

**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.

—[ AFI ]—
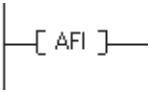
## No Operation (NOP)

The NOP instruction functions as a placeholder

### Operands:

**Relay Ladder**

—[NOP]—

**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

```
         ?
    —<EOT>—
```
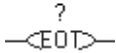
| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| data bit | BOOL | tag | state of the transition (0=executing, 1=completed) |

`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 ).

The Logix implementation of the EOT instruction differs from that in a PLC-5 controller. In a PLC-5 controller, the EOT instruction has no parameters. Instead, the PLC-5 EOT instruction returns rung condition as its state. In a Logix controller, the return parameter returns the transition state since rung condition is not available in all Logix programming languages.

**Arithmetic Status Flags:**  not affected

**Fault Conditions:**  none

## Execution:

| Condition | Relay Ladder Action: | Structured Text Action: |
|-----------|----------------------|-------------------------|
| prescan | The rung-condition-out is set to false. | No action taken. |
| rung-condition-in is false | The rung-condition-out is set to false. | na |
| rung-condition-in is true | The instruction executes. The rung-condition-out is set to true. | na |
| EnableIn is set | na | EnableIn is always set. The instruction executes. |
| instruction execution | The instruction returns the data bit value to the calling routine. | |
| postscan | The rung-condition-out is set to false. | No action taken. |

**Example:**    When both *limit_switch1* and *interlock_1* are set, set state. After *timer_1* completes, EOT returns the value of *state* to the calling routine.

### Relay Ladder
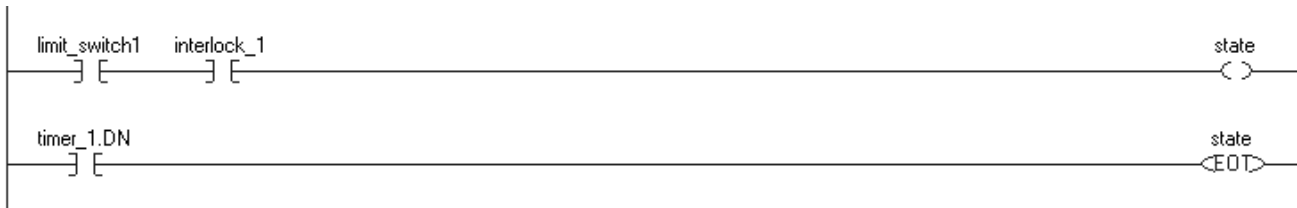


### Structured Text

```
state := limit_switch1 AND interlock_1;


IF timer_1.DN THEN

    EOT(state);

END_IF;
```

# SFC Pause (SFP)

The SFP instruction pauses an SFC routine.

## Operands:
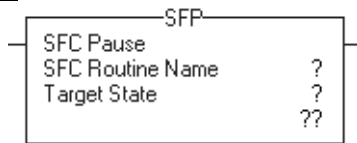
### Relay Ladder

```
        -SFP-
  SFC Pause
  SFC Routine Name      ?
  Target State          ?
                       ??
```

| 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) |

### Structured Text

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

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

**Description:** The SFP instruction lets you pause an executing SFC routine. If an SFC routine is in the paused state, use the SFP instruction again to change the state and resume execution of the routine.

Also, use the SFP instruction to resume SFC execution after using an SFR instruction (see ) to reset an SFC routine.

## Arithmetic Status Flags:

not affected

## Fault Conditions:

| A Major Fault Will Occur If: | Fault Type | Fault Code |
|------------------------------|-----------|-----------|
| the routine type is not an SFC routine | 4 | 85 |

### Execution:

| Condition: | Relay Ladder Action | Structured Text Action |
|---|---|---|
| prescan | The rung-condition-out is set to false. | No action taken. |
| rung-condition-in is false | The rung-condition-out is set to false. | na |
| rung-condition-in is true | The instruction executes. The rung-condition-out is set to true. | na |
| EnableIn is set | na | EnableIn is always set. The instruction executes. |
| instruction execution | The instruction pauses or resumes execution of the specified SFC routine. | |
| postscan | The rung-condition-out is set to false. | No action taken. |

**Example:**   If *sfc_en_p* is set, pause the SFC routine named *normal*. Restart the SFC when *sfc_en_e* is set.

### Relay Ladder

Pause the SFC routine.



Resume executing the SFC routine.



### Structured Text

Pause the SFC routine:
```
IF (sfp_en_p) THEN

     SFP(normal,paused);

     sfp_en_p := 0;

END_IF;
```

Resume executing the SFC routine:

```
IF (sfp_en_e) THEN

    SFP(normal,executing);

    sfp_en_e := 0;

END_IF;
```

## SFC Reset (SFR)

The SFR instruction resets the execution of a SFC routine at a specified step.

**Operands:**
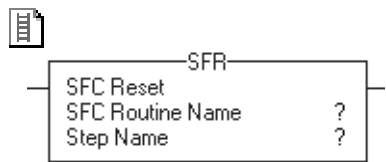
### Relay Ladder Operands

```
-------SFR-------
SFC Reset
SFC Routine Name          ?
Step Name                 ?
```

| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| SFCRoutine Name | ROUTINE | name | SFC routine to reset |
| Step Name | SFC_STEP | tag | target step where to resume execution |

### Structured Text

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

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

**Description:**  When the SFR instruction is enabled:

- In the specified SFC routine, all stored actions stop executing (reset).
- The SFC begins executing at the specified step.

If the target step is 0, the chart will be reset to its initial step

The Logix implementation of the SFR instruction differs from that in a PLC-5 controller. In the PLC-5 controller, the SFR executed when the rung condition was true. After reset, the SFC would remain paused until the rung containing the SFR became false. This allowed the execution following a reset to be delayed. This pause/un-pause feature of the PLC-5 SFR instruction was decoupled from the rung condition and moved into the SFP instruction.

**Arithmetic Status Flags:**  not affected

**Fault Conditions:**

| A Major Fault Will Occur If: | Fault Type | Fault Code |
|------------------------------|------------|------------|
| 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:

📄

```
----EVENT----
Trigger Event Task
Task            ?
```
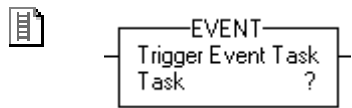
### Relay Ladder

| 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<br><br>SSV | Provides status information about the task. Once the controller sets a bit, you must manually clear the bit to determine if another fault of that type occurred. | |
| | | | **To determine if:** | **Examine this bit:** |
| | | | *An EVENT instruction triggered the task (event task only).* | *0* |
| | | | A 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.<br>The rung-condition-out is set to true. | na |
| EnableIn is set | na | EnableIn is always set.<br>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*

```
Shut_Down_Line    Shut_Down_Line_One_Shot                          ┌──EVENT──────────┐
   ] [────────────────────────[ONS]──────────────────────────────│Trigger Event Task│
                                                                   │Task  Shut_Down   │
                                                                   └──────────────────┘
```

*Program B*

```
Shut_Down_Line    Shut_Down_Line_One_Shot                          ┌──EVENT──────────┐
   ] [────────────────────────[ONS]──────────────────────────────│Trigger Event Task│
                                                                   │Task  Shut_Down   │
                                                                   └──────────────────┘
```
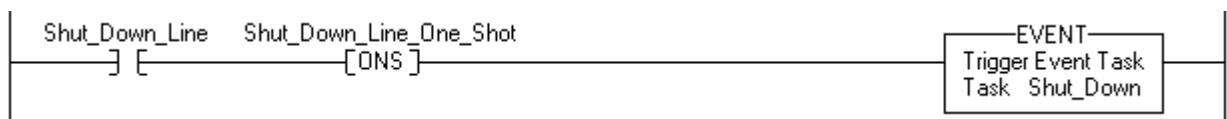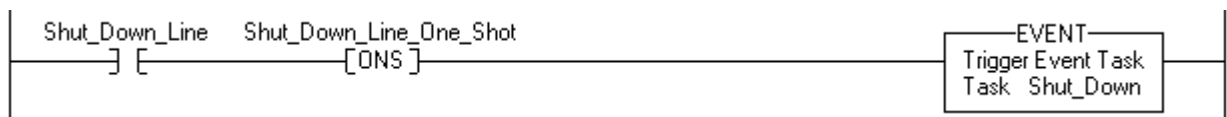
**Structured Text**

*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).

```
Initialize_Task_1    Storage.0                                              ┌──────EVENT──────┐
───┤ ├─────────────[ONS]──────────────────────────────────────────────────┤ Trigger Event Task │
                                                                           │ Task        Task_1 │
                                                                           └──────────────────┘
```
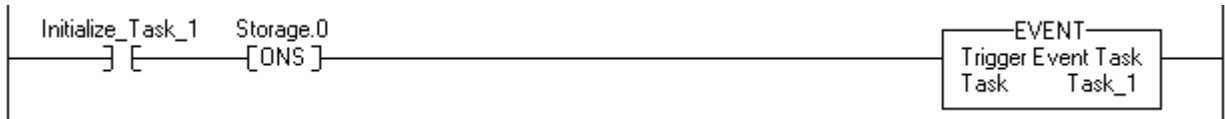
**Task_1 (event task)**

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*).

```
                                                          ┌─────────GSV─────────┐
                                                          │ Get System Value    │
                                                          │ Class Name     TASK │
                                                          │ Instance Name  THIS │
                                                          │ Attribute Name Status │
                                                          │ Dest     Task_Status │
                                                          │                 0 ← │
                                                          └─────────────────────┘
```

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.
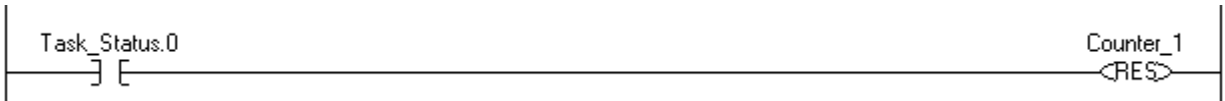
```
Task_Status.0                                                                    Counter_1
───┤ ├──────────────────────────────────────────────────────────────────────────<RES>───
```

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.

```
Task_Status.0                                        Task_Status.0    ┌─────────SSV─────────┐
───┤ ├───────────────────────────────────────────────────(U)─────────┤ Set System Value    │
                                                                     │ Class Name     TASK │
                                                                     │ Instance Name  THIS │
                                                                     │ Attribute Name Status │
                                                                     │ Source    Task_Status │
                                                                     │                 0 ← │
                                                                     └─────────────────────┘
```
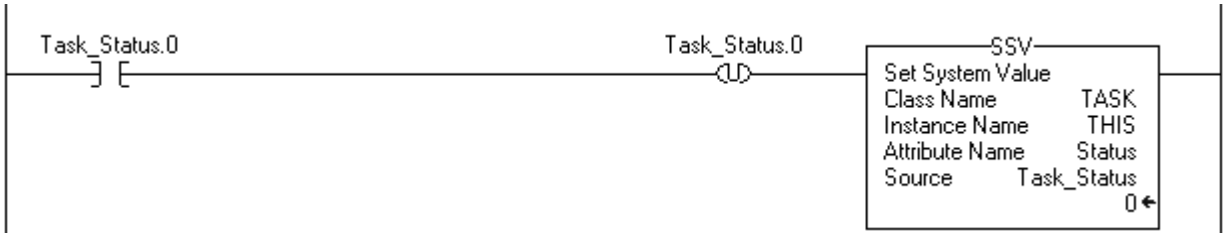
**Notes:**

# 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 | 470 |
| | FOR...DO[1] | structured text | |
| Terminate the repeated execution of a routine. | BRK | relay ladder | 473 |
| | EXIT[1] | structured text | |
| Return to the FOR instruction. | RET | relay ladder | 474 |

[1] Structured text only.

# For (FOR)

The FOR instruction executes a routine repeatedly.

## Operands:
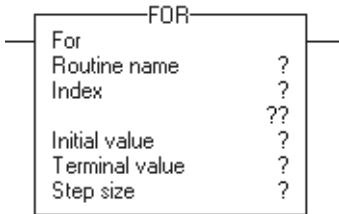
### Relay Ladder

| Operand | Type | Format | Description |
|---|---|---|---|
| Routine name | ROUTINE | routine name | routine to execute |
| Index | DINT | tag | counts how many times the routine has been executed |
| Initial value | SINT | 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 | | |

### Structured Text

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

    <statement>;

END_FOR;
```

Use the FOR...DO construct. See Appendix C, Structured Text Programming for information on structured text constructs.

## Description:

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

- You can put a FOR instruction in the main routine or any other routine.
- If you use a FOR instruction to call the main routine and then put a RET instruction in the main routine, a major fault occurs (type 4, code 31).

When enabled, the FOR instruction repeatedly executes the Routine until the Index value exceeds the Terminal value. This instruction does not pass parameters to the routine.

Each time the FOR instruction executes the routine, it adds the Step size to the Index.

Be careful not to loop too many times in a single scan. An excessive number of repetitions can cause the controller's watchdog to timeout, which causes a major fault.

**Arithmetic Status Flags:**    not affected

**Fault Conditions:**

| A Major Fault Will Occur If | Fault Type | Fault Code |
|---|---|---|
| main routine contains a RET instruction | 4 | 31 |

**Execution:**

| Condition | Relay Ladder Action |
|---|---|
| prescan | The rung-condition-out is set to false. |
| | The controller executes the subroutine once. |
| | If recursive FOR instruction0s exist to the same subroutine, the subroutine is prescanned only the first time. If multiple FOR instructions exist (non-recursive) to the same subroutine, the subroutine is prescanned each time. |
| rung-condition-in is false | The rung-condition-out is set to false. |

| Condition | Relay Ladder Action |
|---|---|
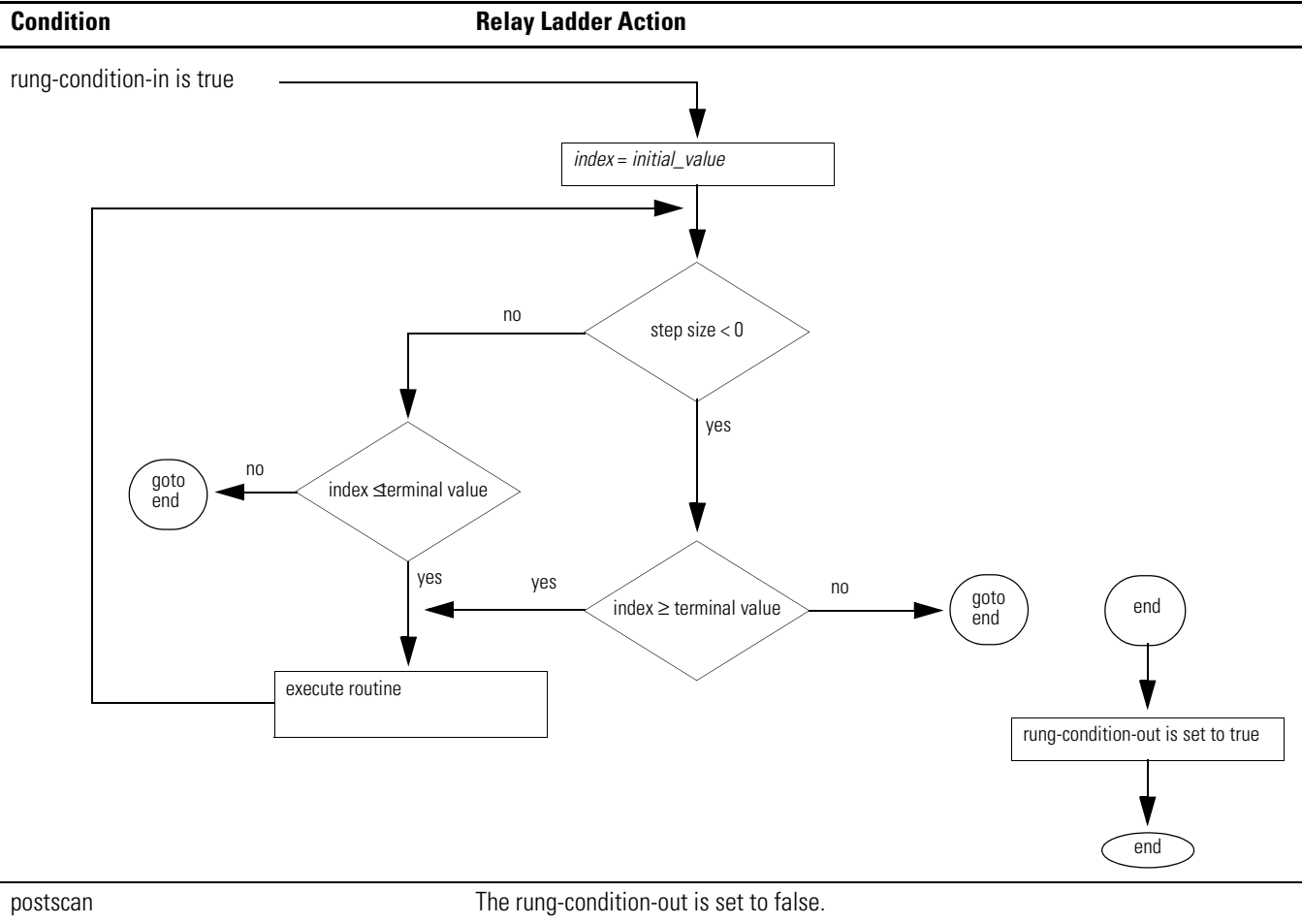
rung-condition-in is true

index = initial_value

step size < 0

no

yes

index ≤ terminal value

no → goto end

yes

execute routine

index ≥ terminal value

yes

no → goto end

end

rung-condition-out is set to true

end

| postscan | The rung-condition-out is set to false. |
|---|---|

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

```
-----FOR-----
For
Routine name    routine_2
Index           value_2
                  ?? ←
Initial value        0
Terminal value      10
Step size            1
```
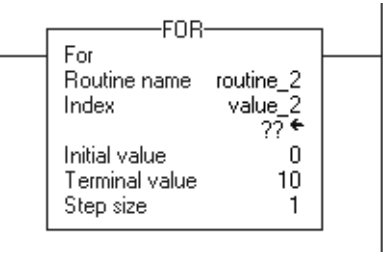
# Break (BRK)

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

## Operands:

📄  —(BRK)—
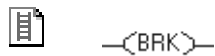
### Relay Ladder
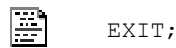
📄  EXIT;

### 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.

—(BRK)—
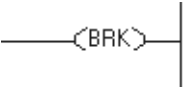
# Return (RET)

The RET instruction returns to the calling FOR instruction.

## Operands:
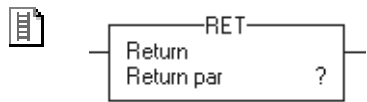


### Relay Ladder

## Description:

| **IMPORTANT** | Do not place a RET instruction in the main routine. If you place a RET instruction in the main routine, a major fault occurs (type 4, code 31). |

When enabled, the RET instruction returns to the FOR instruction. The FOR instruction increments the Index value by the Step size and executes the subroutine again. If the Index value exceeds the Terminal value, the FOR instruction completes and execution moves on to the instruction that follows the FOR instruction.

The FOR instruction does not use parameters. The FOR instruction ignores any parameters you enter in a RET instruction.

You could also use a TND instruction to end execution of a subroutine.

## Arithmetic Status Flags:

not affected

## Fault Conditions:

| A Major Fault Will Occur If | Fault Type | Fault Code |
|---|---|---|
| main routine contains a RET instruction | 4 | 31 |

## Execution:

| Condition: | Relay Ladder Action |
|---|---|
| prescan | The rung-condition-out is set to false. |
| rung-condition-in is false | The rung-condition-out is set to false. |
| rung-condition-in is true | Returns the specified parameters to the calling routine. The rung-condition-out is set to true. |
| postscan | The rung-condition-out is set to false. |

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

The RET instruction returns to the calling FOR instruction. The FOR instruction either executes the subroutine again and increments the Index value by the Step size or, if the Index value exceeds the Terminal value, the FOR instruction is complete and execution moves on to the instruction that follows the FOR instruction.

**calling routine**　　　　　　　　　　　　　　　　　　　　**subroutine**

```
              ┌──FOR──────────────────┐                              ┌──RET──────────┐
──────────────┤ For                   ├──────    ◄──────────────────┤ Return        ├──────
              │ Routine name  routine_2│                              └───────────────┘
              │ Index          value_2 │
              │                    ?? ← │
              │ Initial value        0  │
              │ Terminal value      10  │
              │ Step size            1  │
              └────────────────────────┘
```
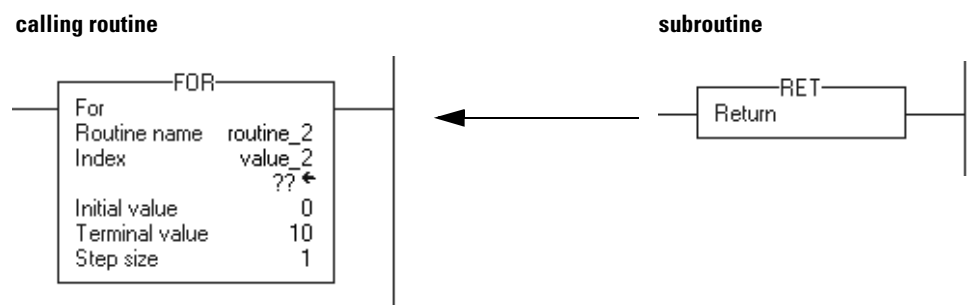
**Notes:**

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

**Introduction**

The special instructions perform application-specific operations.

| If You Want To | Use This Instruction | Available In These Languages | See Page |
|---|---|---|---|
| Compare data against a known, good reference and record any mismatches. | FBC | relay ladder | 478 |
| Compare data against a known, good reference, record any mismatches, and update the reference to match the source. | DDT | relay ladder | 486 |
| 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 | 494 |
| Control a PID loop. | PID | relay ladder | 497 |
| | | structured text | |

# File Bit Comparison (FBC)

The FBC instruction compares bits in a Source array with bits in a Reference array.

## Operands:
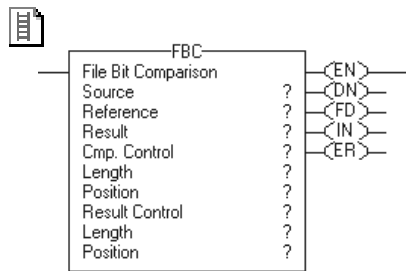


## Relay Ladder

| Operand | Type | Format | Description: |
|---------|------|--------|--------------|
| Source | DINT | array tag | array to compare to the reference |
|  |  |  | **do not** use CONTROL.POS in the subscript |
| Reference | DINT | array tag | array to compare to the source |
|  |  |  | **do not** use CONTROL.POS in the subscript |
| Result | DINT | array tag | array to store the result |
|  |  |  | **do not** use CONTROL.POS in the subscripts |
| Cmp control | CONTROL | structure | control structure for the compare |
| Length | DINT | immediate | number of bits to compare |
| Position | DINT | immediate | current position in the source |
|  |  |  | initial value is typically 0 |
| Result control | CONTROL | structure | control structure for the results |
| Length | DINT | immediate | number of storage locations in the result |
| Position | DINT | immediate | current position in the result |
|  |  |  | initial value is typically 0 |

**ATTENTION**

Use different tags for the compare control structure and the result control structure. Using the same tag for both could result in unpredictable operation, possibly causing equipment damage and/or injury to personnel.

## COMPARE Structure

| Mnemonic: | Data Type | Description: |
|---|---|---|
| .EN | BOOL | The enable bit indicates that the FBC instruction is enabled. |
| .DN | BOOL | The done bit is set when the FBC instruction compares the last bit in the Source and Reference arrays. |
| .FD | BOOL | The found bit is set each time the FBC instruction records a mismatch (one-at-a-time operation) or after recording all mismatches (all-per-scan operation). |
| .IN | BOOL | The inhibit bit indicates the FBC search mode.<br><br>0 = all mode<br><br>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.<br><br>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.<br><br>Each time the rung-condition-in goes from false to true, the FSC instruction searches for all mismatches between the Source and Reference arrays. |

**Arithmetic Status Flags:**    not affected

**Fault Conditions:**

| A Major Fault Will Occur If: | Fault Type | Fault Code |
| --- | --- | --- |
| Result.POS > size of Result array | 4 | 20 |

**Execution:**

| Condition | Relay Ladder Action |
|---|---|

prescan

compare.EN bit is cleared

examine compare.DN bit

compare.DN = 0

compare.DN = 1

compare.DN bit is cleared

compare.POS value is cleared

result.DN bit is cleared

rung-condition-out is set to **false**

end

| Condition | Relay Ladder Action |
|---|---|

rung-condition-in is false

compare.EN bit is cleared

examine
compare.DN bit

compare.DN = 0

compare.DN = 1

compare DN bit is cleared

compare.POS value is cleared

result.DN bit is cleared

rung-condition-out is set to
**false**

end

| Condition | Relay Ladder Action |
|---|---|

rung-condition-in is true

examine compare.EN bit

compare.EN = 1 → goto exit

compare.EN = 0

compare EN bit is set →

examine compare.DN bit

compare.DN = 1 → goto exit

compare.DN = 0

compare.ER bit is cleared →

compare.LEN ≤0

yes

no

compare.POS < 0

yes

no

compare.ER bit is set

compare

page 484

goto exit

**exit**

rung-condition-out is set to true

end

| Condition | Relay Ladder Action |
|---|---|

| postscan | The rung-condition-out is set to false. |
|---|---|

**Example:**    When enabled, the FBC instruction compares the source *array_dint1* to the reference *array_dint2* and stores the locations of any mismatches in the result *array_dint3*.

```
                    ┌─────────FBC──────────────┐
                    │ File Bit Comparison       │──(EN)──
                    │ Source      array_dint1[0]│──(DN)──
                    │ Reference   array_dint2[0]│──(FD)──
                    │ Result      array_dint3[0]│──(IN )──
                    │ Cmp. Control      control_1│──(ER)──
                    │ Length              10 ←  │
                    │ Position             0 ←  │
                    │ Result Control    control_3│
                    │ Length              10 ←  │
                    │ Position             0 ←  │
                    └───────────────────────────┘
```

source
*array_dint1*

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

reference
*array_dint2*

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |

result
*array_dint3*

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | 5 | 3 |

# Diagnostic Detect (DDT)

The DDT instruction compares bits in a Source array with bits in a Reference array to determine changes of state.

## Operands:
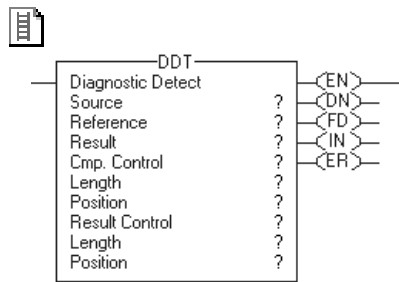
### Relay Ladder

| Operand | Type | Format | Description |
|---|---|---|---|
| Source | DINT | array tag | array to compare to the reference |
| | | | **do not** use CONTROL.POS in the subscript |
| Reference | DINT | array tag | array to compare to the source |
| | | | **do not** use CONTROL.POS in the subscript |
| Result | DINT | array tag | array to store the results |
| | | | **do not** use CONTROL.POS in the subscript |
| Cmp control | CONTROL | structure | control structure for the compare |
| Length | DINT | immediate | number of bits to compare |
| Position | DINT | immediate | current position in the source |
| | | | initial value typically 0 |
| Result control | CONTROL | structure | control structure for the results |
| Length | DINT | immediate | number of storage locations in the result |
| Position | DINT | immediate | current position in the result |
| | | | initial value typically 0 |

**ATTENTION**

Use different tags for the compare control structure and the result control structure. Using the same tag for both could result in unpredictable operation, possibly causing equipment damage and/or injury to personnel.

## COMPARE Structure

| Mnemonic | Data Type | Description |
|---|---|---|
| .EN | BOOL | The enable bit indicates that the DDT instruction is enabled. |
| .DN | BOOL | The done bit is set when the DDT instruction compares the last bit in the Source and Reference arrays. |
| .FD | BOOL | The found bit is set each time the DDT instruction records a mismatch (one-at-a-time operation) or after recording all mismatches (all-per-scan operation). |
| .IN | BOOL | The inhibit bit indicates the DDT search mode. <br><br> 0 = all mode <br><br> 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 |

**Execution:**

| Condition: | Relay Ladder Action |
|---|---|

prescan

compare.EN bit is cleared

examine
compare.DN bit

compare.DN = 0

compare.DN = 1

compare.DN bit is cleared

compare.POS value is cleared

result.DN bit is cleared

rung-condition-out is set to
**false**

end

| Condition: | Relay Ladder Action |
|---|---|

rung-condition-in is false

```
                          │
                          ▼
         ┌────────────────────────────────┐
         │ compare.EN bit is cleared       │
         └────────────────────────────────┘
                          │
                          ▼
                    ╱─────────╲            compare.DN = 0
                   ╱  examine   ╲ ──────────────────────────┐
                   ╲ compare.DN  ╱                          │
                    ╲   bit    ╱                            │
                     ╲────────╱                             │
                          │ compare.DN = 1                  │
                          ▼                                 │
         ┌────────────────────────────────┐                │
         │ compare DN bit is cleared        │               │
         │                                  │               │
         │ compare.POS value is cleared     │               │
         │                                  │               │
         │ result.DN bit is cleared         │               │
         └────────────────────────────────┘                │
                          │                                 │
                          ◄─────────────────────────────────┘
                          │
                          ▼
         ┌────────────────────────────────┐
         │ rung-condition-out is set to     │
         │ **false**                        │
         └────────────────────────────────┘
                          │
                          ▼
                      ( end )
```

| Condition: | Relay Ladder Action |
|---|---|

rung-condition-in is true

examine compare.EN bit

compare.EN = 1 → goto exit

compare.EN = 0

compare EN bit is set →

examine compare.DN bit

compare.DN bit = 1 → goto exit

compare.DN bit = 0

compare.ER bit is cleared
compare.FD bit is cleared →

compare.LEN ≤0

yes

no

compare.POS < 0

yes

no

compare

page 492

compare.ER bit is set

goto exit

**exit**

rung-condition-out is set to true

end

| Condition: | Relay Ladder Action |
|---|---|

compare

compare.POS ≥ compare.LEN — yes → compare.POS = compare.LEN → goto exit

no

source[compare.POS] = reference[compare.POS] — no → compare.FD bit is set reference[compare.POS] = source[compare.POS] → examine result.DN bit

result.DN = 1 → result.DN bit is cleared result.POS value is cleared

result.DN = 0

yes

compare.POS = compare.POS + 1

result.POS < 0 — yes

no

result.LEN ≤ 0 — yes

no

compare.ER bit is set

result.POS > size of result array — yes → major fault

no

goto exit

result[result.POS] = compare.POS
result.POS = result.POS + 1

result.POS ≥ result.LEN

no

yes

result.DN bit is set

| 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*.

```
            ┌──────DDT──────────────────┐
            │ Diagnostic Detect         │──(EN)──
   ─────────│ Source      array_dint1[0]│──(DN)──
            │ Reference   array_dint2[0]│──(FD)──
            │ Result      array_dint3[0]│──(IN)──
            │ Cmp. Control     control_1│──(ER)──
            │ Length              10 ◄  │
            │ Position             0 ◄  │
            │ Result Control   control_3│
            │ Length              10 ◄  │
            │ Position             0 ◄  │
            └───────────────────────────┘
```

source
array_dint1

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

reference (before compare)
array_dint2

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0

result
array_dint3

  5 3

reference (after compare)
array_dint2

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

# Data Transitional (DTR)

The DTR instruction passes the Source value through a Mask and compares the result with the Reference value.

## Operands:
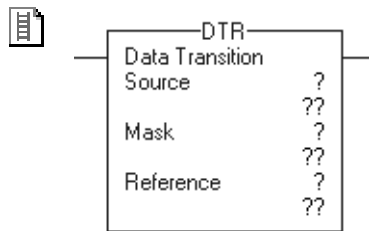


### Relay Ladder

| Operand: | Type | Format | Description |
|---|---|---|---|
| Source | DINT | immediate<br><br>tag | array to compare to the reference |
| Mask | DINT | immediate<br><br>tag | which bits to block or pass |
| Reference | DINT | tag | array to compare to the source |

**Description:** The DTR instruction passes the Source value through a Mask and compares the result with the Reference value. The DTR instruction also writes the masked Source value into the Reference value for the next comparison. The Source remains unchanged.

A "1" in the mask means the data bit is passed. A "0" in the mask means the data bit is blocked.

When the masked Source differs from the Reference, the rung-condition-out goes true for one scan. When the masked Source is the same as the Reference, the rung-condition-out is false.

---

**ATTENTION**

⚠

Online programming with this instruction can be dangerous. If the Reference value is different than the Source value, the rung-condition-out goes true. Use caution if you insert this instruction when the processor is in Run or Remote Run mode.

---

## Enter an immediate mask value

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

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

**Arithmetic Status Flags:**  not affected

**Fault Conditions:**  none

**Execution:**

| Condition | Relay Ladder Action |
|-----------|---------------------|
| prescan | The Reference = Source AND Mask. |
| | The rung-condition-out is set to false. |
| rung-condition-in is false | The Reference = Source AND Mask. |
| | The rung-condition-out is set to false. |

rung-condition-in is true



| | |
|---|---|
| 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.

```
          ┌──────DTR──────┐
          │ Data Transition│
          │ Source   value_1│
          │            0 ←  │
          │ Mask     mask_1 │
          │            0 ←  │
          │ Reference value_2│
          │            0 ←  │
          └────────────────┘
```

**example 1**

| | | | | source |
|---|---|---|---|---|
| 7 | 1 | 8 | 3 | |

mask = 0FFF

| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

reference

| | | | | |
|---|---|---|---|---|
| current scan | 0 | 1 | 8 | 3 |
| previous scan | 0 | 1 | 8 | 3 |

The rung remains false as long as the input value does not change.

**example 2**

| | | | |
|---|---|---|---|
| 9 | 1 | 8 | 7 |

| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| 0 | 1 | 8 | 7 | current scan |
| 0 | 1 | 8 | 3 | previous scan |

The rung remains true for one scan when a change is detected.

13385

A 0 in the mask leaves the bit unchanged.

# Proportional Integral Derivative (PID)

The PID instruction controls a process variable such as flow, pressure, temperature, or level.

### Operands:

### Relay Ladder



| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| PID | PID | structure | PID structure |
| Process variable | SINT INT DINT REAL | tag | value you want to control |
| Tieback | SINT INT DINT REAL | immediate tag | *(optional)* output of a hardware hand/auto station which is bypassing the output of the controller<br><br>Enter 0 if you don't want to use this parameter. |
| Control variable | SINT INT DINT REAL | tag | value which goes to the final control device (valve, damper, etc.)<br><br>If you are using the deadband, the Control variable must be REAL or it will be forced to 0 when the error is within the deadband. |
| PID master loop | PID | structure | (*optional*) PID tag for the master PID<br><br>If you are performing cascade control and this PID is a slave loop, enter the name of the master PID. Enter 0 if you don't want to use this parameter. |
| Inhold bit | BOOL | tag | (*optional*) current status of the inhold bit from a 1756 analog output channel to support bumpless restart<br><br>Enter 0 if you don't want to use this parameter. |
| Inhold value | SINT INT DINT REAL | tag | (*optional*) data readback value from a 1756 analog output channel to support bumpless restart<br><br>Enter 0 if you don't want to use this parameter. |

| 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.

### PID Structure

| Mnemonic: | Data Type | Description |
|---|---|---|
| .CTL | DINT | The .CTL member provides access to the status members (bits) in one, 32-bit word. The PID instruction sets bits 07 -15. |

| This Bit | Is This Member |
|---|---|
| 31 | .EN |
| 30 | .CT |
| 29 | .CL |
| 28 | .PVT |
| 27 | .DOE |
| 26 | .SWM |
| 25 | .CA |
| 24 | .MO |
| 23 | .PE |
| 22 | .NDF |
| 21 | .NOBC |
| 20 | .NOZC |

| This Bit: | Is This Member, Which the PID Instruction Sets |
|---|---|
| 15 | .INI |
| 14 | .SPOR |
| 13 | .OLL |
| 12 | .OLH |
| 11 | .EWD |
| 10 | .DVNA |
| 09 | .DVPA |
| 08 | .PVLA |
| 07 | .PVHA |

| Mnemonic: | Data Type | Description | |
|-----------|-----------|-------------|--|
| .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 | |
| .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<br>(0=derivative smoothing filter enabled; 1=derivative smoothing filter disabled) | | |
| .NOBC | BOOL | no bias back calculation<br>(0=bias back calculation enabled; 1=bias back calculation disabled) | | |
| .NOZC | BOOL | no zero crossing deadband<br>(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.

.EN bit

rung state

execution of the PID instruction
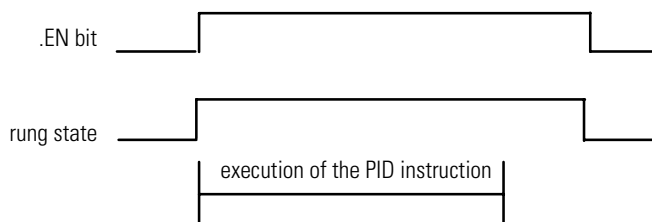
**Arithmetic Status Flags:**   not affected

**Fault Conditions:**

| IMPORTANT | These faults were major faults in the PLC-5 controller. |
|---|---|

| A Minor Fault Will Occur If | Fault Type | Fault Code |
|---|---|---|
| .UPD ≤0 | 4 | 35 |
| setpoint out of range | 4 | 36 |

**Execution:**

| Condition | Action | Action |
|---|---|---|
| prescan | The rung-condition-out is set to false. | No action taken. |
| rung-condition-in is false | The rung-condition-out is set to false. | na |
| rung-condition-in is true | The instruction executes.<br><br>The rung-condition-out is set to true. | na |
| EnableIn is set | na | EnableIn is always set.<br><br>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.

```
            ┌─────────PID─────────────┐
            │ Proportional Integral Derivative │
            │ PID                  ?  [...]│ ◄─── Click here to configure the
            │ Process Variable     ?      │      PID instruction
            │ Tieback              ?      │
            │ Control Variable     ?      │
            │ PID Master Loop      ?      │
            │ Inhold Bit           ?      │
            │ Inhold Value         ?      │
            │ Setpoint            ??      │
            │ Process Variable    ??      │
            │ Output %            ??      │
            └─────────────────────────┘
```
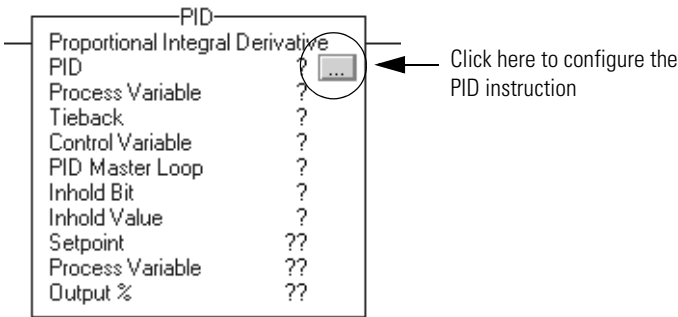
## Specify Tuning

Select the Tuning tab. Changes take affect as soon as you click on another field, click OK, click Apply, or press Enter.

| In This Field | Specify: |
|---|---|
| Setpoint (SP) | Enter a setpoint value (.SP). |
| Set output % | Enter a set output percentage (.SO). |
|  | In software manual mode, this value is used for the output. |
|  | In auto mode, this value displays the output %. |
| Output bias | Enter an output bias percentage (.BIAS). |
| Proportional gain ($K_p$) | Enter the proportional gain (.KP). |
|  | For independent gains, it's the proportional gain (unitless). |
|  | For dependent gains, it's the controller gain (unitless). |
| Integral gain ($K_i$) | Enter the integral gain (.KI). |
|  | For independent gains, it's the integral gain (1/sec). |
|  | For dependent gains, it's the reset time (minutes per repeat). |
| Derivative time ($K_d$) | Enter the derivative gain (.KD). |
|  | For independent gains, it's the derivative gain (seconds). |
|  | For dependent gains, it's the rate time minutes). |
| Manual mode | Select either manual (.MO) or software manual (.SWM). |
|  | Manual mode overrides software manual mode if both are selected. |

## Specify Configuration

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

| In this field | Specify |
|---|---|
| PID equation | Select independent gains or dependent gains (.PE). |
| | Use independent when you want the three gains (P, I, and D) to operate independently. Use dependent when you want an overall controller gain that affects all three terms (P, I, and D). |
| Control action | Select either E=PV-SP or E=SP-PV for the control action (.CA). |
| Derivative of | Select PV or error (.DOE). |
| | Use the derivative of PV to eliminate output spikes resulting from setpoint changes. Use the derivative of error for fast responses to setpoint changes when the algorithm can tolerate overshoots. |
| Loop update time | Enter the update time (.UPD) for the instruction. |
| CV high limit | Enter a high limit for the control variable (.MAXO). |
| 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:
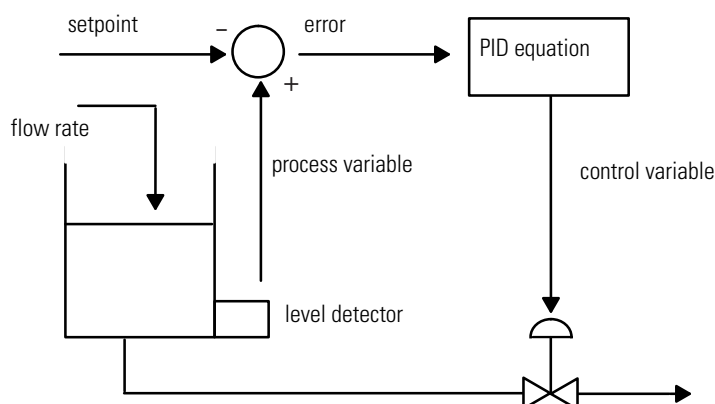


In the above example, the level in the tank is compared against the setpoint. If the level is higher than the setpoint, the PID equation increases the control variable and causes the outlet valve from the tank to open; thereby decreasing the level in the tank.

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

Where:

| Variable | Description |
|----------|-------------|
| $K_P$ | proportional gain (unitless) |
|  | $K_p = K_c$ unitless |
| $K_i$ | integral gain (seconds $^{-1}$) |
|  | To convert between $K_i$ (integral gain) and $T_i$ (reset time), use: |
|  | $$K_i = \frac{K_C}{60\,T_i}$$ |
| $K_d$ | derivative gain (seconds) |
|  | To convert between $K_d$ (derivative gain) and $T_d$ (rate time), use: |
|  | $K_d = K_c\,(T_d)\,60$ |
| $K_C$ | controller gain (unitless) |
| $T_i$ | reset time (minutes/repeat) |
| $T_d$ | rate time (minutes) |
| SP | setpoint |
| PV | process variable |
| E | error [(SP-PV) or (PV-SP)] |
| BIAS | feedforward or bias |
| CV | control variable |
| dt | loop update time |

If you do not want to use a particular term of the PID equation, just set its gain to zero. For example if you want no derivative action, set $K_d$ or $T_d$ equal to zero.

## Anti-reset Windup And Bumpless Transfer From Manual To Auto

The PID instruction automatically avoids reset windup by preventing the integral term from accumulating whenever the CV output reaches its maximum or minimum values, as set by .MAXO and .MINO. The accumulated integral term remains frozen until the CV output drops below its maximum limit or rises above its minimum limit. Then normal integral accumulation automatically resumes.

The PID instruction supports two manual modes of control:

| Manual Mode of Control | Description |
| --- | --- |
| software manual (.SWM) | also known as set output mode |
| | lets the user set the output % from the software |
| | The set output (.SO) value is used as the output of the loop. The set output value typically comes from an operator input from an operator interface device. |
| manual (.MO) | takes the tieback value, as an input, and adjusts its internal variables to generate the same value at the output |
| | The tieback input to the PID instruction is scaled to 0-100% according to the values of .MINTIE and .MAXTIE and is used as the output of the loop. The tieback input typically comes from the output of a hardware hand/auto station which is bypassing the output from the controller. |
| | **Note:** Manual mode overrides software manual mode if both mode bits are set on. |

The PID instruction also automatically provides bumpless transfers from software manual mode to auto mode or from manual to auto mode. The PID instruction back-calculates the value of the integral accumulation term required to make the CV output track either the set output (.SO) value in software manual mode or the tieback input in manual mode. In this manner, when the loop switches to auto mode, the CV output starts off from the set output or tieback value and no "bump" in output value occurs.

The PID instruction can also automatically provide a bumpless transfer from manual to auto even if integral control is not used (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
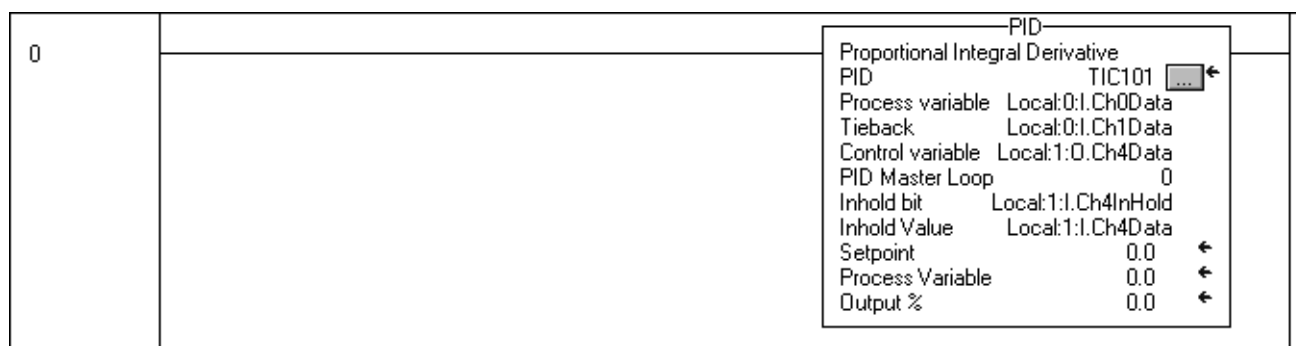


### Structured Text

```
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**
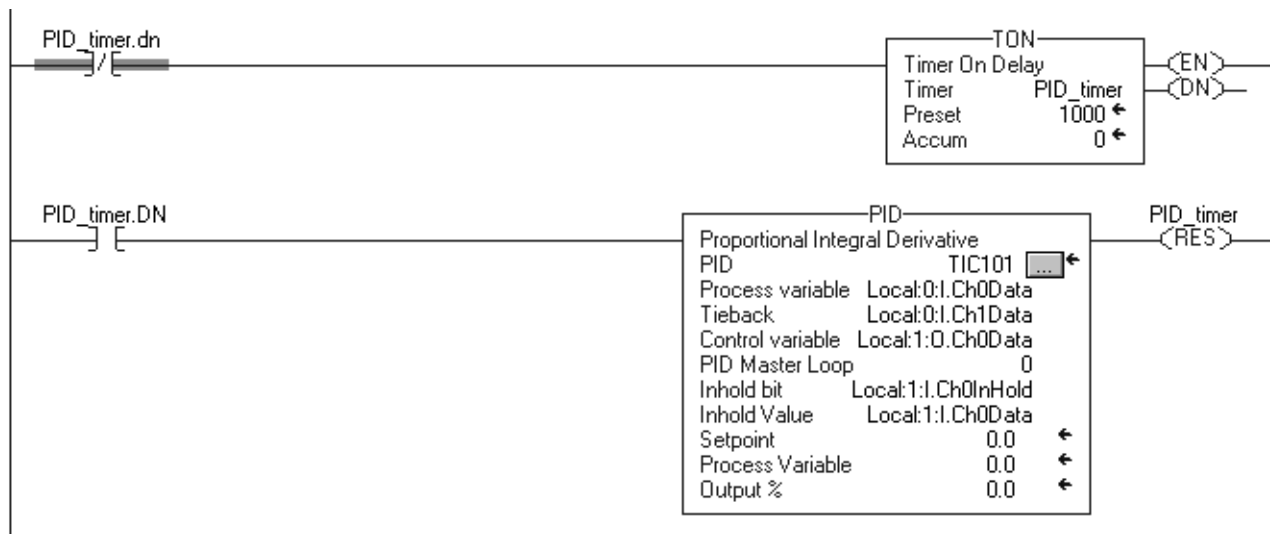


**Structured Text**

```
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
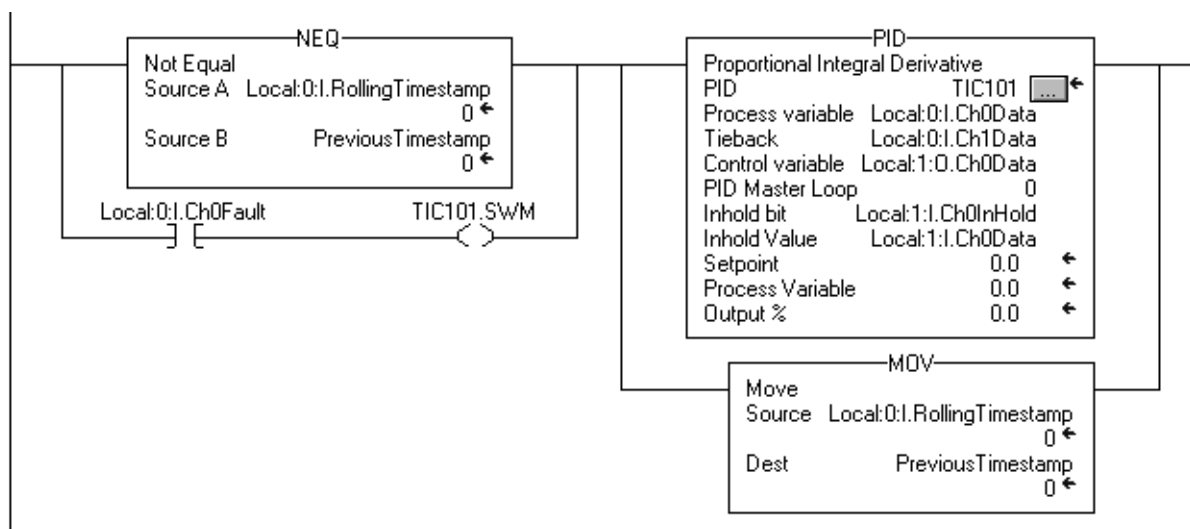
### Structured Text

```
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:O.Ch0Data,0,Local:1:I.Ch0InHold,
    Local:1:I.Ch0Data);

END_IF;
```

## Bumpless Restart

The PID instruction can interact with the 1756 analog output modules to support a bumpless restart when the controller changes from Program to Run mode or when the controller powers up.

When a 1756 analog output module loses communications with the controller or senses that the controller is in Program mode, the analog output module sets its outputs to the fault condition values you specified when you configured the module. When the controller then returns to Run mode or re-establishes communications with the analog output module, you can have the PID instruction automatically reset its control variable output equal to the analog output by using the Inhold bit and Inhold Value parameters on the PID instruction.

To set a bumpless restart:

| Do This | Details: |
| --- | --- |
| Configure the 1756 analog output module's channel which receives the control variable from the PID instruction | Select the "hold for initialization" check box on the properties page for the specific channel of the module. |
| | This tells the analog output module that when the controller returns to Run mode or re-establishes communications with the module, the module should hold the analog output at its current value until the value sent from the controller matches (within 0.1% of span) the current value used by the output channel. The controller's output will ramp to the currently held output value by making use of the .BIAS term. This ramping is similar to auto bumpless transfer. |
| Enter the Inhold bit tag and Inhold Value tag in the PID instruction | The 1756 analog output module returns two values for each channel in its input data structure. The InHold status bit (.Ch2InHold, for example), when true, indicates that the analog output channel is holding its value. The Data readback value (.Ch2Data, for example) shows the current output value in engineering units. |
| | Enter the tag of the InHold status bit as the InHold bit parameter of the PID instruction. Enter the tag of the Data readback value as the Inhold Value parameter. |
| | When he Inhold bit goes true, the PID instruction moves the Inhold Value into the Control variable output and re-initializes to support a bumpless restart at that value. When the analog output module receives this value back from the controller, it turns off the InHold status bit, which allows the PID instruction to start controlling normally. |

## Derivative Smoothing

The derivative calculation is enhanced by a derivative smoothing filter. This first order, low pass, digital filter helps to minimize large derivative term spikes caused by noise in the PV. This smoothing becomes more aggressive with larger values of derivative gain. You can disable derivative smoothing if your process requires very large values of derivative gain ($K_d > 10$, for example). To disable derivative smoothing, select the "No derivative smoothing" option on the Configuration tab or set the .NDF bit in the PID structure.

## Set the Deadband

The adjustable deadband lets you select an error range above and below the setpoint where output does not change as long as the error remains within this range. This deadband lets you control how closely the process variable matches the setpoint without changing the output. The deadband also helps to minimize wear and tear on your final control device.
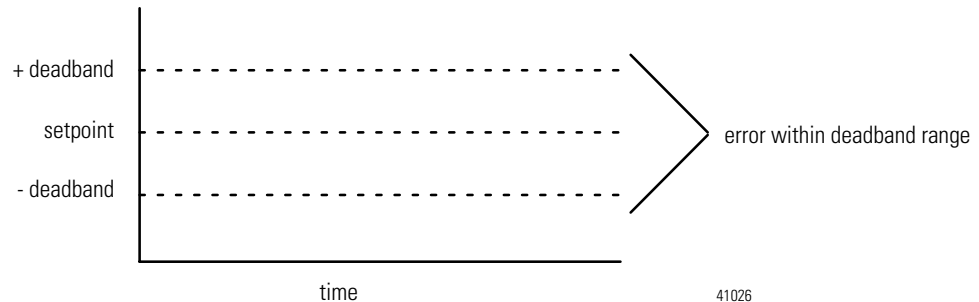


Zero-crossing is deadband control that lets the instruction use the error for computational purposes as the process variable crosses into the deadband until the process variable crosses the setpoint. Once the process variable crosses the setpoint (error crosses zero and changes sign) and as long as the process variable remains in the deadband, the output will not change.

The deadband extends above and below the setpoint by the value you specify. Enter zero to inhibit the deadband. The deadband has the same scaled units as the setpoint. You can use the deadband without the zero-crossing feature by selecting the "no zero crossing for deadband" option on the Configuration tab or set the .NOZC bit in the PID structure.

If you are using the deadband, the Control variable must be REAL or it will be forced to 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
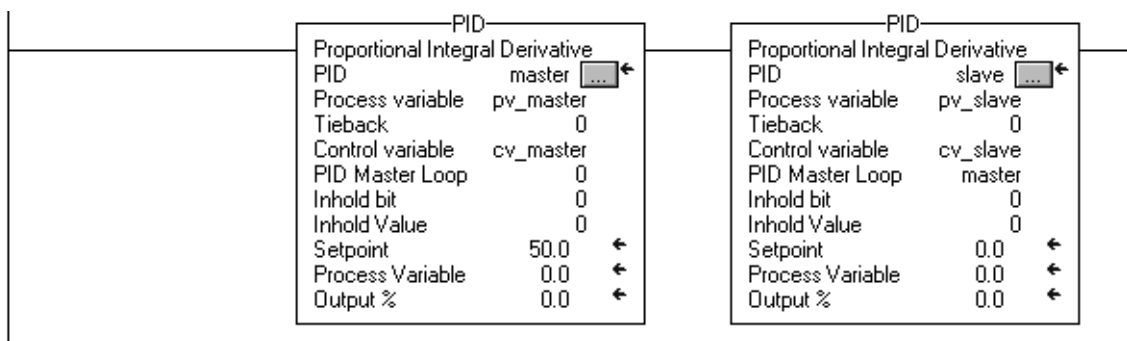


### Structured Text

```
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
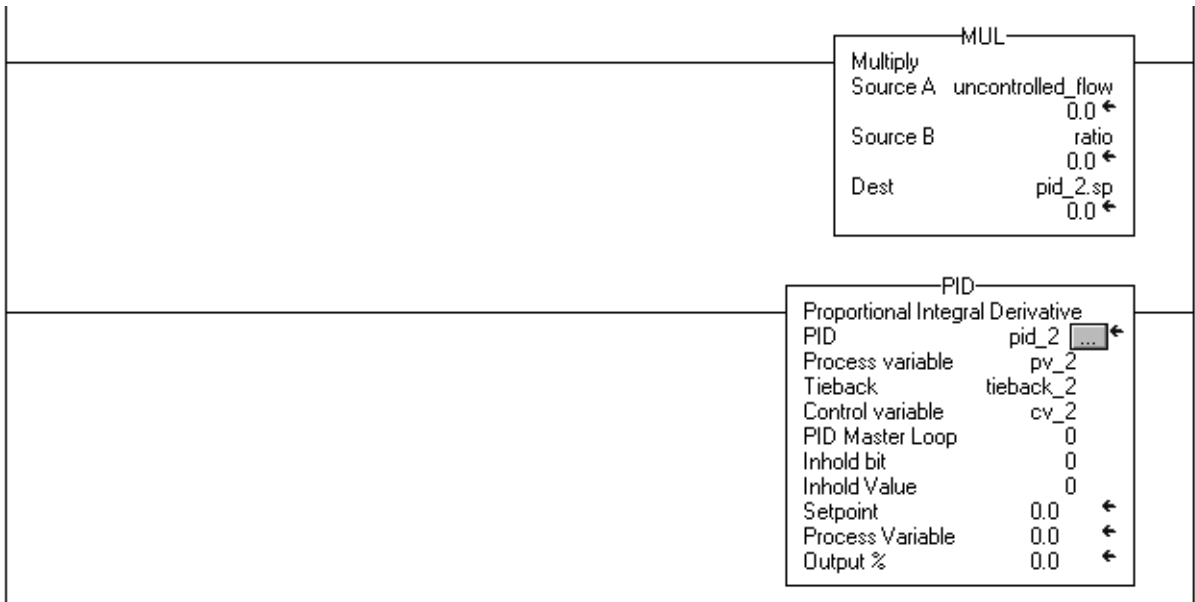


### Structured Text

```
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
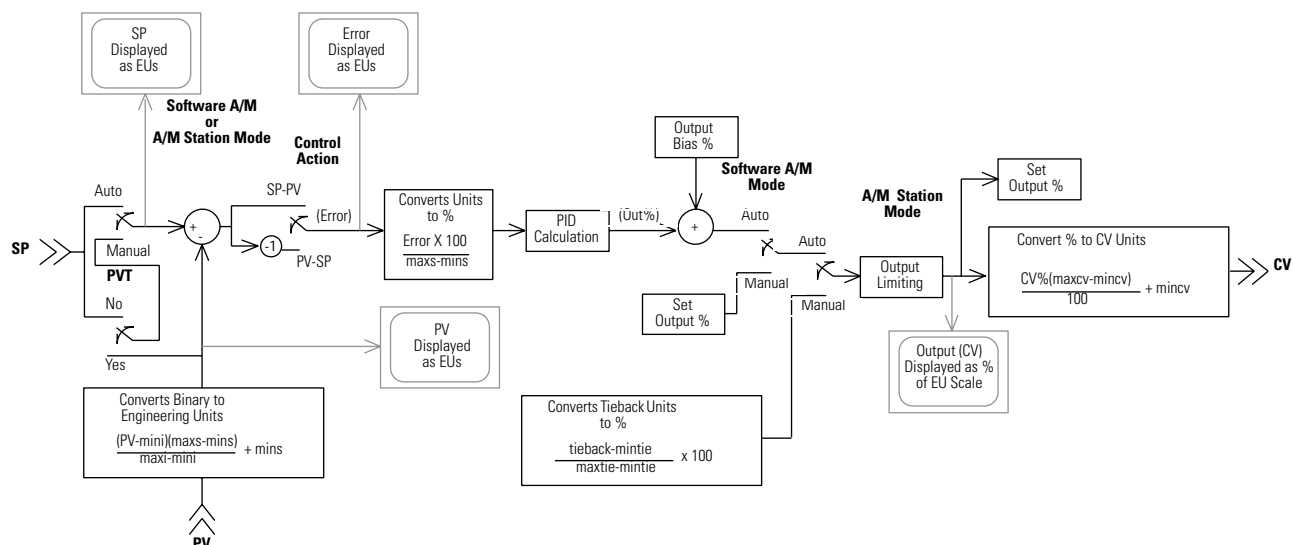


## PID Process With Master/slave Loops
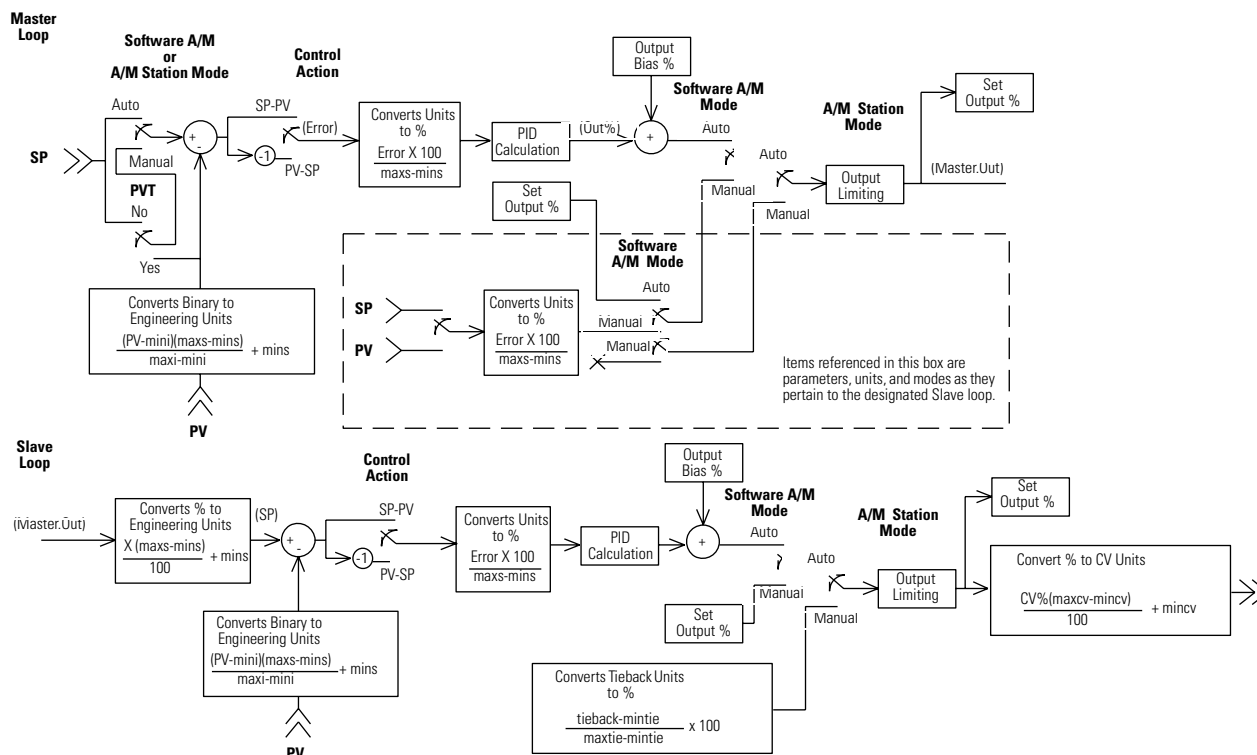
**Notes:**

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

**Introduction**

The trigonometric instructions evaluate arithmetic operations using trigonometric operations.

| If You Want To | Use This Instruction | Available In These Languages | See Page |
|---|---|---|---|
| Take the sine of a value. | SIN | relay ladder structured text function block | 520 |
| Take the cosine of a value. | COS | relay ladder structured text function block | 523 |
| Take the tangent of a value. | TAN | relay ladder structured text function block | 526 |
| Take the arc sine of a value. | ASN ASIN[1] | relay ladder structured text function block | 529 |
| Take the arc cosine of a value. | ACS ACOS[1] | relay ladder structured text function block | 532 |
| Take the arc tangent of a value. | ATN ATAN[1] | relay ladder structured text function block | 535 |

[1] Structured text only.

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

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

# Sine (SIN)

The SIN instruction takes the sine of the Source value (in radians) and stores the result in the Destination.
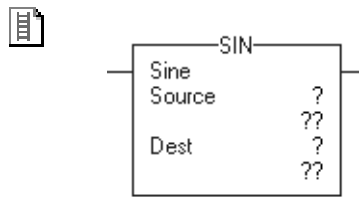
### Operands:

### Relay Ladder

| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| Source | SINT | immediate | find the sine of this value |
| | INT | tag | |
| | DINT | | |
| | REAL | | |
| Destination | SINT | tag | tag to store the result |
| | INT | | |
| | DINT | | |
| | REAL | | |

### Structured Text

`dest := SIN(source);`

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

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

### Function Block
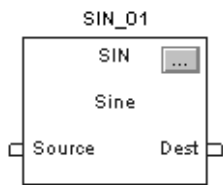
| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| SIN tag | FBD_MATH_ADVANCED | structure | SIN structure |

### FBD_MATH_ADVANCED Structure

| Input Parameter | Data Type | Description |
|-----------------|-----------|-------------|
| EnableIn | BOOL | Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set. |
| Source | REAL | Input to the math instruction. Valid = any float |

| Input Parameter | Data Type | Description |
|---|---|---|
| **Output Parameter** | **Data Type** | **Description** |
| EnableOut | BOOL | The instruction produced a valid result. |
| Dest | REAL | Result of the math instruction. Arithmetic status flags are set for this output. |

**Description:** The Source must be greater than or equal to -205887.4 ($-2\pi x2^{15}$) and less than or equal to 205887.4 ($2\pi x2^{15}$). The resulting value in the Destination is always greater than or equal to -1 and less than or equal to 1.

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

**Fault Conditions:** none

**Execution:**



## Relay Ladder

| Condition: | Action: |
|---|---|
| prescan | The rung-condition-out is set to false. |
| rung-condition-in is false | The rung-condition-out is set to false. |
| rung-condition-in is true | The controller calculates the sine of the Source and places the result in the Destination. |
| | The rung-condition-out is set to true. |
| postscan | The rung-condition-out is set to false. |



## Function Block

| Condition | Action |
|---|---|
| prescan | No action taken. |
| instruction first scan | No action taken. |
| 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. |

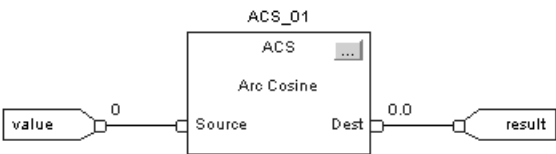**Example:** Calculate the sine of *value* and place the result in *result*.

### Relay Ladder

```
            ┌─SIN──────────────┐
            │ Sine             │
            │ Source     value │
            │      0.78539819 ←│
            │ Dest      result │
            │      0.70710677 ←│
            └──────────────────┘
```
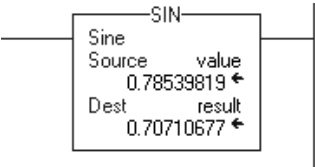
### Structured Text

```
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.
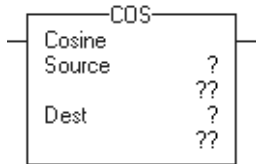
## Operands:



### 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** | | |

### Structured Text

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

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

See Appendix C, Structured Text Programming for information on the syntax of expressions within structured text.
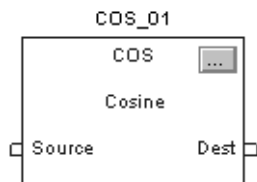
### Function Block



| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| COS tag | FBD_MATH_ADVANCED | structure | COS structure |

## **FBD_MATH_ADVANCED Structure**

| Input Parameter | Data Type | Description |
|---|---|---|
| EnableIn | BOOL | Enable input. If cleared, the instruction does not execute and outputs are not updated. |
| | | Default is set. |
| Source | REAL | Input to the math instruction. |
| | | Valid = any float |

| Output Parameter | Data Type | Description |
|---|---|---|
| EnableOut | BOOL | The instruction produced a valid result. |
| Dest | REAL | Result of the math instruction. Arithmetic status flags are set for this output. |

**Description:** The Source must be greater than or equal to -205887.4 ($-2\pi x2^{15}$) and less than or equal to 205887.4 ($2\pi x2^{15}$). The resulting value in the Destination is always greater than or equal to -1 and less than or equal to 1.

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

**Fault Conditions:** none

**Execution:**

📄 **Relay Ladder**

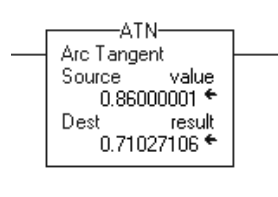| Condition | Action |
|---|---|
| prescan | The rung-condition-out is set to false. |
| rung-condition-in is false | The rung-condition-out is set to false. |
| rung-condition-in is true | The controller calculates the cosine of the Source and places the result in the Destination. |
| | The rung-condition-out is set to true. |
| postscan | The rung-condition-out is set to false. |

**Function Block**

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

**Example:**    Calculate the cosine of *value* and place the result in *result*.
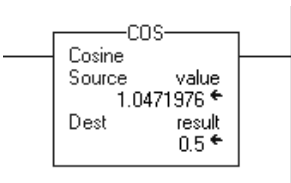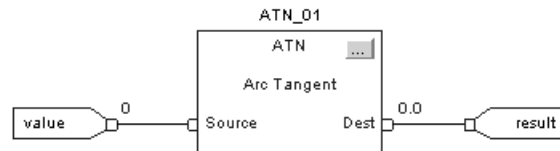
### Relay Ladder



### Structured Text

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

### Function Block
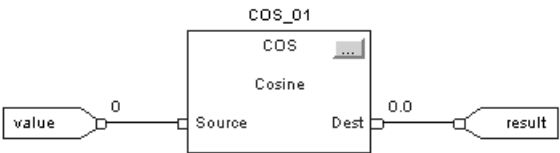
# Tangent (TAN)

The TAN instruction takes the tangent of the Source value (in radians) and stores the result in the Destination.
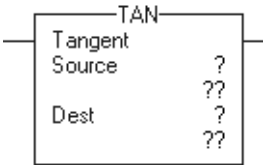
## Operands:

### Relay Ladder

| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| Source | SINT | immediate | find the tangent of this value |
| | INT | tag | |
| | DINT | | |
| | **REAL** | | |
| Destination | SINT | tag | tag to store the result |
| | INT | | |
| | DINT | | |
| | **REAL** | | |

### Structured Text

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

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

See Appendix C, Structured Text Programming for information on the syntax of expressions within structured text.
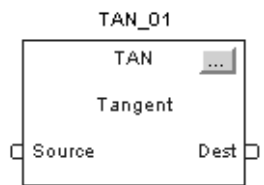
### Function Block

| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| TAN tag | FBD_MATH_ADVANCED | structure | TAN structure |

## FBD_MATH_ADVANCED Structure

| Input Parameter | Data Type | Description |
|---|---|---|
| EnableIn | BOOL | Enable input. If cleared, the instruction does not execute and outputs are not updated. |
| | | Default is set. |
| Source | REAL | Input to the math instruction. |
| | | Valid = any float |

| Output Parameter | Data Type | Description |
|---|---|---|
| EnableOut | BOOL | The instruction produced a valid result. |
| Dest | REAL | Result of the math instruction. Arithmetic status flags are set for this output. |

**Description:** The Source must be greater than or equal to -102943.7($-2\pi x2^{14}$) and less than or equal to 102943.7 ($2\pi x2^{14}$).

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

**Fault Conditions:** none

**Execution:**

### Relay Ladder

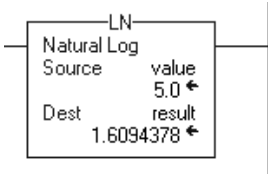| Condition | Action |
|---|---|
| prescan | The rung-condition-out is set to false. |
| rung-condition-in is false | The rung-condition-out is set to false. |
| rung-condition-in is true | The controller calculates the tangent of the Source and places the result in the Destination. |
| | The rung-condition-out is set to true. |
| postscan | The rung-condition-out is set to false. |

### Function Block

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

**Example:** Calculate the tangent of *value* and place the result in *result*.

### Relay Ladder

```
              ┌──────TAN──────┐
              │ Tangent       │
              │ Source  value │
              │       2.0943952 ←
              │ Dest    result │
              │      -1.7320507 ←
              └───────────────┘
```
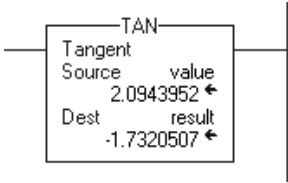
### Structured Text

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

### Function Block
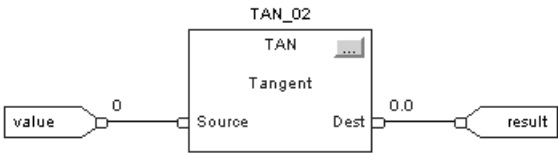
# Arc Sine (ASN)

The ASN instruction takes the arc sine of the Source value and stores the result in the Destination (in radians).
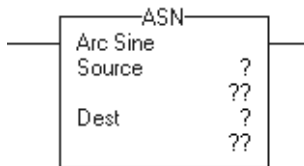
## Operands:



### Relay Ladder

| Operand | Type | Format | Description |
|---|---|---|---|
| Source | SINT | immediate | find the arc sine of this value |
| | INT | tag | |
| | DINT | | |
| | REAL | | |
| Destination | SINT | tag | tag to store the result |
| | INT | | |
| | DINT | | |
| | REAL | | |

### Structured Text

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

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

See [Appendix C, Structured Text Programming](#) for information on the syntax of expressions within structured text.
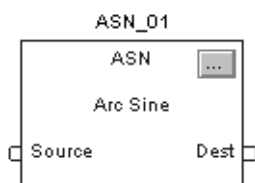
### Function Block



| Operand | Type | Format | Description |
|---|---|---|---|
| ASN tag | FBD_MATH_ADVANCED | structure | ASN structure |

### FBD_MATH_ADVANCED Structure

| Input Parameter | Data Type | Description |
|---|---|---|
| EnableIn | BOOL | Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set. |
| Source | REAL | Input to the math instruction. Valid = any float |

| Input Parameter | Data Type | Description |
|---|---|---|
| **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**

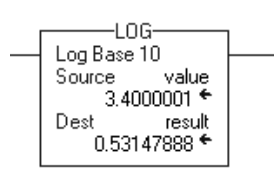| Condition | Action |
|---|---|
| prescan | The rung-condition-out is set to false. |
| rung-condition-in is false | The rung-condition-out is set to false. |
| rung-condition-in is true | The controller calculates the arc sine of the Source and places the result in the Destination.<br><br>The rung-condition-out 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.<br><br>EnableOut is set. |
| postscan | No action taken. |

**Example:**    Calculate the arc sine of *value* and place the result in *result*.
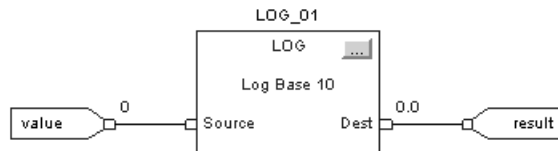
## Relay Ladder

```
        ─ASN────
  Arc Sine
  Source      value
               0.2 ←
  Dest       result
       0.20135795 ←
```
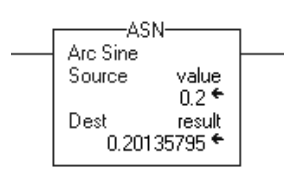
## Structured Text

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

## Function Block
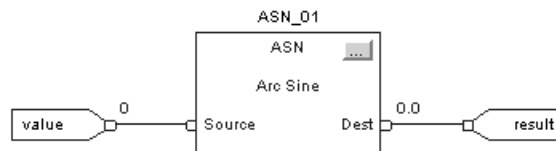
# Arc Cosine (ACS)

The ACS instruction takes the arc cosine of the Source value and stores the result in the Destination (in radians).
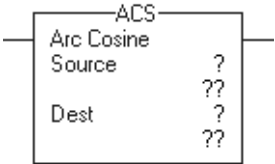
### Operands:



### Relay Ladder

| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| Source | SINT | immediate | find the arc cosine of this value |
| | INT | tag | |
| | DINT | | |
| | **REAL** | | |
| Destination | SINT | tag | tag to store the result |
| | INT | | |
| | DINT | | |
| | **REAL** | | |

### Structured Text

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

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

See [Appendix C, Structured Text Programming](#) for information on the syntax of expressions within structured text.
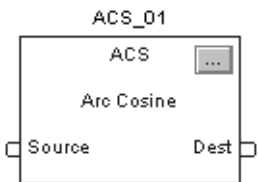
### Function Block



| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| ACS tag | FBD_MATH_ADVANCED | structure | ACS structure |

### FBD_MATH_ADVANCED Structure

| Input Parameter | Data Type | Description: |
|-----------------|-----------|--------------|
| EnableIn | BOOL | Enable input. If cleared, the instruction does not execute and outputs are not updated. |
| | | Default is set. |
| Source | REAL | Input to the math instruction. |
| | | Valid = any float |

| Input Parameter | Data Type | Description: |
|---|---|---|
| **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. |
| EnableIn is cleared | EnableOut is cleared. |
| EnableIn is set | The instruction executes. EnableOut is set. |
| postscan | No action taken. |

**Example:**   Calculate the arc cosine of *value* and place the result in *result*.

## Relay Ladder

```
        ┌─────ACS─────┐
        │Arc Cosine   │
────────┤Source  value│────────
        │  -0.60000002 ←
        │Dest   result│
        │    2.2142975 ←
        └─────────────┘
```
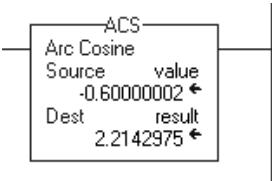
## Structured Text

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

## Function Block

# Arc Tangent (ATN)

The ATN instruction takes the arc tangent of the Source value and stores the result in the Destination (in radians).
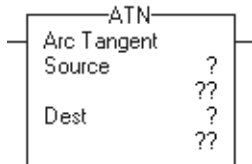
### Operands:



### Relay Ladder

| Operand: | Type | Format | Description |
|----------|------|--------|-------------|
| Source | SINT | immediate | find the arc tangent of this value |
|  | INT | tag |  |
|  | DINT |  |  |
|  | **REAL** |  |  |
| Destination | SINT | tag | tag to store the result |
|  | INT |  |  |
|  | DINT |  |  |
|  | **REAL** |  |  |

### Structured Text

`dest := ATAN(source);`

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

See Appendix C, Structured Text Programming for information on the syntax of expressions within structured text.
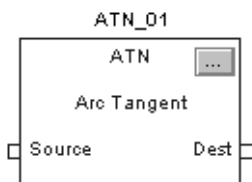
### Function Block

| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| ATN tag | FBD_MATH_ADVANCED | structure | ATN structure |

### FBD_MATH_ADVANCED Structure

| Input Parameter | Data Type | Description |
|-----------------|-----------|-------------|
| EnableIn | BOOL | Enable input. If cleared, the instruction does not execute and outputs are not updated. |
|  |  | Default is set. |
| Source | REAL | Input to the math instruction. |
|  |  | Valid = any float |

| Input Parameter | Data Type | Description |
|---|---|---|
| **Output Parameter** | **Data Type** | **Description** |
| EnableOut | BOOL | The instruction produced a valid result. |
| Dest | REAL | Result of the math instruction. Arithmetic status flags are set for this output. |

**Description:** The resulting value in the Destination is always greater than or equal to $-\pi/2$ and less than or equal to $\pi/2$ (where $\pi = 3.141593$).

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

**Fault Conditions:** none

**Execution:**

  **Relay Ladder**

| Condition | Action |
|---|---|
| prescan | The rung-condition-out is set to false. |
| rung-condition-in is false | The rung-condition-out is set to false. |
| rung-condition-in is true | The controller calculates the arc tangent of the Source and places the result in the Destination. |
| | The rung-condition-out is set to true. |
| postscan | The rung-condition-out is set to false. |

  **Function Block**

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

**Example:**    Calculate the arc tangent of *value* and place the result in *result*.

## Relay Ladder

```
              ATN
          Arc Tangent
          Source      value
                 0.86000001 ←
          Dest        result
                 0.71027106 ←
```

## Structured Text

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

## Function Block

```
                    ATN_01
                     ATN     ...
                   Arc Tangent
    value    0                       0.0
          ─────┤ Source      Dest ├─────┤ result
```

**Notes:**

# Advanced Math Instructions
## (LN, LOG, XPY)

**Introduction**

The advanced math instructions include these instructions:

| If You Want To | Use This Instruction | Available In These Languages | See Page |
|---|---|---|---|
| Take the natural log of a value. | LN | relay ladder<br><br>structured text<br><br>function block | 540 |
| Take the log base 10 of a value. | LOG | relay ladder<br><br>structured text<br><br>function block | 543 |
| Raise a value to the power of another value. | XPY | relay ladder<br><br>structured text[1]<br><br>function block | 546 |

[1] There is no equivalent structured text instruction. Use the operator in an expression.

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

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

# Natural Log (LN)

The LN instruction takes the natural log of the Source and stores the result in the Destination.
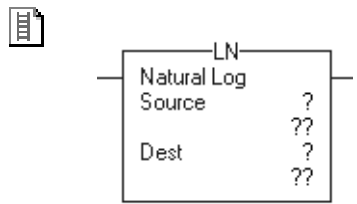
## Operands:

### Relay Ladder

| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| Source | SINT | immediate | find the natural log of this value |
| | INT | tag | |
| | DINT | | |
| | REAL | | |
| Destination | SINT | tag | tag to store the result |
| | INT | | |
| | DINT | | |
| | REAL | | |

### Structured Text

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

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

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

### Function Block

| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| LN tag | FBD_MATH_ADVANCED | structure | LN structure |

### FBD_MATH_ADVANCED Structure

| Input Parameter | Data Type | Description |
|-----------------|-----------|-------------|
| EnableIn | BOOL | Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set. |
| Source | REAL | Input to math instruction. Valid = any float |

| Output Parameter | Data Type | Description |
|------------------|-----------|-------------|
| EnableOut | BOOL | The instruction produced a valid result. |

**Description:** The Source must be greater than zero, otherwise the overflow status bit (S:V) is set. The resulting Destination is greater than or equal to -87.33655 and less than or equal to 88.72284.

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

**Fault Conditions:** none

**Execution:**

### Relay Ladder

| Condition | Action |
|---|---|
| prescan | The rung-condition-out is set to false. |
| rung-condition-in is false | The rung-condition-out is set to false. |
| rung-condition-in is true | The controller calculates the natural log of the Source and places the result in the Destination. The rung-condition-out is set to true. |
| postscan | The rung-condition-out is set to false. |

### Function Block

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

**Example:**  Calculate the natural log of *value* and place the result in *result*.

### Relay Ladder Example

```
          ─LN─
        Natural Log
        Source      value
                     5.0 ←
        Dest        result
               1.6094378 ←
```

### Structured Text

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

### Function Block
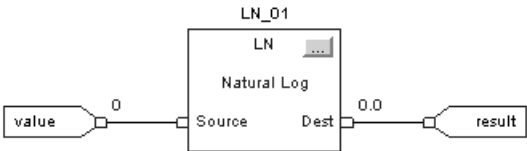
# Log Base 10 (LOG)

The LOG instruction takes the log base 10 of the Source and stores the result in the Destination.
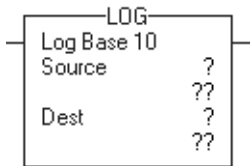
## Operands:

### Relay Ladder

| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| Source | SINT | immediate | find the log of this value |
| | INT | tag | |
| | DINT | | |
| | **REAL** | | |
| Destination | SINT | tag | tag to store the result |
| | INT | | |
| | DINT | | |
| | **REAL** | | |

### Structured Text

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

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

See Appendix C, Structured Text Programming for information on the syntax of expressions within structured text.
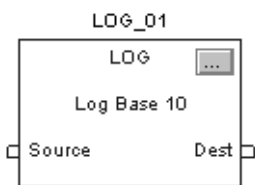
### Function Block

| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| LOG tag | FBD_MATH_ADVANCED | structure | LOG structure |

## FBD_MATH_ADVANCED Structure

| Input Parameter | Data Type | Description |
|---|---|---|
| EnableIn | BOOL | Enable input. If cleared, the instruction does not execute and outputs are not updated. |
| | | Default is set. |
| Source | REAL | Input to math instruction. |
| | | Valid = any float |

| Output Parameter | Data Type | Description |
|---|---|---|
| EnableOut | BOOL | The instruction produced a valid result. |
| Dest | REAL | Result of the math instruction. Arithmetic status flags are set for this output. |

**Description:** The Source must be greater than zero, otherwise the overflow status bit (S:V) is set. The resulting Destination is greater than or equal to -37.92978 and less than or equal to 38.53184.

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

**Fault Conditions:** none

**Execution:**

### Relay Ladder

| Condition | Action |
|---|---|
| prescan | The rung-condition-out is set to false. |
| rung-condition-in is false | The rung-condition-out is set to false. |
| rung-condition-in is true | The controller calculates the log of the Source and places the result in the Destination. |
| | The rung-condition-out is set to true. |
| postscan | The rung-condition-out is set to false. |

### Function Block

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

**Example:**    Calculate the log of *value* and place the result in *result*.

### Relay Ladder

```
        -LOG-
       Log Base 10
       Source      value
                3.4000001  ←
       Dest        result
                0.53147888 ←
```

### Structured Text

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

### Function Block

```
              LOG_01
                LOG      [...]

              Log Base 10
        value  0                    0.0
                 ─┤ Source    Dest ├─   result
```

# 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.
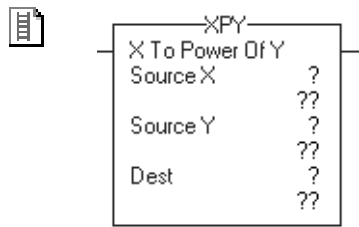
## Operands:

### 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** | | |

### Structured Text

`dest := sourceX ** sourceY;`

Use two, adjacent multiply signs "**" as an operator within an expression. This expression takes *sourceX* to the power of *sourceY* and stores the result in *dest*.

See <u>Appendix C, Structured Text Programming</u> for information on the syntax of expressions within structured text.
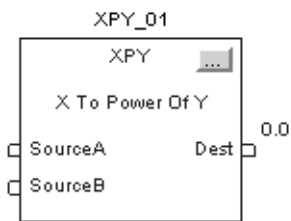
### Function Block

| 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
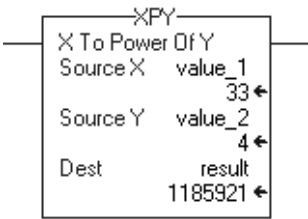


### Structured Text

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

### Function Block
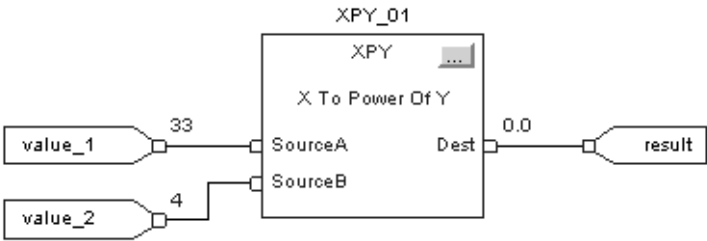
# Math Conversion Instructions

## (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 | 550 |
| | | structured text | |
| | | function block | |
| Convert degrees to radians. | RAD | relay ladder | 553 |
| | | structured text | |
| | | function block | |
| Convert an integer value to a BCD value. | TOD | relay ladder | 556 |
| | | function block | |
| Convert a BCD value to an integer value. | FRD | relay ladder | 559 |
| | | function block | |
| Remove the fractional part of a value | TRN | relay ladder | 561 |
| | TRUNC[1] | structured text | |
| | | function block | |

[1] Structured text only.

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

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

# Degrees (DEG)

The DEG instruction converts the Source (in radians) to degrees and stores the result in the Destination.
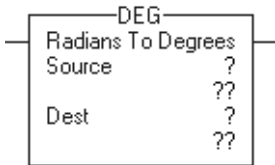
## Operands:



### Relay Ladder

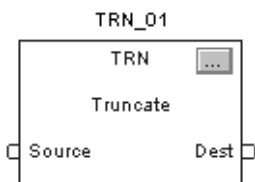| Operand | Type | Format | Description |
|---|---|---|---|
| Source | SINT | immediate | value to convert to degrees |
|  | INT | tag |  |
|  | DINT |  |  |
|  | REAL |  |  |
| Destination | SINT | tag | tag to store the result |
|  | INT |  |  |
|  | DINT |  |  |
|  | REAL |  |  |



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

### Structured Text

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

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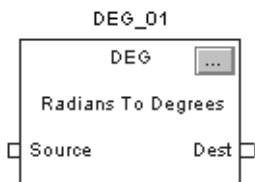


### Function Block

| Operand | Type | Format | Description |
|---|---|---|---|
| DEG tag | FBD_MATH_ADVANCED | structure | DEG structure |

### FBD_MATH_ADVANCED Structure

| Input Parameter | Data Type | Description |
|---|---|---|
| EnableIn | BOOL | Enable input. If cleared, the instruction does not execute and outputs are not updated. |
|  |  | Default is set. |
| Source | REAL | Input to the conversion instruction. |
|  |  | Valid = any float |

| Input Parameter | Data Type | Description |
|---|---|---|
| **Output Parameter** | **Data Type** | **Description** |
| EnableOut | BOOL | The instruction produced a valid result. |
| Dest | REAL | Result of the conversion instruction. Arithmetic status flags are set for this output. |

**Description:** The DEG instruction uses this algorithm:
Source*180/π (where π = 3.141593)

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

**Fault Conditions:** none

**Execution:**



### Relay Ladder

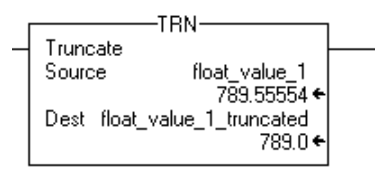| Condition | Action |
|---|---|
| prescan | The rung-condition-out is set to false. |
| rung-condition-in is false | The rung-condition-out is set to false. |
| rung-condition-in is true | The controller converts the Source to degrees and places the result in the Destination. |
| | The rung-condition-out is set to true. |
| postscan | The rung-condition-out is set to false. |



### Function Block

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

**Example:**    Convert *value* to degrees and place the result in *result*.

### Relay Ladder

```
        ┌─DEG──────────────┐
        │ Radians To Degrees│
        │ Source     value  │
        │            15.0 ← │
        │ Dest      result  │
        │      859.43671 ← │
        └───────────────────┘
```
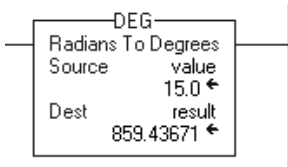
### Structured Text

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

### Function Block
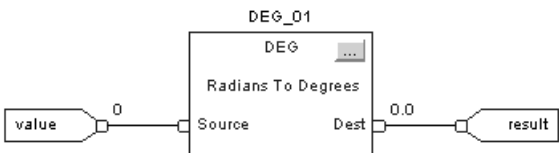
# Radians (RAD)

The RAD instruction converts the Source (in degrees) to radians and stores the result in the Destination.
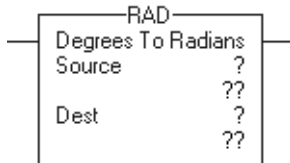
### Operands:



### Relay Ladder

| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| Source | SINT | immediate | value to convert to radians |
| | INT | tag | |
| | DINT | | |
| | REAL | | |
| Destination | SINT | tag | tag to store the result |
| | INT | | |
| | DINT | | |
| | REAL | | |

### Structured Text

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

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

See Appendix C, Structured Text Programming for information on the syntax of expressions within structured text.
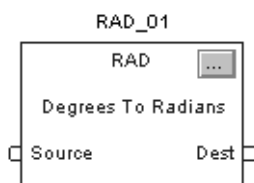
### Function Block



| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| RAD tag | FBD_MATH_ADVANCED | structure | RAD structure |

### FBD_MATH_ADVANCED Structure

| Input Parameter | Data Type | Description |
|-----------------|-----------|-------------|
| EnableIn | BOOL | Enable input. If cleared, the instruction does not execute and outputs are not updated.<br><br>Default is set. |
| Source | REAL | Input to the conversion instruction.<br><br>Valid = any float |

| Input Parameter | Data Type | Description |
|---|---|---|
| **Output Parameter** | **Data Type** | **Description** |
| EnableOut | BOOL | The instruction produced a valid result. |
| Dest | REAL | Result of the conversion instruction. Arithmetic status flags are set for this output. |

**Description:** The RAD instruction uses this algorithm:

Source*$\pi$/180 (where $\pi$ = 3.141593)

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

**Fault Conditions:** none

**Execution:**

### Relay Ladder

| Condition | Action |
|---|---|
| prescan | The rung-condition-out is set to false. |
| rung-condition-in is false | The rung-condition-out is set to false. |
| rung-condition-in is true | The controller converts the Source to radians and places the result in the Destination. The rung-condition-out is set to true. |
| postscan | The rung-condition-out is set to false. |

### Function Block

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

**Example**  Convert *value* to radians and place the result in *result*.

## Relay Ladder

```
┌─RAD──────────────────┐
│ Degrees To Radians   │
│ Source        value  │
│              45.0 ←  │
│ Dest         result  │
│        0.78539819 ←  │
└──────────────────────┘
```
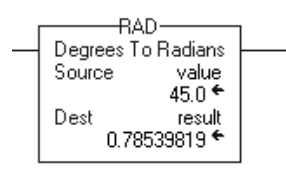
## Structured Text

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

## Function Block
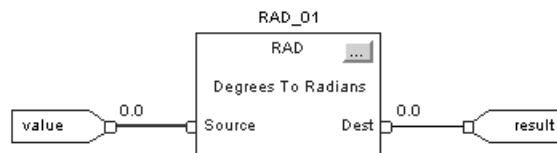
# Convert to BCD (TOD)

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

## Operands:

### Relay Ladder

| Operand | Type | Format | Description |
|---|---|---|---|
| Source | SINT | 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
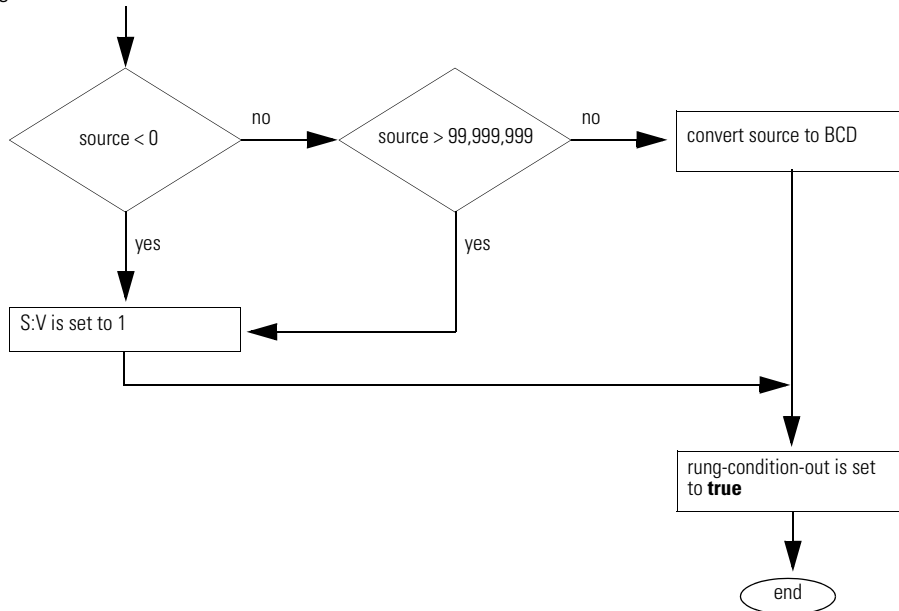


| | |
|---|---|
| rung-condition-in is true | The controller converts the Source to BCD and places the result in the Destination. |
| | The rung-condition-out is set to true. |
| postscan | The rung-condition-out is set to false. |

**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
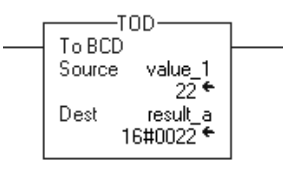


### Function Block
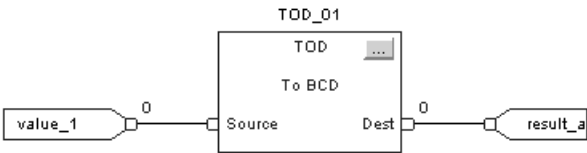
# Convert to Integer (FRD)

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.<br><br>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.<br><br>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
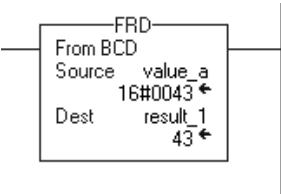


### Function Block
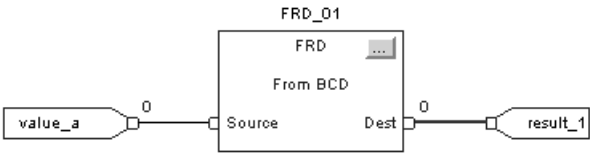
# Truncate (TRN)

The TRN instruction removes (truncates) the fractional part of the Source and stores the result in the Destination.

## Operands:

⊟

```
-TRN-
Truncate
Source        ?
             ??
Dest          ?
             ??
```
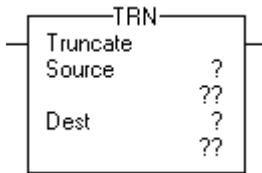
### Relay Ladder

| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| Source | REAL | immediate | value to truncate |
| | | tag | |
| Destination | SINT | tag | tag to store the result |
| | INT | | |
| | DINT | | |
| | REAL | | |

### Structured Text

⊟

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

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

See [Appendix C, Structured Text Programming](#) for information on the syntax of expressions within structured text.

### Function Block

⊟

```
        TRN_01
        TRN    [...]
        Truncate
□ Source      Dest □
```

| 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

```
           ─────TRN──────
          │ Truncate          │
          │ Source    float_value_1
          │              789.55554 ←
          │ Dest  float_value_1_truncated
          │                  789.0 ←
           ───────────────────
```

### Structured Text

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

### Function Block
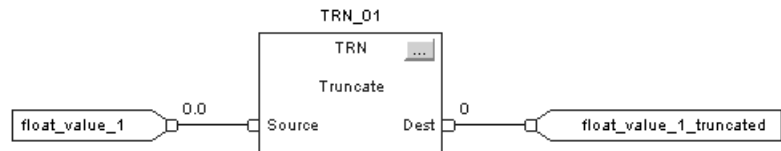
**Notes:**

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

## 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<br><br>structured text | 570 |
| count the characters in the buffer | check for the required number of characters before reading the buffer | ACB | relay ladder<br><br>structured text | 573 |
| clear the buffer<br><br>clear out ASCII Serial Port instructions that are currently executing or are in the queue | • remove old data from the buffer at start-up<br>• synchronize the buffer with a device | ACL | relay ladder<br><br>structured text | 575 |
| obtain the status of the serial port control lines<br><br>turn on or off the DTR signal<br><br>turn on or off the RTS signal | cause a modem to hang up | AHL | relay ladder<br><br>structured text | 577 |
| read a fixed number of characters | read data from a device that sends the same number of characters each transmission | ARD | relay ladder<br><br>structured text | 581 |
| 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<br><br>structured text | 585 |
| 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<br><br>structured text | 589 |
| send characters | send messages that use a variety of termination characters | AWT | relay ladder<br><br>structured text | 594 |

## Instruction Execution

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

**Logic**

| Rung-condition-in of instruction transitions from false to true |
|---|

Instruction enters the ASCII queue.

**ASCII Queue**

| Instruction 1 |
|---|
| Instruction 2 |
| Instruction 3 |
| Instruction 4 |

**ASCII Task**

| Instruction at the top of the queue executes. |
|---|

Data flows between task and buffer.

Data flows between buffer and serial port.
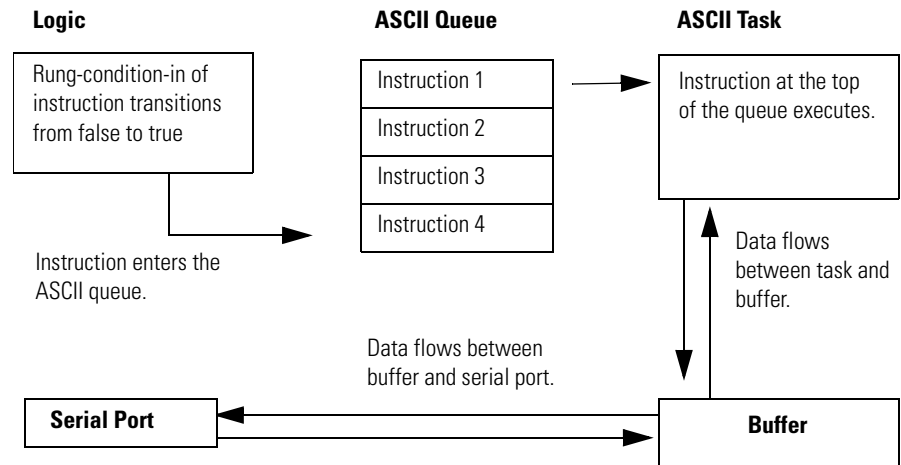
| **Serial Port** |
|---|

| **Buffer** |
|---|

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.
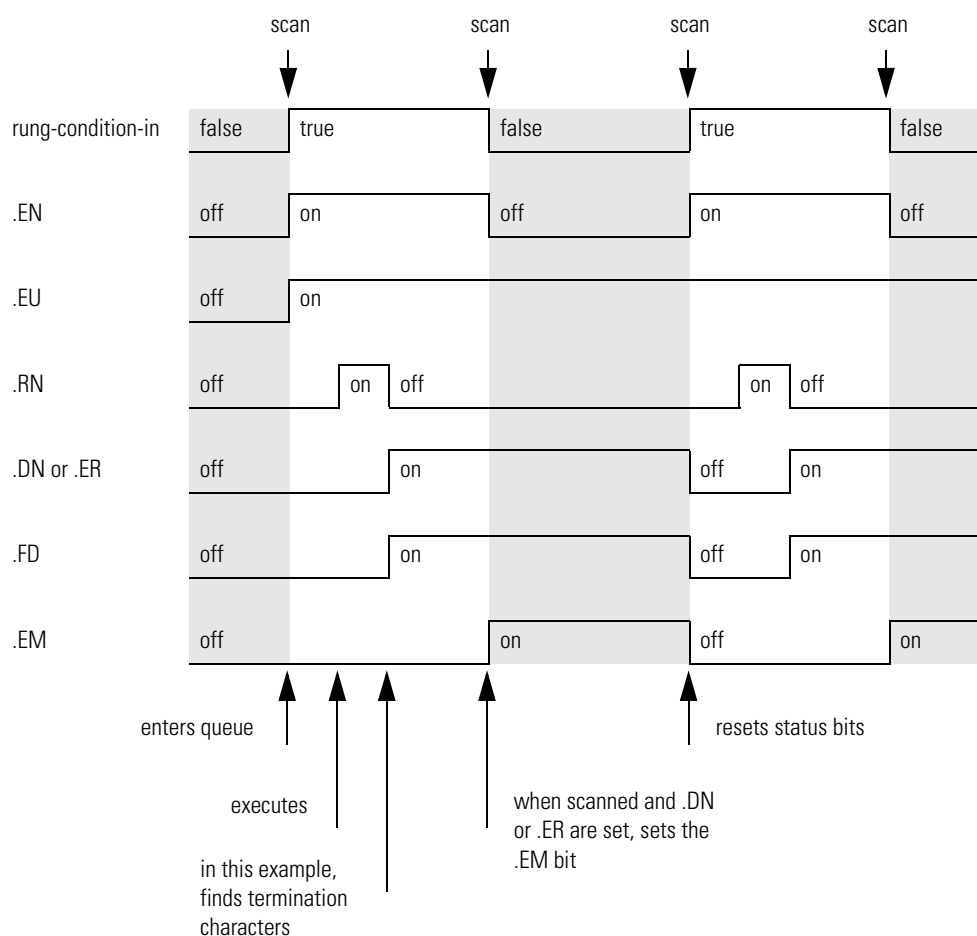


The ASCII queue holds up to 16 instructions. When the queue is full, an instruction tries to enter the queue on each subsequent scan of the instruction, as depicted below:
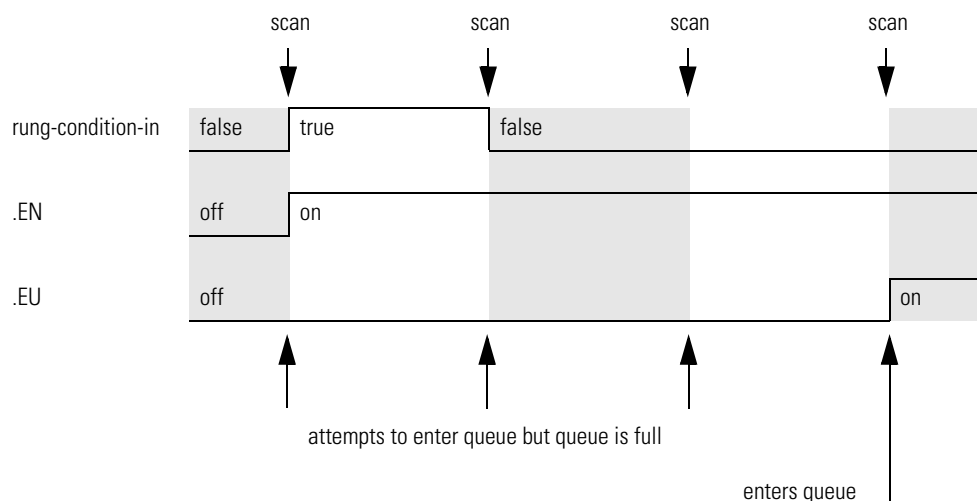
## ASCII Error Codes

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 | The LEN automatically updates to the new count of characters whenever you: <br><br>• use the String Browser dialog box to enter characters <br><br>• use instructions that read, convert, or manipulate a string <br><br>The LEN shows the length of the current string. The DATA member may contain additional, old characters, which are not included in the LEN count. |
| DATA | SINT array | ASCII characters of the string | • To access the characters of the string, address the name of the tag. <br> For example, to access the characters of the *string_1* tag, enter *string_1*. <br><br>• Each element of the DATA array contains one character. <br><br>• You can create new string data types that store less or more characters. |

# ASCII Test For Buffer Line (ABL)

The ABL instruction counts the characters in the buffer up to and including the first termination character.

## Operands:

### Relay Ladder

```
          ┌─ABL────────────────────┐
      ─────┤ ASCII Test For Buffer Line ├──(EN)──
          │ Channel                ? │  ─(DN)──
          │ SerialPort Control       │  ─(ER)──
          │ Character Count          │    ?
          └──────────────────────────┘
```
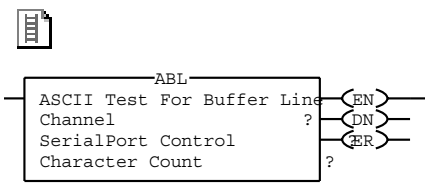
| Operand | Type | Format | Description |
|---|---|---|---|
| Channel | DINT | immediate | 0 |
| | | tag | |
| Serial Port Control | SERIAL_PORT_ CONTROL | tag | tag that controls the operation |
| Character Count | DINT | immediate | 0 |
| | | | During execution, displays the number of characters in the buffer, including the first set of termination characters. |

### Structured Text

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

The operands are the same as those for the relay ladder ABL instruction. You access the Character Count value via the .POS member of the SERIAL_PORT_CONTROL structure.

### SERIAL_PORT_CONTROL Structure

| Mnemonic | Data Type | Description |
|---|---|---|
| .EN | BOOL | The enable bit indicates that the instruction is enabled. |
| .EU | BOOL | The queue bit indicates that the instruction entered the ASCII queue. |
| .DN | BOOL | The done bit indicates when the instruction is done, but it is asynchronous to the logic scan. |
| .RN | BOOL | The run bit indicates that the instruction is executing. |
| .EM | BOOL | The empty bit indicates that the instruction is done, but it is synchronous to the logic scan. |
| .ER | BOOL | The error bit indicates when the instruction fails (errors). |
| .FD | BOOL | The found bit indicates that the instruction found the termination character or characters. |
| .POS | DINT | The position determines the number of characters in the buffer, up to and including the first set of termination characters. The instruction only returns this number after it finds the termination character or characters. |
| .ERROR | DINT | The error contains a hexadecimal value that identifies the cause of an error. |

**Description**    The ABL instruction searches the buffer for the first set of termination characters. If the instruction finds the termination characters, it:

- 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
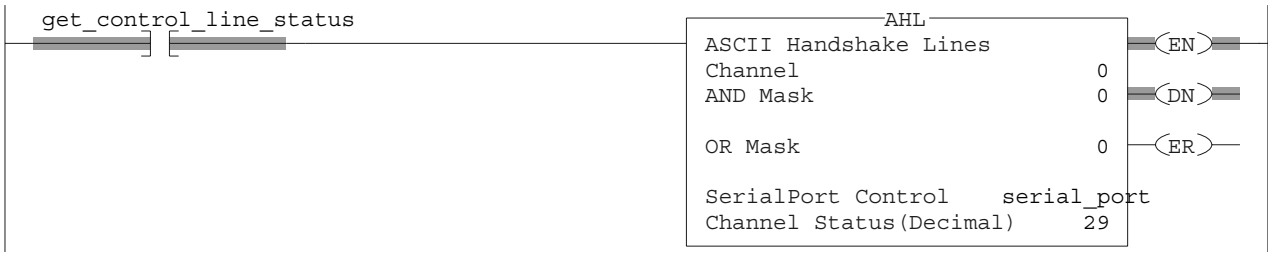
**Fault Conditions:**    none

**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.<br><br>The rung-condition-out is set to true. | na |
| EnableIn is set | na | EnableIn is always set.<br><br>The instruction executes. |
| instruction execution | The instruction counts the characters in the buffer.<br><br>The .EN bit is set.<br><br>The remaining status bits, except .UL, are cleared.<br><br>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

```
MV_line.EN                                              ┌─────ABL─────────────────────┐
  ──┤ / ├──────────────────────────────────────────────│ ASCII Test For Buffer Line  │──(EN)──
                                                        │ Channel                 0   │──(DN)──
                                                        │ SerialPort Control  MV_line │──(ER)──
                                                        │ Character Count         0   │
                                                        └─────────────────────────────┘
```
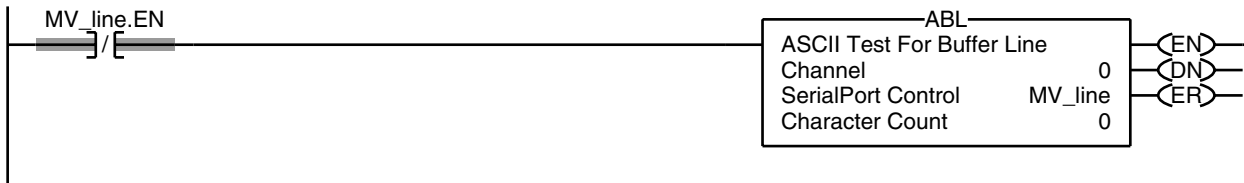
### Structured Text

```
ABL(0,MV_line);
```

## ASCII Chars in Buffer (ACB)

The ACB instruction counts the characters in the buffer.

### Operands:
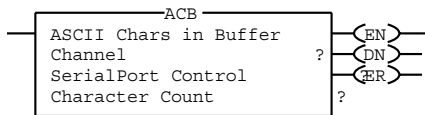


```
        ACB
ASCII Chars in Buffer      EN
Channel              ?     DN
SerialPort Control         ER
Character Count      ?
```

### Relay Ladder

| Operand | Type | Format | Enter |
|---------|------|--------|-------|
| Channel | DINT | immediate | 0 |
| | | tag | |
| Serial Port Control | SERIAL_PORT_ CONTROL | tag | tag that controls the operation |
| Character Count | DINT | immediate | 0 |
| | | | During execution, displays the number of characters in the buffer. |



```
ACB(Channel
    SerialPortControl);
```

### Structured Text

The operands are the same as those for the relay ladder ACB instruction. However, you specify the Character Count value by accessing the .POS member of the SERIAL_PORT_CONTROL structure, rather than by including the value in the operand list.

### SERIAL_PORT_CONTROL Structure

| Mnemonic | Data Type | Description |
|----------|-----------|-------------|
| .EN | BOOL | The enable bit indicates that the instruction is enabled. |
| .EU | BOOL | The queue bit indicates that the instruction entered the ASCII queue. |
| .DN | BOOL | The done bit indicates when the instruction is done, but it is asynchronous to the logic scan. |
| .RN | BOOL | The run bit indicates that the instruction is executing. |
| .EM | BOOL | The empty bit indicates that the instruction is done, but it is synchronous to the logic scan. |
| .ER | BOOL | The error bit indicates when the instruction fails (errors). |
| .FD | BOOL | The found bit indicates that the instruction found a character. |
| .POS | DINT | The position determines the number of characters in the buffer, up to and including the first set of termination characters. |
| .ERROR | DINT | The error contains a hexadecimal value that identifies the cause of an error. |

**Description:** The ACB instruction counts the characters in the buffer.

To program the ACB instruction, follow these guidelines:

**1.** Configure the serial port of the controller for user mode.

**2.** This is a transitional instruction:
- In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.
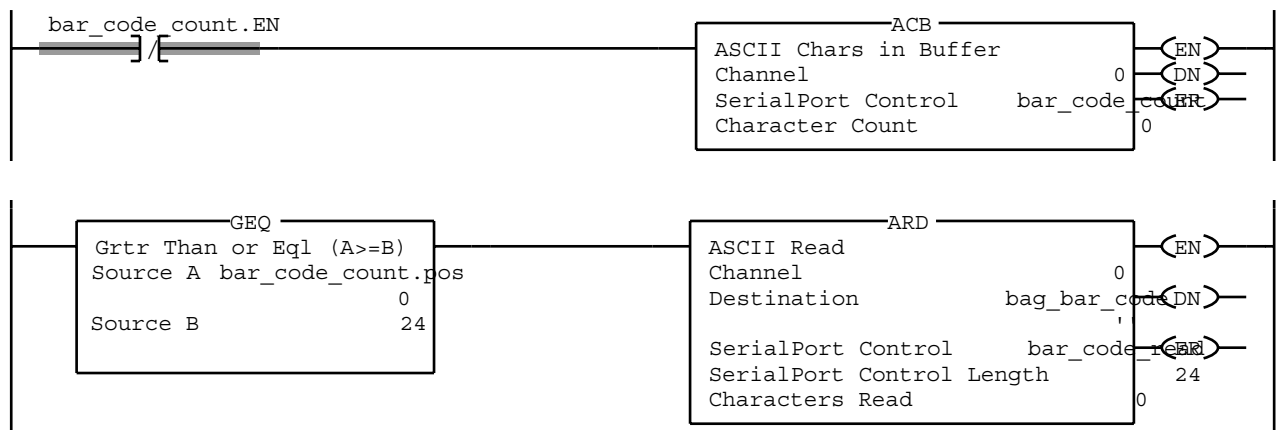- In structured text, condition the instruction so that it only executes on a transition.

**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.<br><br>The rung-condition-out is set to true. | na |
| EnableIn is set | na | EnableIn is always set.<br>The instruction executes. |
| instruction execution | The instruction counts the characters in the buffer.<br>The .EN bit is set.<br>The remaining status bits, except .UL, are cleared.<br>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**

```
  bar_code_count.EN                                    ┌─────────ACB────────────┐
  ───────]/[───────────────────────────────────────── ASCII Chars in Buffer    ─(EN)─
                                                        Channel              0  ─(DN)─
                                                        SerialPort Control  bar_code_count ─(ER)─
                                                        Character Count         0
                                                       └────────────────────────┘
```
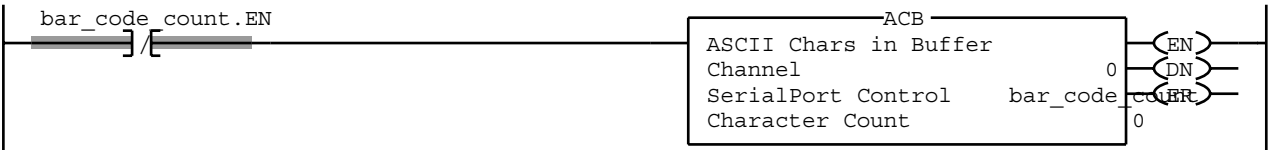
**Structured Text**

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

## ASCII Clear Buffer (ACL)

The ACL instruction immediately clears the buffer and ASCII queue.

**Operands:**

### Relay Ladder

```
┌─────────ACL─────────┐
│ASCII Clear Buffer   │
│Channel            ? │
│Clear Serial Port Read│
│Clear Serial Port Write│
└─────────────────────┘
```
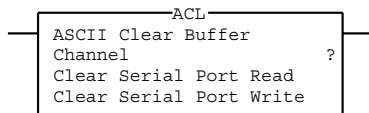
| Operand | Type | Format | Enter |
|---|---|---|---|
| Channel | DINT | immediate | 0 |
| | | tag | |
| Clear Serial Port Read | BOOL | immediate | To empty the buffer and remove ARD and ARL instructions from the queue, enter Yes. |
| | | tag | |
| Clear Serial Port Write | BOOL | immediate | To remove AWA and AWT instructions from the queue, enter Yes. |
| | | tag | |

### Structured Text

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

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

**Description:** The ACL instruction immediately performs one or both of the following actions:

- clears the buffer of characters and clears the ASCII queue of read instructions
- clears the ASCII queue of write instructions

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 |
| *does not* use ARD or ARL instructions | Select either System or User mode |

2. To determine if an instruction was removed from the queue or aborted, examine the following of the appropriate instruction:

- .ER bit is set
- .ERROR member is 16#E

**Arithmetic Status Flags:**  not affected

**Fault Conditions:**  none

### Execution:

| Condition | Relay Ladder Action | Structured Text Action |
|---|---|---|
| prescan | The rung-condition-out is set to false. | No action taken. |
| rung-condition-in is false | The rung-condition-out is set to false. | na |
| rung-condition-in is true | The instruction executes.<br><br>The rung-condition-out is set to true. | na |
| EnableIn is set | na | EnableIn is always set.<br><br>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

```
      S:FS                                                ┌─────────ACL─────────┐
   ───┤ ├───────────────────────────────────────────────┤ ASCII Clear Buffer   ├──
                                                          │ Channel           0 │
                                                          │ Clear Serial Port Read  1 │
                                                          │ Clear Serial Port Write 1 │
                                                          └─────────────────────┘
```
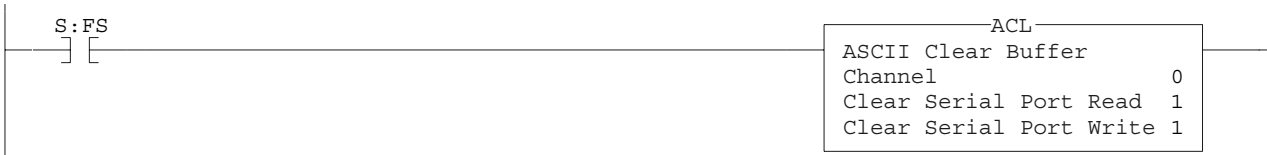
### Structured Text

```
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

```
                ┌─AHL────────────────────┐
                │ ASCII Handshake Lines   │──⟨EN⟩──
                │ Channel            ?    │
                │ AND Mask           ?    │──⟨DN⟩──
                │                    ??   │
                │ OR Mask            ?    │──⟨ER⟩──
                │                    ??   │
                │ SerialPort Control    ? │
                │ Channel Status(Decimal) ?│
                └─────────────────────────┘
```
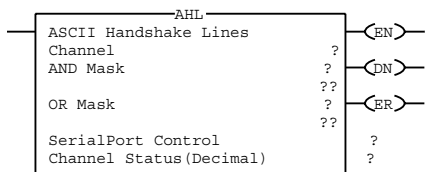
| Operand | Type | Format | Enter |
|---|---|---|---|
| Channel | DINT | immediate | 0 |
|  |  | tag |  |
| ANDMask | DINT | immediate | Refer to the description. |
|  |  | tag |  |
| ORMask | DINT | immediate |  |
|  |  | tag |  |
| Serial Port Control | SERIAL_PORT_CONTROL | tag | tag that controls 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 |

### Structured Text

```
AHL(Channel,ANDMask,ORMask,
    SerialPortControl);
```

The operands are the same as those for the relay ladder AHL instruction. However, you specify the Channel Status value by accessing the .POS member of the SERIAL_PORT_CONTROL structure, rather than by including the value in the operand list.

## SERIAL_PORT_CONTROL Structure

| Mnemonic | Data Type | Description |
|---|---|---|
| .EN | BOOL | The enable bit indicates that the instruction is enabled. |
| .EU | BOOL | The queue bit indicates that the instruction entered the ASCII queue. |
| .DN | BOOL | The done bit indicates when the instruction is done, but it is asynchronous to the logic scan. |
| .RN | BOOL | The run bit indicates that the instruction is executing. |
| .EM | BOOL | The empty bit indicates that the instruction is done, but it is synchronous to the logic scan. |
| .ER | BOOL | The error bit indicates when the instruction fails (errors). |
| .FD | BOOL | The found bit does not apply to this instruction. |
| .POS | DINT | The position stores the status of the control lines. |
| .ERROR | DINT | The error contains a hexadecimal value that identifies the cause of an error. |

**Description:**  The AHL instruction can:

- obtain the status of the control lines of the serial port
- turn on or off the data terminal ready (DTR) signal
- turn on or off the request to send signal (RTS)

To program the AHL instruction, follow these guidelines:

**1.** Configure the serial port of the controller:

| If Your Application | Then |
|---|---|
| uses ARD or ARL instructions | Select User mode |
| *does not* use ARD or ARL instructions | Select either System or User mode |

**2.** Use the following table to select the correct values for the ANDMask and ORMask operands:

| To Turn DTR | And Turn RTS: | Enter This ANDMask Value | And Enter This ORMask Value |
|---|---|---|---|
| off | off | 3 | 0 |
| | on | 1 | 2 |
| | unchanged | 1 | 0 |
| on | off | 2 | 1 |
| | on | 0 | 3 |
| | unchanged | 0 | 1 |
| unchanged | off | 2 | 0 |
| | on | 0 | 2 |
| | unchanged | 0 | 0 |

**3.** This is a transitional instruction:

- In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.

- In structured text, condition the instruction so that it only executes on a transition.

**Arithmetic Status Flags:**   not affected

## Fault Conditions:

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

## Execution:

| Condition | Relay Ladder Action | Structured Text Action |
|-----------|---------------------|------------------------|
| prescan | The rung-condition-out is set to false. | No action taken. |
| rung-condition-in is false | The rung-condition-out is set to false. | na |
| rung-condition-in is true | The instruction executes when rung-condition-in toggles from cleared to set.<br><br>The rung-condition-out is set to true. | na |
| EnableIn is set | na | EnableIn is always set.<br><br>The instruction executes. |
| instruction execution | The instruction obtains the control line status and turns on or off DTR and RTS signals.<br><br>The .EN bit is set.<br><br>The remaining status bits, except .UL, are cleared.<br><br>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

```
get_control_line_status                    ┌─────────AHL─────────┐
    ─────┤ ├─────                          │ ASCII Handshake Lines    │──(EN)──
                                           │ Channel              0   │
                                           │ AND Mask             0   │──(DN)──
                                           │                          │
                                           │ OR Mask              0   │──(ER)──
                                           │                          │
                                           │ SerialPort Control  serial_port │
                                           │ Channel Status(Decimal)  29 │
                                           └──────────────────────┘
```

### 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

```
            ARD
   ┌─────────────────────────┐
───┤ ASCII Read          ⟨EN⟩├───
   │ Channel            ?    │
   │ Destination        ⟨DN⟩ ├───
   │                 ??      │
   │ SerialPort Control ⟨ER⟩ ├───
   │ SerialPort Control Length │
   │ Characters Read     ?   │
   └─────────────────────────┘
```

| Operand | Type | Format | Enter | Notes |
|---|---|---|---|---|
| Channel | DINT | immediate<br><br>tag | 0 | |
| Destination | **string**<br><br>SINT<br><br>INT<br><br>DINT | tag | tag into which the characters are moved (read):<br><br>• For a string data type, enter the name of the tag.<br>• For a SINT, INT, or DINT array, enter the first element of the array. | • If you want to compare, convert, or manipulate the characters, use a string data type.<br>• String data types are:<br>  • default STRING data type<br>  • any new string data type that you create |
| Serial Port<br><br>Control | SERIAL_PORT_<br><br>CONTROL | tag | tag that controls the operation | |
| Serial Port<br><br>Control Length | DINT | immediate | number of characters to move to the destination (read) | • The Serial Port Control Length must be less than or equal to the size of the Destination.<br>• If you want to set the Serial Port Control Length equal to the size of the Destination, enter 0. |
| Characters Read | DINT | immediate | 0 | During execution, displays the number of characters that were read. |

### Structured Text

```
ARD(Channel,Destination,
    SerialPortControl);
```

The operands are the same as those for the relay ladder ARD instruction. However, you specify the Serial Port Control Length and the Characters Read values by accessing the .LEN and .POS members of the SERIAL_PORT_CONTROL structure, rather than by including the values in the operand list.

## SERIAL_PORT_CONTROL Structure

| Mnemonic | Data Type | Description |
|---|---|---|
| .EN | BOOL | The enable bit indicates that the instruction is enabled. |
| .EU | BOOL | The queue bit indicates that the instruction entered the ASCII queue. |
| .DN | BOOL | The done bit indicates when the instruction is done, but it is asynchronous to the logic scan. |
| .RN | BOOL | The run bit indicates that the instruction is executing. |
| .EM | BOOL | The empty bit indicates that the instruction is done, but it is synchronous to the logic scan. |
| .ER | BOOL | The error bit indicates when the instruction fails (errors). |
| .FD | BOOL | The found bit does not apply to this instruction. |
| .LEN | DINT | The length indicates the number of characters to move to the destination (read). |
| .POS | DINT | The position displays the number of characters that were read. |
| .ERROR | DINT | The error contains a hexadecimal value that identifies the cause of an error. |

**Description:** The ARD instruction removes the specified number of characters from the buffer and stores them in the Destination.

- The ARD instruction continues to execute until it removes the specified number of characters (Serial Port Control Length).
- While the ARD instruction is executing, no other ASCII Serial Port instruction executes.

To program the ARD instruction, follow these guidelines:

1. Configure the serial port of the controller for user mode.

2. Use the results of an ACB instruction to trigger the ARD instruction. This prevents the ARD instruction from holding up the ASCII queue while it waits for the required number of characters.

3. This is a transitional instruction:
   - In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.
   - In structured text, condition the instruction so that it only executes on a transition.

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 characters from the buffer and stores them in the destination. | |
| | The .EN bit is set. | |
| | The remaining status bits, except .UL, are cleared. | |
| | The instruction attempts to enter the ASCII queue. | |
| postscan | The rung-condition-out is set to false. | No action taken. |

**Example:**  A bar code reader sends bar codes to the serial port (channel 0) of the controller. Each bar code contains 24 characters. To determine when the controller receives a bar code, the ACB instruction continuously counts the characters in the buffer. When the buffer contains at least 24 characters, the controller has received a bar code. The ARD instruction moves the bar code to the DATA member of the *bag_bar_code* tag, which is a string.

### Relay Ladder

**Structured Text**

```
ACB(0,bar_code_count);

IF bar_code_count.POS >= 24 THEN

    bar_code_read.LEN := 24;

    ARD(0,bag_bar_code,bar_code_read);

END_IF;
```

# ASCII Read Line (ARL)

The ARL instruction removes specified characters from the buffer and stores them in the Destination.

## Operands:

### Relay Ladder



```
              ┌──ARL─────────────────┐
              │ ASCII Read Line       ├──(EN)──
              │ Channel              ?│
              │ Destination           ├──(DN)──
              │                     ??│
              │ SerialPort Control    ├──(ER)──
              │ SerialPort Control Length   ?│
              │ Characters Read      ?│
              └───────────────────────┘
```

| Operand | Type | Format | Enter | Notes |
|---|---|---|---|---|
| Channel | DINT | immediate<br><br>tag | 0 | |
| Destination | **string**<br><br>SINT<br><br>INT<br><br>DINT | tag | tag into which the characters are moved (read):<br><br>• For a string data type, enter the name of the tag.<br>• For a SINT, INT, or DINT array, enter the first element of the array. | • If you want to compare, convert, or manipulate the characters, use a string data type.<br>• String data types are:<br>  • default STRING data type<br>  • any new string data type that you create |
| Serial Port Control | SERIAL_PORT_ CONTROL | tag | tag that controls the operation | |
| Serial Port Control Length | DINT | immediate | maximum number of characters to read if no termination characters are found | • Enter the maximum number of characters that any message will contain (that is, when to stop reading if no termination characters are found).<br><br>For example, if messages range from 3 to 6 characters in length, enter 6.<br><br>• The Serial Port Control Length must be less than or equal to the size of the Destination.<br>• If you want to set the Serial Port Control Length equal to the size of the Destination, enter 0. |
| Characters Read | DINT | immediate | 0 | During execution, displays the number of characters that were read. |

### Structured Text



```
ARL(Channel,Destination,
    SerialPortControl);
```

The operands are the same as those for the relay ladder ARL instruction. However, you specify the Serial Port Control Length and the Characters Read values by accessing the .LEN and .POS members of the

SERIAL_PORT_CONTROL structure, rather than by including the values in the operand list.

## SERIAL_PORT_CONTROL Structure

| Mnemonic | Data Type | Description |
|---|---|---|
| .EN | BOOL | The enable bit indicates that the instruction is enabled. |
| .EU | BOOL | The queue bit indicates that the instruction entered the ASCII queue. |
| .DN | BOOL | The done bit indicates when the instruction is done, but it is asynchronous to the logic scan. |
| .RN | BOOL | The run bit indicates that the instruction is executing. |
| .EM | BOOL | The empty bit indicates that the instruction is done, but it is synchronous to the logic scan. |
| .ER | BOOL | The error bit indicates when the instruction fails (errors). |
| .FD | BOOL | The found bit does not apply to this instruction. |
| .LEN | DINT | The length indicates the maximum number of characters to move to the destination (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.

**4.**

**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.<br><br>The rung-condition-out is set to true. | na |
| EnableIn is set | na | EnableIn is always set.<br><br>The instruction executes. |
| instruction execution | The instruction removes the specified characters from the buffer and stores them in the destination.<br><br>The .EN bit is set.<br><br>The remaining status bits, except .UL, are cleared.<br><br>The instruction attempts to enter the ASCII queue. | |
| postscan | The rung-condition-out is set to false. | No action taken. |

**Example:**  Continuously test the buffer for a message from a MessageView terminal. Since each message ends in a carriage return ($r), the carriage return is configured as the termination character in the Controller Properties dialog box, User Protocol tab. When the ABL finds a carriage return, its sets the FD bit.

When the ABL instruction finds the carriage return (*MV_line*.FD is set), the controller has received a complete message. The ARL instruction removes the characters from the buffer, up to and including the carriage return, and places them in the DATA member of the *MV_msg* tag, which is a string.

## Relay Ladder

```
MV_line.EN                                      ┌─ABL──────────────────────┐
  ─┘/├─────────────────────────────────────────│ ASCII Test For Buffer Line │──(EN)
                                                │ Channel                  0 │──(DN)
                                                │ SerialPort Control  MV_line │──(ER)
                                                │ Character Count          0 │
                                                └────────────────────────────┘

MV_line.FD                                      ┌─ARL──────────────────────┐
  ─┘├──────────────────────────────────────────│ ASCII Read Line            │──(EN)
                                                │ Channel                  0 │
                                                │ Destination       MV_msg  │──(DN)
                                                │                         "  │
                                                │ SerialPort Control MV_read │──(ER)
                                                │ SerialPort Control Length 12│
                                                │ Characters Read          0 │
                                                └────────────────────────────┘
```
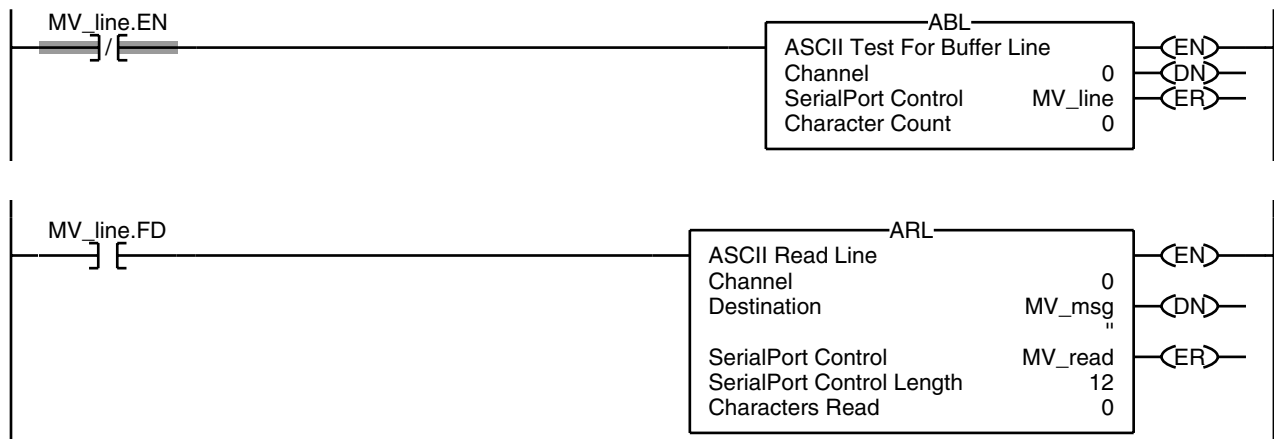
## Structured Text

```
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

```
        ┌──────── AWA ────────┐
  ┌─────┤ ASCII Write Append  ├──(EN)──
        │ Channel          ?  │
        │ Source           ?  ├──(DN)──
        │                 ??  │
        │ SerialPort Control  ├──(ER)──
        │ SerialPort Control Length   ? │
        │ Characters Sent     │  ?
        └─────────────────────┘
```

| Operand | Type | Format | Enter | Notes |
|---------|------|--------|-------|-------|
| Channel | DINT | immediate<br><br>tag | 0 | |
| Source | **string**<br><br>SINT<br><br>INT<br><br>DINT | tag | tag that contains the characters to send:<br><br>• For a string data type, enter the name of the tag.<br>• For a SINT, INT, or DINT array, enter the first element of the array. | • If you want to compare, convert, or manipulate the characters, use a string data type.<br>• String data types are:<br>  • default STRING data type<br>  • any new string data type that you create |
| Serial Port Control | SERIAL_PORT_ CONTROL | tag | tag that controls the operation | |
| Serial Port Control Length | DINT | immediate | number of characters to send | • The Serial Port Control Length must be less than or equal to the size of the Source.<br>• If you want to set the Serial Port Control Length equal to the number of characters in the Source, enter 0. |
| Characters Sent | DINT | immediate | 0 | During execution, displays the number of characters that were sent. |

### Structured Text

```
AWA(Channel,Source,
    SerialPortControl);
```

The operands are the same as those for the relay ladder AWA instruction. However, you specify the Serial Port Control Length and the Characters Sent values by accessing the .LEN and .POS members of the SERIAL_PORT_CONTROL structure, rather than by including the values in the operand list.

## SERIAL_PORT_CONTROL Structure

| Mnemonic | Data Type | Description |
|---|---|---|
| .EN | BOOL | The enable bit indicates that the instruction is enabled. |
| .EU | BOOL | The queue bit indicates that the instruction entered the ASCII queue. |
| .DN | BOOL | The done bit indicates when the instruction is done, but it is asynchronous to the logic scan. |
| .RN | BOOL | The run bit indicates that the instruction is executing. |
| .EM | BOOL | The empty bit indicates that the instruction is done, but it is synchronous to the logic scan. |
| .ER | BOOL | The error bit indicates when the instruction fails (errors). |
| .FD | BOOL | The found bit does not apply to this instruction. |
| .LEN | DINT | The length indicates the number of characters to send. |
| .POS | DINT | The position displays the number of characters that were sent. |
| .ERROR | DINT | The error contains a hexadecimal value that identifies the cause of an error. |

**Description:**    The AWA instruction:

- sends the specified number of characters (Serial Port Control Length) of the Source tag to the device that is connected to the serial port of the controller
- adds to the end of the characters (appends) either one or two characters that are defined in the Controller Properties dialog box, User Protocol tab

To program the AWA instruction, follow these guidelines:

1. Configure the serial port of the controller:

   a. Does your application also include ARD or ARL instructions?

   | If | Then |
   |---|---|
   | Yes | Select User mode |
   | No | Select either System or User mode |

   b. Define the characters to append to the data.

2. This is a transitional instruction:
   - In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.
   - In structured text, condition the instruction so that it only executes on a transition.

**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

```
temp_high                                              ┌────────AWA────────┐
  ] [                                                  │ ASCII Write Append │──<EN>
                                                       │ Channel          0 │
                                                       │ Source    string[1]│──<DN>
                                                       │           '$1425\1'│
                                                       │ SerialPort Control │──<ER>
                                                       │       temp_high_write│
                                                       │ SerialPort Control Length  5│
                                                       │ Characters Sent    6│
                                                       └────────────────────┘
```
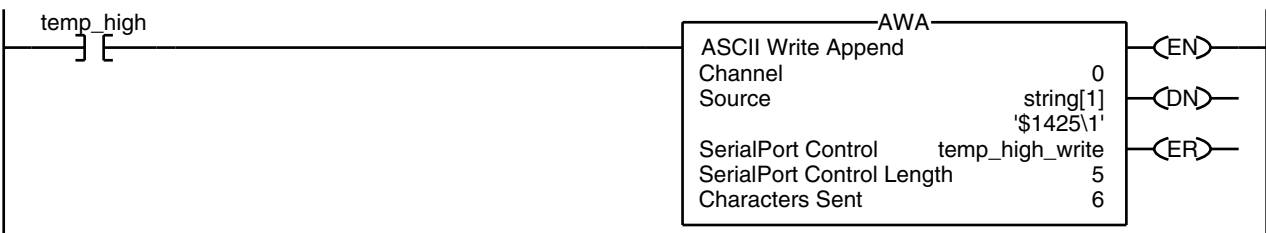
### Structured Text

```
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

```
  alarm              ┌──────MOV──────┐   ┌──────────AWA──────────┐
───┤ ├───────────────┤Move           ├───┤ ASCII Write Append    ├──<EN>──
                      │Source alarm_msg.LEN│ Channel             0 │
                      │              5│   │ Source       alarm_msg├──<DN>──
                      │Dest  alarm_write.LEN│              '$1425\1'│
                      │              5│   │ SerialPort Control  alarm_write├──<ER>──
                      └───────────────┘   │ SerialPort Control Length   5 │
                                          │ Characters Sent             6 │
                                          └───────────────────────┘
```
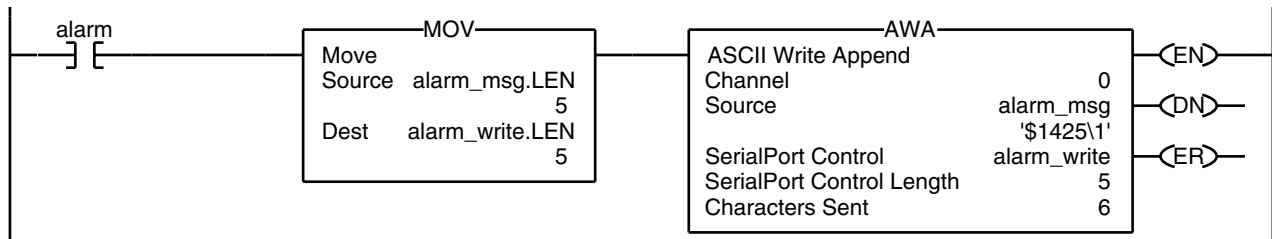
### Structured Text

```
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 | **string** | tag | tag that contains the characters to send: | • If you want to compare, convert, or manipulate the characters, use a string data type. |
| | SINT | | | |
| | INT | | • For a string data type, enter the name of the tag. | • String data types are: |
| | DINT | | | • default STRING data type |
| | | | • For a SINT, INT, or DINT array, enter the first element of the array. | • any new string data type that you create |
| Serial Port Control | SERIAL_PORT_ CONTROL | tag | tag that controls the operation | |
| Serial Port Control Length | DINT | immediate | number of characters to send | • The Serial Port Control Length must be less than or equal to the size of the Source. |
| | | | | • If you want to set the Serial Port Control Length equal to the number of characters in the Source, enter 0. |
| Characters Sent | DINT | immediate | 0 | During execution, displays the number of characters that were sent. |

### Structured Text

```
AWT(Channel,Source,
    SerialPortControl);
```

The operands are the same as those for the relay ladder AWT instruction. However, you specify the Serial Port Control Length and the Characters Sent values by accessing the .LEN and .POS members of the SERIAL_PORT_CONTROL structure, rather than by including the values in the operand list

## SERIAL_PORT_CONTROL Structure

| Mnemonic | Data Type | Description |
|---|---|---|
| .EN | BOOL | The enable bit indicates that the instruction is enabled. |
| .EU | BOOL | The queue bit indicates that the instruction entered the ASCII queue. |
| .DN | BOOL | The done bit indicates when the instruction is done, but it is asynchronous to the logic scan. |
| .RN | BOOL | The run bit indicates that the instruction is executing. |
| .EM | BOOL | The empty bit indicates that the instruction is done, but it is synchronous to the logic scan. |
| .ER | BOOL | The error bit indicates when the instruction fails (errors). |
| .FD | BOOL | The found bit does not apply to this instruction. |
| .LEN | DINT | The length indicates the number of characters to send. |
| .POS | DINT | The position displays the number of characters that were sent. |
| .ERROR | DINT | The error contains a hexadecimal value that identifies the cause of an error. |

**Description:** The AWT instruction sends the specified number of characters (Serial Port Control Length) of the Source tag to the device that is connected to the serial port of the controller.

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 |
| *does not* use ARD or ARL instructions | Select either System or User mode |

2. This is a transitional instruction:
   - In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.
   - In structured text, condition the instruction so that it only executes on a transition.

3. Each time the instruction executes, do you always send the same number of characters?

| If | Then |
|---|---|
| Yes | In the Serial Port Control Length, enter the number of characters to send. |
| No | Before the instruction executes, move the LEN member of the Source tag to the LEN member of the Serial Port Control tag. |

**Arithmetic Status Flags:**  not affected

**Fault Conditions:**  none

**Execution:**

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

```
  temp_low                                    ─────AWT─────
──┤ ├┤├────────────────────────────────────  ASCII Write              ─⟨EN⟩─
                                              Channel              0
                                              Source          string[2]  ─⟨DN⟩─
                                                        '$142224\01$r'
                                              SerialPort Control  temp_low_write  ─⟨ER⟩─
                                              SerialPort Control Length    9
                                              Characters Sent            9
                                              ─────────────
```
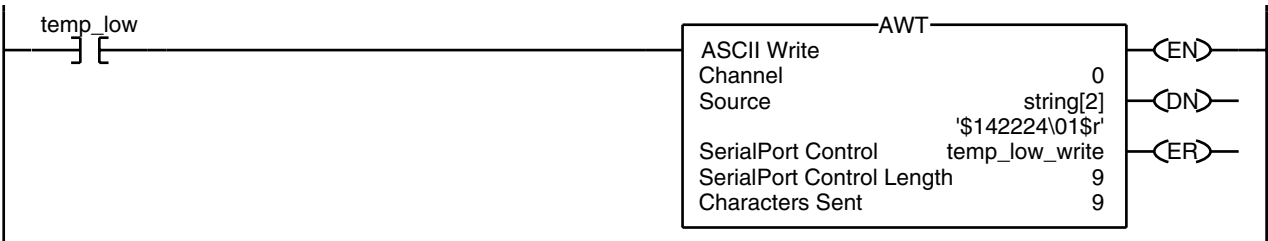
### Structured Text

```
osri_1.InputBit := temp_low;

OSRI(osri_1);

IF (osri_1.OutputBit) THEN
     temp_low_write.LEN := 9;
     AWT(0,string[2],temp_low_write);
END_IF;
```

**Example 2:**    When *MV_update* is set, the AWT instruction sends the characters in *MV_msg*. Because the number of characters in *MV_msg* varies, the rung first moves the length of the string (*MV_msg*.LEN) to the Serial Port Control Length of the AWT instruction (*MV_write*.LEN). In *MV_msg*, the *$16* counts as one character. It is the hex code for the Ctrl-V character.

### Relay Ladder
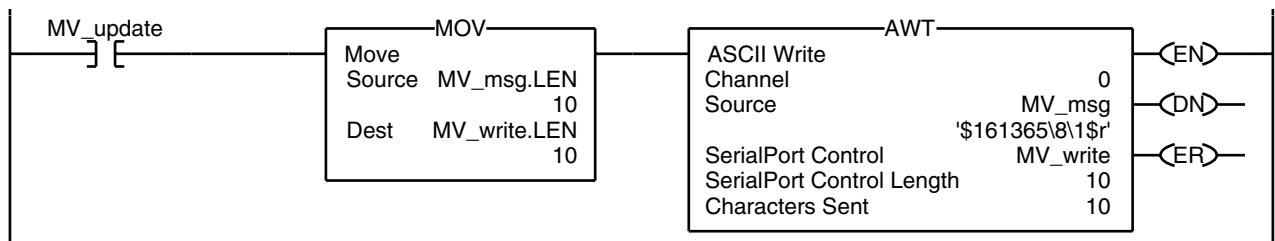


### Structured Text

```
osri_1.InputBit := MV_update;

OSRI(osri_1);


IF (osri_1.OutputBit) THEN

    MV_write.LEN := Mv_msg.LEN;

    AWT(0,MV_msg,MV_write);

END_IF;
```

**Notes:**

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

## Introduction

Use the ASCII string instructions to modify and create strings of ASCII characters.

| If you want to | For example | Use this instruction | Available in these languages | See page |
|---|---|---|---|---|
| add characters to the end of a string | add termination characters or delimiters to a string | CONCAT | relay ladder<br><br>structured text | 601 |
| delete characters from a string | remove header or control characters from a string | DELETE | relay ladder<br><br>structured text | 603 |
| determine the starting character of a sub-string | locate a group of characters within a string | FIND | relay ladder<br><br>structured text | 605 |
| insert characters into a string | create a string that uses variables | INSERT | relay ladder<br><br>structured text | 607 |
| extract characters from a string | extract information from a bar code | MID | relay ladder<br><br>structured text | 609 |

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 | 206 |
| see if the characters are equal to specific characters | EQU | 211 |
| see if the characters are not equal to specific characters | NEQ | 242 |
| see if the characters are equal to or greater than specific characters | GEQ | 211 |
| see if the characters are greater than specific characters | GRT | 219 |
| see if the characters are equal to or less than specific characters | LEQ | 223 |
| see if the characters are less than specific characters | LES | 227 |
| rearrange the bytes of a INT, DINT, or REAL tag | SWPB | 299 |
| find a string in an array of strings | FSC | 346 |
| convert characters to a SINT, INT, DINT, or REAL value | STOD | 614 |

| If you want to | Use this instruction | See page |
|---|---|---|
| convert characters to a REAL value | STOR | 616 |
| convert a SINT, INT, DINT, or REAL value to a string of ASCII characters | DTOS | 619 |
| convert REAL value to a string of ASCII characters | RTOS | 621 |

## 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 | The LEN automatically updates to the new count of characters whenever you: <br><br> • use the String Browser dialog box to enter characters <br> • use instructions that read, convert, or manipulate a string <br><br> The LEN shows the length of the current string. The DATA member may contain additional, old characters, which are not included in the LEN count. |
| DATA | SINT array | ASCII characters of the string | • To access the characters of the string, address the name of the tag. <br>   For example, to access the characters of the *string_1* tag, enter *string_1*. <br> • Each element of the DATA array contains one character. <br> • You can create new string data types that store less or more characters. |

# String Concatenate (CONCAT)

The CONCAT instruction adds ASCII characters to the end of a string.

### Operands:

### Relay Ladder

```
        CONCAT
  String Concatenate
  Source A        ?
                 ??
  Source B        ?
                 ??
  Dest            ?
                 ??
```
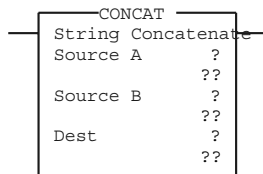
| Operand | Type | Format | Enter | Notes |
|---|---|---|---|---|
| Source A | string | tag | tag that contains the initial characters | String data types are: |
| Source B | string | tag | tag that contains the end characters | • default STRING data type<br>• any new string data type that you create |
| Destination | string | tag | tag to store the result | |

### Structured Text

```
CONCAT(SourceA,SourceB,
    Dest);
```

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

**Description:** The CONCAT instruction combines the characters in Source A with the characters in Source B and places the result in the Destination.

- The characters from Source A are first, followed by the characters from Source B.
- Unless Source A and the Destination are the same tag, Source A remains unchanged.

**Arithmetic Status Flags:** not affected

### Fault Conditions:

| Type | Code | Cause | Recovery Method |
|---|---|---|---|
| 4 | 51 | The LEN value of the string tag is greater than the DATA size of the string tag. | 1. Check that no instruction is writing to the LEN member of the string tag.<br><br>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.<br><br>The rung-condition-out is set to true. | na |
| EnableIn is set | na | EnableIn is always set.<br><br>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

```
 add_node                                        ┌─── CONCAT ───┐
───┤ ├──────────────────────────────────────────│ String Concatenate      │
                                                 │ Source A        string_1│
                                                 │                  '$1423\'│
                                                 │ Source B  node_num_ascii│
                                                 │                       '1│
                                                 │ Dest               msg  │
                                                 │                 '$1423\1'│
                                                 └─────────────────────────┘
```

### 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

```
        ┌─ DELETE ──────┐
        │ String Delete  │
        │ Source      ?  │
        │            ??  │
        │ Qty         ?  │
        │            ??  │
        │ Start       ?  │
        │            ??  │
        │ Dest        ?  │
        │            ??  │
        └────────────────┘
```
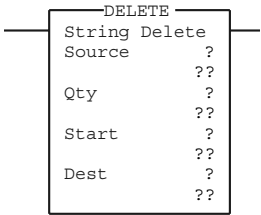
| Operand | Type | Format | Enter | Notes |
|---------|------|--------|-------|-------|
| Source | string | tag | tag that contains the string from which you want to delete characters | String data types are:<br>• default STRING data type<br>• any new string data type that you create |
| Quantity | SINT | 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 |  |  |
|  | **DINT** |  |  |  |
| Start | SINT | immediate | position of the first character to delete | Enter a number between 1 and the DATA size of the Source. |
|  | INT | tag |  |  |
|  | **DINT** |  |  |  |
| Destination | string | tag | tag to store the result |  |

### Structured Text

```
DELETE(Source,Qty,Start,
    Dest);
```

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

**Description:** The DELETE instruction deletes (removes) a group of characters from the Source and places the remaining characters in the Destination.

- The Start position and Quantity define the characters to remove.
- Unless the Source and Destination are the same tag, the Source remains unchanged.

**Arithmetic Status Flags:**  not affected

**Fault Conditions:**

| Type | Code | Cause | Recovery Method |
|------|------|-------|-----------------|
| 4 | 51 | The LEN value of the string tag is greater than the DATA size of the string tag. | 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. | na |
| | The rung-condition-out is set to true. | |
| 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**

```
 term_read.EM                                        ┌─────DELETE─────┐
──────┤ ├──────────────────────────────────────────│ String Delete  │
                                                     │ Source  term_input│
                                                     │          '$0655'│
                                                     │ Qty           1 │
                                                     │                 │
                                                     │ Start         1 │
                                                     │                 │
                                                     │ Dest    term_text│
                                                     │            '55  │
                                                     └─────────────────┘
```
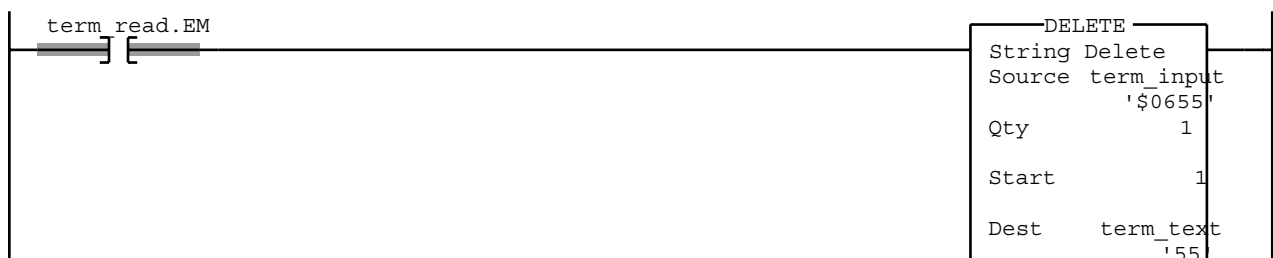
**Structured Text**

```
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

```
         ┌──FIND──┐
         Find String
         Source      ?
                    ??
         Search      ?
                    ??
         Start       ?
                    ??
         Result      ?
                    ??
```
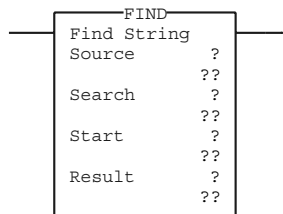
| Operand | Type | Format | Enter | Notes |
|---------|------|--------|-------|-------|
| Source | string | tag | string to search in | String data types are: |
| Search | string | tag | string to find | • default STRING data type <br> • any new string data type that you create |
| Start | SINT | immediate | 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. <br> 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. | na |
| | The rung-condition-out is set to true. | |
| 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**

```
MV_read.EM                                              ┌──────FIND──────┐
──────┤ ├──────                                         │Find String     │
                                                        │Source    MV_msg│
                                                        │  '$06324\12\1\$r'│
                                                        │Search      find│
                                                        │            '\' │
                                                        │Start          1│
                                                        │                │
                                                        │Result   find_pos│
                                                        │               5│
                                                        └────────────────┘
```
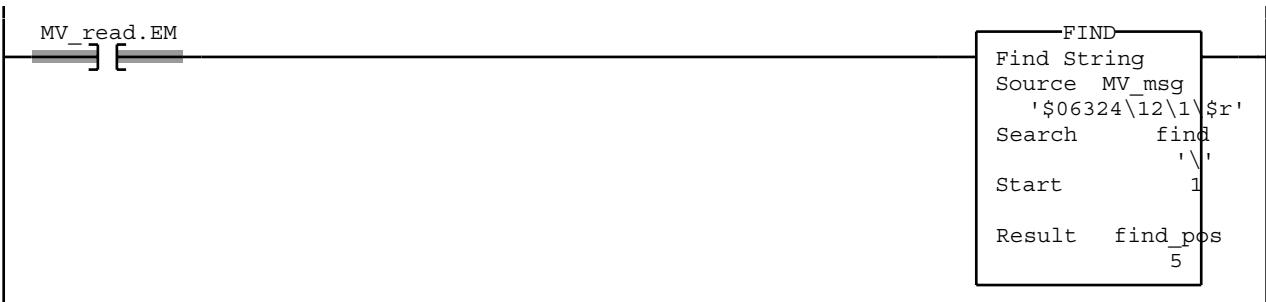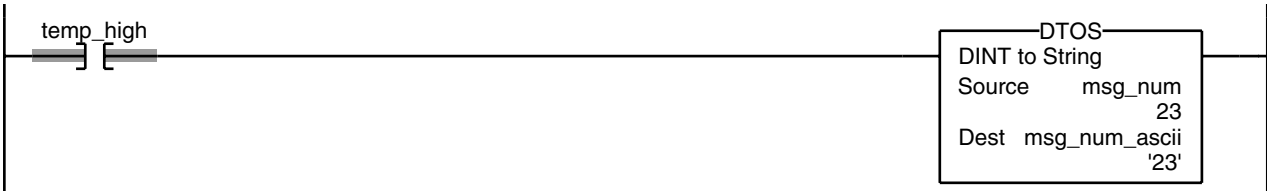
**Structured Text**

```
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

```
       ┌─INSERT───────┐
       │Insert String │
       │Source A     ?│
       │            ??│
       │Source B     ?│
       │            ??│
       │Start        ?│
       │            ??│
       │Dest         ?│
       │            ??│
       └──────────────┘
```
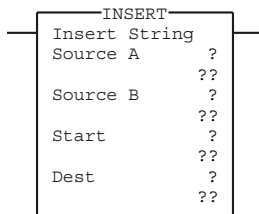
| Operand | Type | Format | Enter | Notes |
|---------|------|--------|-------|-------|
| Source A | string | tag | string to add the characters to | String data types are:<br><br>• default STRING data type<br>• any new string data type that you create |
| Source B | string | tag | string containing the characters to add | |
| Start | SINT | immediate | position in Source A to add the characters | Enter a number between 1 and the DATA size of the Source. |
| | INT | tag | | |
| | **DINT** | | | |
| Result | string | tag | string to store the result | |

### Structured Text

```
INSERT(SourceA,SourceB,
    Start,Dest);
```

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.<br><br>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.<br><br>The rung-condition-out is set to true. | na |
| EnableIn is set | na | EnableIn is always set.<br><br>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

```
    temp_high                                      ┌─────INSERT─────┐
──────┤ ├─────────────────────────────────────────│Insert String   │
                                                   │Source A     str│
                                                   │         'AD'    │
                                                   │Source B     str│
                                                   │         'BC'    │
                                                   │Start            │
                                                   │                 │
                                                   │Dest            s│
                                                   │       'ABCD'    │
                                                   └─────────────────┘
```

### 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

```
      -MID-
  Middle String
  Source      ?
              ??
  Qty         ?
              ??
  Start       ?
              ??
  Dest        ?
              ??
```
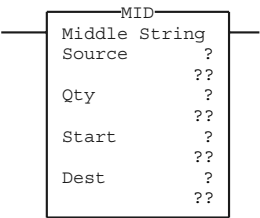
| Operand | Type | Format | Enter | Notes |
|---|---|---|---|---|
| Source | string | tag | string to copy characters from | String data types are:<br><br>• default STRING data type<br><br>• any new string data type that you create |
| Quantity | SINT | immediate | number of characters to copy | The Start plus the Quantity must be less than or equal to the DATA size of the Source. |
|  | INT | tag |  |  |
|  | **DINT** |  |  |  |
| Start | SINT | immediate | position of the first character to copy | Enter a number between 1 and the DATA size of the Source. |
|  | INT | tag |  |  |
|  | **DINT** |  |  |  |
| Destination | string | tag | string to copy the characters to |  |

### Structured Text

```
MID(Source,Qty,Start,
    Dest);
```

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

**Description:** The MID instruction copies a group of characters from the Source and places the result in the Destination.

- The Start position and Quantity define the characters to copy.
- Unless the Source and Destination are the same tag, the Source remains unchanged.

**Arithmetic Status Flags:** not affected

**Fault Conditions:**

| Type | Code | Cause | Recovery Method |
|---|---|---|---|
| 4 | 51 | The LEN value of the string tag is greater than the DATA size of the string tag. | 1. Check that no instruction is writing to the LEN member of the string tag. |
| | | | 2. In the LEN value, enter the number of characters that the string contains. |
| 4 | 56 | The Start or Quantity value is invalid. | 1. Check that the Start value is between 1 and the DATA size of the Source. |
| | | | 2. Check that the Start value plus the Quantity value is less than or equal to the DATA size of the Source. |

**Execution:**

| Condition | Relay Ladder Action | Structured Text Action |
|---|---|---|
| prescan | The rung-condition-out is set to false. | No action taken. |
| rung-condition-in is false | The rung-condition-out is set to false. | na |
| rung-condition-in is true | The instruction executes. The rung-condition-out is set to true. | na |
| EnableIn is set | na | EnableIn is always set. The instruction executes. |
| instruction execution | The instruction copies the specified characters from a string and stores them in another string. | |
| postscan | The rung-condition-out is set to false. | No action taken. |

**Example:**     In a baggage handling conveyor of an airport, each bag gets a bar code. Characters 9 - 17 of the bar code are the flight number and destination airport of the bag. After the bar code is read (*bag_read.EM* is set) the MID instruction copies the flight number and destination airport to the *bag_flt_and_dest* string.

**Relay Ladder**

```
bag_read.EM                                        ┌────────MID────────────┐
──────┤ ├──────────────────────────────────────────│Middle String          │
                                                    │Source      bag_barcode│
                                                    │  'NWA HOP 5058 AMS 01'│
                                                    │Qty                   9│
                                                    │                       │
                                                    │Start                  │
                                                    │                       │
                                                    │Dest     bag_flt_and_dest
                                                    │         '5058 AMS '   │
                                                    └───────────────────────┘
```
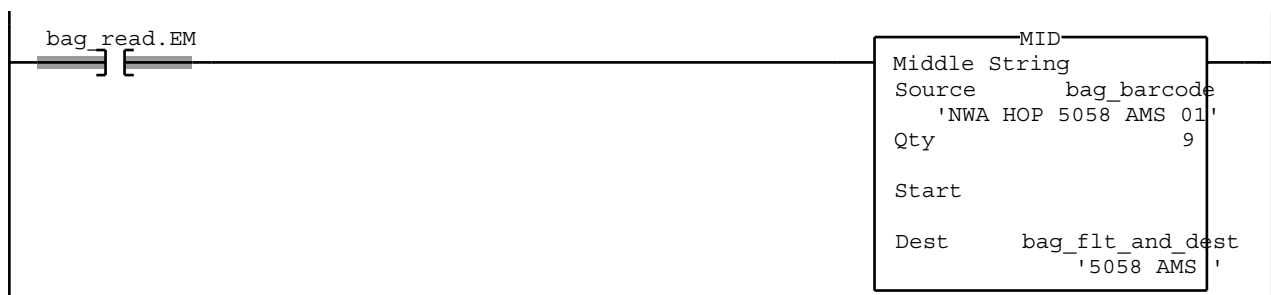
**Structured Text**

```
IF bag_read.EM THEN

    MID(bar_barcode,9,9,bag_flt_and_dest);

    bag_read.EM := 0;

END_IF;
```

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

## Introduction

Use the ASCII conversion instructions to alter the format of data.

| If You Want To | For Example | Use This Instruction | Available In These Languages | See Page |
|---|---|---|---|---|
| convert the ASCII representation of an integer value to a SINT, INT, DINT, or REAL value | convert a value from a weight scale or other ASCII device to an integer so you can use it in your logic | STOD | relay ladder<br><br>structured text | 614 |
| 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<br><br>structured text | 616 |
| 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<br><br>structured text | 619 |
| 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<br><br>structured text | 621 |
| 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<br><br>structured text | 623 |
| 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<br><br>structured text | 625 |

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 | 601 |
| delete characters from a string | DELETE | 603 |
| determine the starting character of a sub-string | FIND | 605 |
| insert characters into a string | INSERT | 607 |
| extract characters from a string | MID | 609 |
| rearrange the bytes of a INT, DINT, or REAL tag | SWPB | 299 |
| compare a string to another string | CMP | 206 |
| see if the characters are equal to specific characters | EQU | 211 |
| see if the characters are not equal to specific characters | NEQ | 242 |
| see if the characters are equal to or greater than specific characters | GEQ | 215 |
| see if the characters are greater than specific characters | GRT | 219 |
| see if the characters are equal to or less than specific characters | LEQ | 223 |
| see if the characters are less than specific characters | LES | 227 |
| find a string in an array of strings | FSC | 346 |

# 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 | The LEN automatically updates to the new count of characters whenever you: <br><br> • use the String Browser dialog box to enter characters <br> • use instructions that read, convert, or manipulate a string <br><br> The LEN shows the length of the current string. The DATA member may contain additional, old characters, which are not included in the LEN count. |
| DATA | SINT array | ASCII characters of the string | • To access the characters of the string, address the name of the tag. <br> For example, to access the characters of the *string_1* tag, enter *string_1*. <br><br> • Each element of the DATA array contains one character. <br> • You can create new string data types that store less or more characters. |

# String To DINT (STOD)

The STOD instruction converts the ASCII representation of an integer to an integer or REAL value.

## Operands:

### Relay Ladder

```
┌──── STOD ────┐
│ String To DINT │
│ Source      ? │
│            ?? │
│ Dest        ? │
│            ?? │
└───────────────┘
```

| Operand | Type | Format | Enter | Notes |
|---------|------|--------|-------|-------|
| Source | string | tag | tag that contains the value in ASCII | String data types are:<br>• default STRING data type<br>• any new string data type that you create |
| Destination | SINT<br><br>INT<br><br>**DINT**<br><br>REAL | tag | tag to store the integer value | If the Source value is a floating-point number, the instruction converts only the non-fractional part of the number (regardless of the destination data type). |

    STOD(Source,Dest);

### Structured Text

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

**Description:** The STOD converts the Source to an integer and places the result in the Destination.

- The instruction converts positive and negative numbers.
- If the Source string contains non-numeric characters, the STOD converts the first set of contiguous numbers:
  - The instruction skips any initial control or non-numeric characters (except the minus sign in front of a number).
  - If the string contains multiple groups of numbers that are separated by delimiters (for example, / ), the instruction converts only the first group of numbers.

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

### Fault Conditions

| Type | Code | Cause | Recovery Method |
|---|---|---|---|
| 4 | 51 | The LEN value of the string tag is greater than the DATA size of the string tag. | 1. Check that no instruction is writing to the LEN member of the string tag.<br><br>2. In the LEN value, enter the number of characters that the string contains. |
| 4 | 53 | The output number is beyond the limits of the destination data type. | Either:<br><br>• Reduce the size of the ASCII value.<br><br>• Use a larger data type for the destination. |

### Execution:

| Condition | Relay Ladder Action | Structured Text Action |
|---|---|---|
| prescan | The rung-condition-out is set to false. | No action taken. |
| rung-condition-in is false | The rung-condition-out is set to false. | na |
| rung-condition-in is true | The instruction executes.<br><br>The rung-condition-out is set to true. | na |
| EnableIn is set | na | EnableIn is always set.<br><br>The instruction executes. |
| instruction execution | SC is set.<br><br>Destination is cleared.<br><br>The instruction converts the Source.<br><br>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

```
  MV_read.EM                                              ┌─────STOD─────────┐
──────┤ ├──────────────────────────────────────────────  │String To DINT    │
                                                          │Source      MV_msg│
                                                          │    '$06324\12\1\$r'│
                                                          │Dest   MV_msg_nmbr│
                                                          │               324│
                                                          └──────────────────┘
```

### 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

```
         ┌──── STOR ─────┐
         │ String to Real│
     ────┤ Source      ? ├────
         │            ?? │
         │ Dest        ? │
         │            ?? │
         └───────────────┘
```

| Operand | Type | Format | Enter | Notes |
|---------|------|--------|-------|-------|
| Source | string | tag | tag that contains the value in ASCII | String data types are:<br><br>• default STRING data type<br><br>• any new string data type that you create |
| Destination | REAL | tag | tag to store the REAL value | |

### Structured Text

STOR(Source,Dest);

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

**Description:**    The STOR converts the Source to a REAL value and places the result in the Destination.

• The instruction converts positive and negative numbers.

• If the Source string contains non-numeric characters, the STOR converts the first set of contiguous numbers, including the decimal point [ . ]:

– The instruction skips any initial control or non-numeric characters (except the minus sign in front of a number).

– If the string contains multiple groups of numbers that are separated by delimiters (for example, / ), the instruction converts only the first group of numbers.

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

### Fault Conditions:

| Type | Code | Cause | Recovery Method |
|------|------|-------|-----------------|
| 4 | 51 | The LEN value of the string tag is greater than the DATA size of the string tag. | 1. Check that no instruction is writing to the LEN member of the string tag.<br><br>2. In the LEN value, enter the number of characters that the string contains. |
| 4 | 53 | The output number is beyond the limits of the destination data type. | Either:<br><br>• Reduce the size of the ASCII value.<br>• Use a larger data type for the destination. |

### Execution:

| Condition | Relay Ladder Action | Structured Text Action |
|-----------|--------------------|------------------------|
| prescan | The rung-condition-out is set to false. | No action taken. |
| rung-condition-in is false | The rung-condition-out is set to false. | na |
| rung-condition-in is true | The instruction executes.<br><br>The rung-condition-out is set to true. | na |
| EnableIn is ste | na | EnableIn is always set.<br><br>The instruction executes. |
| instruction execution | S:C is set.<br><br>Destination is cleared.<br><br>The instruction converts the Source.<br><br>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.
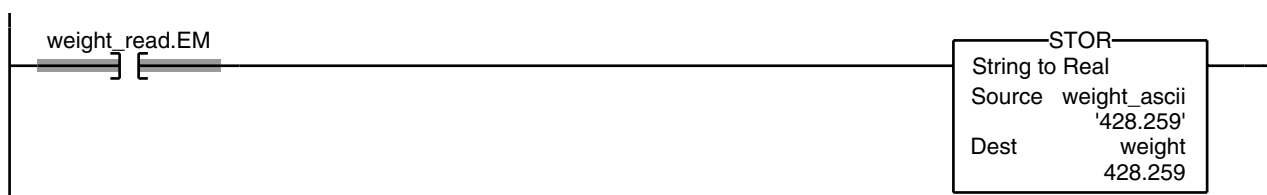
### Relay Ladder

## 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

```
        ┌── DTOS ──────┐
        │ DINT to String│
        │ Source       ?│
        │            ?? │
        │ Dest         ?│
        │            ?? │
        └───────────────┘
```
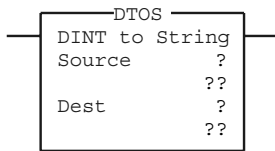
| Operand | Type | Format | Enter | Notes |
|---------|------|--------|-------|-------|
| Source | SINT | tag | tag that contains the value | If the Source is a REAL, the instruction converts it to a DINT value. Refer to REAL to an Integer on page 640. |
| | INT | | | |
| | **DINT** | | | |
| | REAL | | | |
| Destination | string | tag | tag to store the ASCII value | String data types are:<br><br>• default STRING data type<br><br>• any new string data type that you create |

### Structured Text

`DTOS(Source,Dest);`

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

**Description:** The DTOS converts the Source to a string of ASCII characters and places the result in the Destination.

**Arithmetic Status Flags:** not affected

## Fault Conditions:

| Type | Code | Cause | Recovery Method |
|------|------|-------|-----------------|
| 4 | 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.<br><br>2. In the LEN value, enter the number of characters that the string contains. |
| 4 | 52 | The output string is larger than the destination. | Create a new string data type that is large enough for the output string. Use the new string data type as the data type for the destination. |

**Execution:**

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

**Example:**    When *temp_high* is set, the DTOS instruction converts the value in *msg_num* to a string of ASCII characters and places the result in *msg_num_ascii*. Subsequent rungs insert or concatenate *msg_num_ascii* with other strings to produce a complete message for a display terminal.

**Relay Ladder**



**Structured Text**

```
IF temp_high THEN

    DTOS(msg_num,msg_num_ascii);

    temp_high := 0;

END_IF;
```

# REAL to String (RTOS)

The RTOS instruction produces the ASCII representation of a REAL value.

## Operands:

### Relay Ladder

```
         ——— RTOS ———
        Real to String
        Source       ?
                    ??
        Dest         ?
                    ??
```
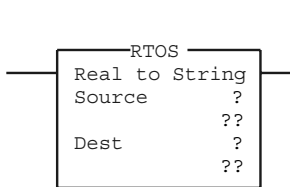
| Operand | Type | Format | Enter | Notes |
|---------|------|--------|-------|-------|
| Source | REAL | tag | tag that contains the REAL value | |
| Destination | string | tag | tag to store the ASCII value | String data types are:<br><br>• default STRING data type<br>• any new string data type that you create |

### Structured Text

`RTOS(Source,Dest);`

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

**Description:** The RTOS converts the Source to a string of ASCII characters and places the result in the Destination.

**Arithmetic Status Flags:** not affected

## Fault Conditions:

| Type | Code | Cause | Recovery Method |
|------|------|-------|-----------------|
| 4 | 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.<br><br>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.<br><br>The rung-condition-out is set to true. | na |
| EnableIn is set | na | EnableIn is always set.<br><br>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**



```
send_data                                                       ┌────RTOS────┐
  ─┤ ├─                                                          │Real to String│
                                                                │Source    data_1│
                                                                │         15.3001│
                                                                │Dest  data_1_ascii│
                                                                │      '15.3001003'│
                                                                └────────────┘
```
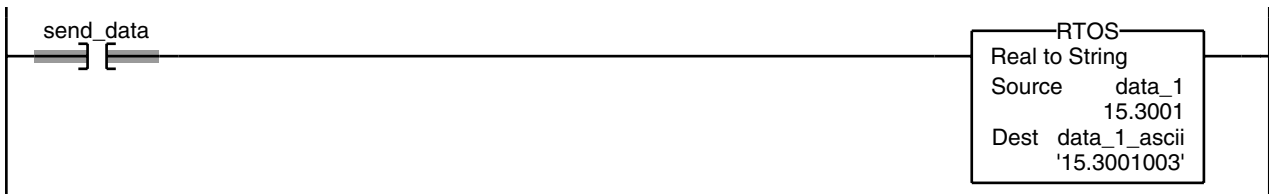
**Structured Text**

```
IF send_data THEN

    RTOS(data_1,data_1_ascii);

    send_data := 0;

END_IF;
```

# Upper Case (UPPER)

The UPPER instruction converts the alphabetical characters in a string to upper case characters.
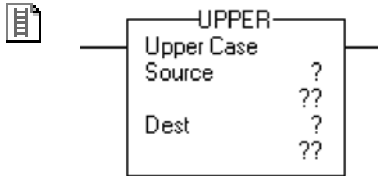
### Operands:

### Relay Ladder

| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| Source | string | tag | tag that contains the characters that you want to convert to upper case |
| Destination | string | tag | tag to store the characters in upper case |

`UPPER(Source,Dest);`

### Structured Text

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

**Description:** The UPPER instruction converts to upper case all the letters in the Source and places the result in the Destination.

- ASCII characters are case sensitive. Upper case "A" ($41) is *not* equal to lower case "a" ($61).
- If operators directly enter ASCII characters, convert the characters to all upper case or all lower case before you compare them.

Any characters in the Source string that are not letters remain unchanged.

**Arithmetic Status Flags:**  not affected

**Fault Conditions:**  none

### Execution:

| Condition | Relay Ladder Action | Structured Text Action |
|-----------|---------------------|------------------------|
| prescan | The rung-condition-out is set to false. | No action taken. |
| rung-condition-in is false | The rung-condition-out is set to false. | na |
| rung-condition-in is true | The instruction executes. The rung-condition-out is set to true. | na |
| EnableIn is set | na | EnableIn is always set. The instruction executes. |
| instruction execution | The instruction converts the Source to upper case. | |
| postscan | The rung-condition-out is set to false. | No action taken. |

**Example:**    To find information about a specific item, an operator enters the catalog number of the item into an ASCII terminal. After the controller reads the input from a terminal (*terminal_read.EM* is set), the UPPER instruction converts the characters in *catalog_number* to all upper case characters and stores the result in *catalog_number_upper_case*. A subsequent rung then searches an array for characters that match those in *catalog_number_upper_case*.

**Relay Ladder**



**Structured Text**

```
IF terminal_read.EM THEN

    UPPER(catalog_number,catalog_number_upper_case);

    terminal_read.EM := 0;

END_IF;
```

# Lower Case (LOWER)

The LOWER instruction converts the alphabetical characters in a string to lower case characters.
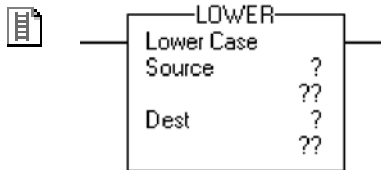
## Operands:

### Relay Ladder



| Operand | Type | Format | Description |
|---|---|---|---|
| Source | string | tag | tag that contains the characters that you want to convert to lower case |
| Destination | string | tag | tag to store the characters in lower case |

### Structured Text

```
LOWER(Source,Dest);
```

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

**Description:** The LOWER instruction converts to lower case all the letters in the Source and places the result in the Destination.

- ASCII characters are case sensitive. Upper case "A" ($41) is *not* equal to lower case "a" ($61).
- If operators directly enter ASCII characters, convert the characters to all upper case or all lower case before you compare them.

Any characters in the Source string that are not letters remain unchanged.

**Arithmetic Status Flags:** not affected

**Fault Conditions:** none

## Execution:

| Condition | Relay Ladder Action | Structured Text Action |
|---|---|---|
| prescan | The rung-condition-out is set to false. | No action taken. |
| rung-condition-in is false | The rung-condition-out is set to false. | na |
| rung-condition-in is true | The instruction executes. The rung-condition-out is set to true. | na |
| EnableIn is set | na | EnableIn is always set. The instruction executes. |
| instruction execution | The instruction converts the Source to lower case. | |
| postscan | The rung-condition-out is set to false. | No action taken. |

**Example:** To find information about a specific item, an operator enters the item number into an ASCII terminal. After the controller reads the input from a terminal (*terminal_read.EM* is set), the LOWER instruction converts the characters in *item_number* to all lower case characters and stores the result in *item_number_lower_case*. A subsequent rung then searches an array for characters that match those in *item_number_lower_case*.

### Relay Ladder

```
terminal_read.EM                                    ┌─────────LOWER─────────┐
────┤ ├─────────────────────────────────────────────│Lower Case             │
                                                     │Source       item_number│
                                                     │                  'ABCD'←│
                                                     │Dest  item_number_lower_case│
                                                     │                   'abcd'←│
                                                     └───────────────────────┘
```

### Structured Text

```
IF terminal_read.EM THEN

    LOWER(item_number,item_number_lower_case);

    terminal_read.EM := 0;

END_IF;
```

# Debug Instructions

## (BPT, TPT)

**Introduction**

Use the debug instructions to monitor the state of your logic when it is in conditions that you determine. These instructions are only compatible with RSLogix Emulate 5000 software, with which you can emulate a Logix 5000 controller on your personal computer.

| If You Want To | Use This Instruction | Available In These Languages | See Page |
|---|---|---|---|
| stop program emulation when a rung is true | BPT | relay ladder | 627 |
| log data you select when a rung is true | TPT | relay ladder | 631 |

**Breakpoints (BPT)**

Breakpoints stop program emulation when a rung is true.
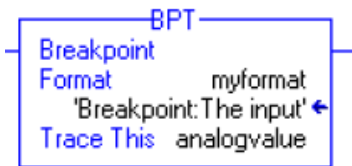
**Operands:**

**Relay Ladder**

| Operand | Type | Format | Description |
|---|---|---|---|
| Format | String | tag | A string that sets the formatting for the text that appears in the trace window for the breakpoint. |
| Trace This | BOOL, SINT, INT, DINT, REAL | tag | The tag that has a value you want to display in the trace window. |

**Description:**

Breakpoints are programmed with the Breakpoint output instruction (BPT). When the inputs on a rung containing a BPT instruction are true, the BPT instruction stops program execution. The software displays a window indicating that the breakpoint triggered and the values that triggered it.

When a breakpoint triggers, the emulator displays a window informing you that a breakpoint occurred. The title bar of the window shows the slot containing the emulator that encountered the breakpoint.

When you click **OK**, the emulator resumes program execution. If the conditions that triggered the breakpoint persist, the breakpoint will recur.

In addition, the emulator opens a trace window for the breakpoint. The trace window displays information about the breakpoint and the values.

| **ATTENTION** | When a breakpoint triggers, you will not be able to edit your project until you permit the execution to continue. You can go online with the emulator to observe the state of your project, but you will not be able to edit it. If you try to accept a rung edit while a breakpoint is triggered, you will see a dialog box saying the controller is not in the correct mode. |
|---|---|

## String Format

With the Format string in the tracepoint and breakpoint instructions, you can control how the traced tags appear in the traces or breakpoint windows. The format of the string is as shown here:

heading:(text)%(type)

where *heading* is a text string identifying the tracepoint or breakpoint, *text* is a string describing the tag (or any other text you choose), and %(*type*) indicates the format of the tag. You need one type indicator for each tag you are tracing with the tracepoint or breakpoint instruction.

For example, you could format a tracepoint string as shown here:

My tracepoint:Tag 1 = %e and Tag 2 = %d

The %e formats the first traced tag as double-precision float with an exponent, and %d formats the second traced tag as a signed decimal integer.

In this case, you would have a tracepoint instruction that has two Trace This operands (one for a REAL and one for an INT, although the value of any tag can be formatted with any flag).

The resulting tracepoint window that would appear when the tracepoint is triggered would look like this.

The slot number indicates the slot containing the emulator module that has the tracepoint or breakpoint being traced in the trace window.

The heading (the text preceding the colon in the format string) appears here.

The text for the REAL (represented in the format string as %e) appears here.

The text for the INT (represented in the format string as %d) appears here.

**Slot 1: My Tracepoint**

Tag 1 = 6.488484e+002 and Tag 2 = 443
Tag 1 = 6.617994e+002 and Tag 2 = 447

Freeze Display          Freeze Log

Clear Display

**Arithmetic Status Flags:**  not affected

**Fault Conditions:**  none

**Execution:**

| Condition: | Relay Ladder Action |
|---|---|
| prescan | The rung-condition-out is set to false. |
| rung-condition-in is false | The rung-condition-out is set to false. |
| rung-condition-in is true | The rung-condition-out is set to true. |
|  | Execution jumps to the rung that contains the LBL instruction with the referenced label name. |
| postscan | The rung-condition-out is set to false. |

**Example:**  You can display many tag values with the BPT instruction. However, the formatting string can contain only 82 characters. Because the formatting string requires two characters for each tag you want in the breakpoint, you cannot trace more than 41 tags with a single BPT instruction. However, to separate tag data in your traces, you will need to include spaces and other formatting, thus reducing the number of tag values that one BPT instruction can effectively display to far fewer than 41.

This rung shows a breakpoint that stops program execution when an analog value is greater than 3.02 or less than 2.01.

```
          ┌──GEQ───────────────┐                    ┌──BPT───────────────────┐
          │Grtr Than or Eql (A>=B)                  │Breakpoint               │
          │Source A    analogvalue                  │Format          myformat │
          │                0.0 ←                    │      'Breakpoint:The input' ←
          │Source B        3.02 │                   │Trace This   analogvalue  │
          └─────────────────────┘                   └──────────────────────────┘

          ┌──LEQ───────────────┐
          │Less Than or Eql (A<=B)
          │Source A    analogvalue
          │                0.0 ←
          │Source B        2.01 │
          └─────────────────────┘
```
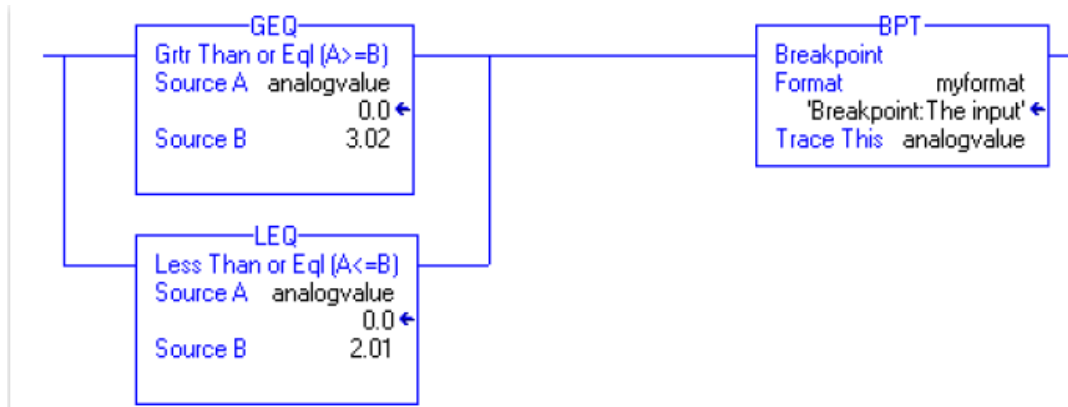
You want to display the breakpoint information in the Format string (myformat). In this case, the format string contains the following text:

```
Breakpoint:The input value is %f
```

When the breakpoint triggers, the breakpoint trace window shows the characters before the colon ("Breakpoint") in the title bar of the trace window. The other characters make up the traces. In this example, %f represents the first (and in this case, the only) tag to be traced ("analogvalue").

The resulting traces appear as shown here.

```
┌─────────────────────────────────────────────┐
│ ▓ Slot 1: Breakpoint            _ □ ✕        │
├─────────────────────────────────────────────┤
│ The input value is 3.040000                  │
│ The input value is 3.030000                  │
│ The input value is 3.122000                  │
│                                              │
│                                              │
│  ┌──────────────┐        ┌──────────────┐    │
│  │ Freeze Display│        │  Freeze Log  │    │
│  └──────────────┘        └──────────────┘    │
│  ┌──────────────┐                            │
│  │ Clear Display │                           │
│  └──────────────┘                            │
└─────────────────────────────────────────────┘
```
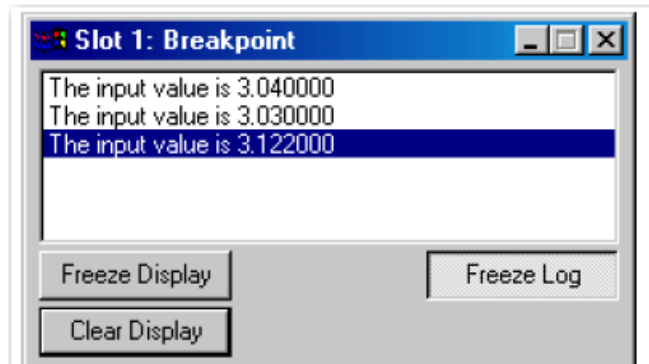
# Tracepoints (TPT)

Trace points log data you select when a rung is true.
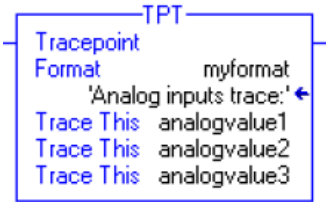
### Operands:

#### Relay Ladder

```
-----TPT-----
Tracepoint
Format         myformat
        'Analog inputs trace:' ←
Trace This   analogvalue1
Trace This   analogvalue2
Trace This   analogvalue3
```

| Operand | Type | Format | Description |
|---|---|---|---|
| Format | String | tag | A string that sets the formatting for the trace reports (both on-screen and logged to disk). |
| Trace This | BOOL, SINT, INT, DINT, REAL | tag | The tag you want to trace. |

### Description:

Tracepoints are programmed with the tracepoint output instruction (TPT). When the inputs on a rung containing a TPT instruction are true, the TPT instruction writes a trace entry to a trace display or log file.

You can trace many tags with the TPT instruction. However, the formatting string can contain only 82 characters. Because the formatting string requires two characters for each tag you want to trace, you cannot trace more than 41 tags with a single TPT instruction. However, to separate tag data in your traces, you will need to include spaces and other formatting, thus reducing the number of tags that one TPT instruction can effectively trace to far fewer than 41.

## String Format

With the Format string in the tracepoint and breakpoint instructions, you can control how the traced tags appear in the traces or breakpoint windows. The format of the string is as shown here:

heading:(text)%(type)

where *heading* is a text string identifying the tracepoint or breakpoint, *text* is a string describing the tag (or any other text you choose), and %(*type*) indicates the format of the tag. You need one type indicator for each tag you are tracing with the tracepoint or breakpoint instruction.

For example, you could format a tracepoint string as shown here:

My tracepoint:Tag 1 = %e and Tag 2 = %d

The %e formats the first traced tag as double-precision float with an exponent, and %d formats the second traced tag as a signed decimal integer.

In this case, you would have a tracepoint instruction that has two Trace This operands (one for a REAL and one for an INT, although the value of any tag can be formatted with any flag).

The resulting tracepoint window that would appear when the tracepoint is triggered would look like this.

The heading (the text preceding the colon in the format string) appears here.

The slot number indicates the slot containing the emulator module that has the tracepoint or breakpoint being traced in the trace window.

The text for the REAL (represented in the format string as %e) appears here.

The text for the INT (represented in the format string as %d) appears here.



**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.<br><br>Execution jumps to the rung that contains the LBL instruction with the referenced label name. |
| postscan | The rung-condition-out is set to false. |

**Example:**   This rung triggers a trace of three analog values when any one of them exceeds a given value (30.01).
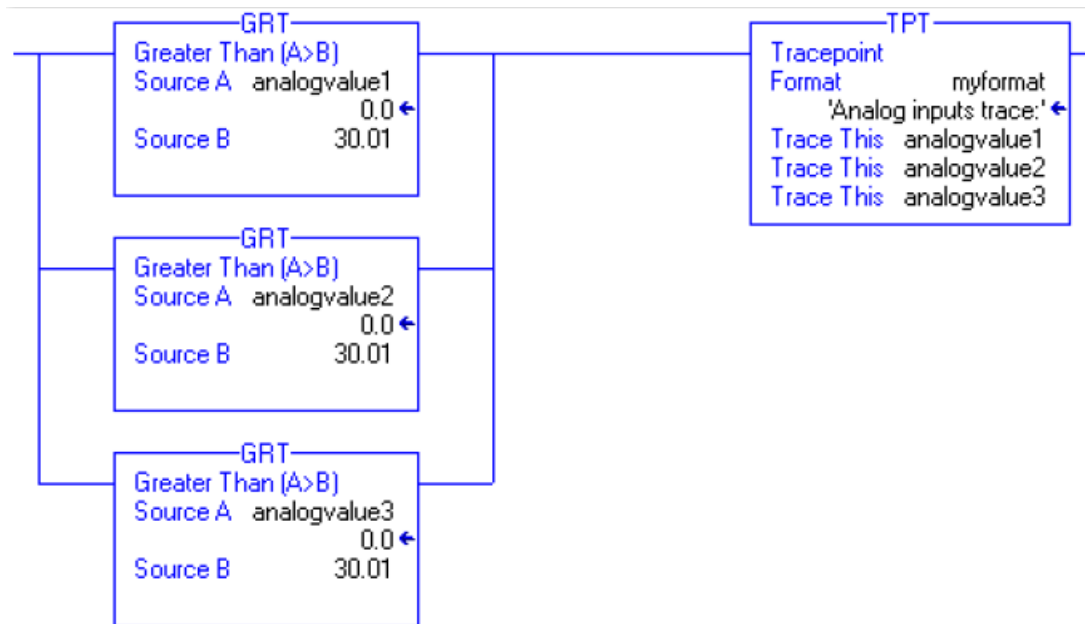


You want to display the tracepoint information in the Format string (myformat). In this case, the format string contains this text:

```
Analog inputs trace:Analog inputs = %f, %f, and %f
```

When the tracepoint triggers, the characters before the colon ("Analog inputs trace") appear in the title bar of the trace window. The other characters make up the traces. In this example, %f represents the tags to be traced ("analogvalue1," "analogvalue2," and "analogvalue3").

The resulting traces appear as shown here.
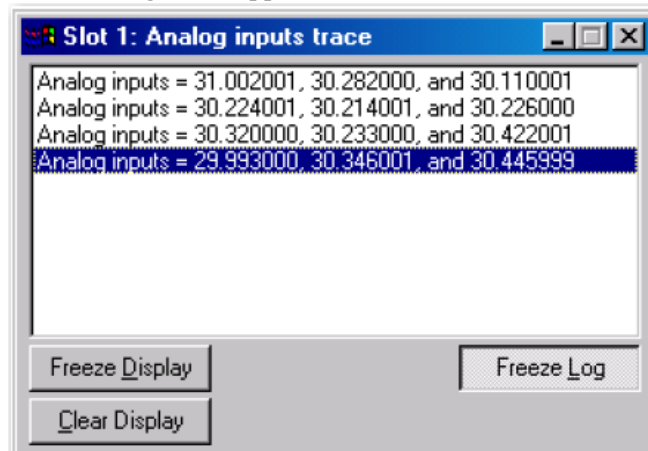


When this trace is logged to disk, the characters before the colon appear in the traces.

This indicates which tracepoint caused which trace entry. This is an example of a trace entry. "Analog inputs trace:" is the heading text from the tracepoint's format string.

```
Analog inputs trace:Analog inputs = 31.00201, 30.282000, and 30.110001
```

# Common Attributes

## Introduction

This appendix describes attributes that are common to the Logix instructions.

| 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.

```
                    EQU
        Equal
        Source A      remote_rack_1:I.Data[0]
                2#1111_1111_1111_1111
        Source B  2#1111_1111_1111_1111
```

42093

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.

```
              ┌────EQU────┐
              │ Equal                         │
              │ Source A   remote_rack_1:I.Data[0] │
              │         2#1111_1111_1111_1111 │
              │ Source B                int_0 │
              │         2#1111_1111_1111_1111 │
              └───────────┘
                                          42093
```

**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.

```
        ┌────MOV────┐              ┌────EQU────┐
        │ Move               │              │ Equal                         │
        │ Source  2#1111_1111_1111_1111 │    │ Source A   remote_rack_1:I.Data[0] │
        │                    │              │         2#1111_1111_1111_1111 │
        │ Dest          int_0 │              │ Source B                int_0 │
        │       2#1111_1111_1111_1111 │      │         2#1111_1111_1111_1111 │
        └───────────┘              └───────────┘
                                                           42093
```

## Integer to REAL

The controller stores REAL values in IEEE single-precision, floating-point number format. It uses one bit for the sign of the value, 23 bits for the base value, and eight bits for the exponent (32 bits total). If you mix an integer tag (SINT, INT, or DINT) and a REAL tag as inputs in the same instruction, the controller converts the integer value to a REAL value before the instruction executes.

- A SINT or INT value always converts to the same REAL value.

- A DINT value may not convert to the same REAL value:

  - A REAL value uses up to 24 bits for the base value (23 stored bits plus a "hidden" bit).
  - A DINT value uses up to 32 bits for the value (one for the sign and 31 for the value).
  - If the DINT value requires more than 24 significant bits, it *may not* convert to the same REAL value. If it will not, the controller rounds to the nearest REAL value using 24 significant bits.

## DINT to SINT or INT

To convert a DINT value to a SINT or INT value, the controller truncates the upper portion of the DINT and sets the overflow status flag, if necessary. The following example shows the result of a DINT to SINT or INT conversion.

---

| EXAMPLE | Conversion of a DINT to an INT and a SINT |
|---|---|

| This DINT Value | Converts To This Smaller Value | |
|---|---|---|
| 16#0001_0081 (65,665) | INT: | 16#0081 (129) |
| | SINT: | 16#81 (-127) |

## REAL to an Integer

To convert a REAL value to an integer value, the controller rounds the fractional part and truncates the upper portion of the non-fractional part. If data is lost, the controller sets the overflow status flag. Numbers round as follows:

- Numbers other than *x*.5 round to the nearest whole number.
- *X*.5 rounds to the nearest even number.

The following example show the result of converting REAL values to DINT values.

---

**EXAMPLE**   Conversion of REAL values to DINT values

| This REAL Value | Converts To This DINT Value |
| --- | --- |
| -2.5 | -2 |
| -1.6 | -2 |
| -1.5 | -2 |
| -1.4 | -1 |
| 1.4 | 1 |
| 1.5 | 2 |
| 1.6 | 2 |
| 2.5 | 2 |

---

**IMPORTANT**   The arithmetic status flags are set based on the value being stored. Instructions that normally do not affect arithmetic status keywords might appear to do so if type conversion occurs because of mixed data types for the instruction parameters. The type conversion process sets the arithmetic status keywords.

# Function Block Attributes

## Introduction

This appendix describes issues that are unique with function block instructions. Review the information in this appendix to make sure you understand how your function block routines will operate.

| **IMPORTANT** | When programming in function block, restrict the range of engineering units to $+/-10^{+/-15}$ 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 $(+/-10^{+/-38})$. |
|---|---|

## Choose the Function Block Elements

To control a device, use the following elements:
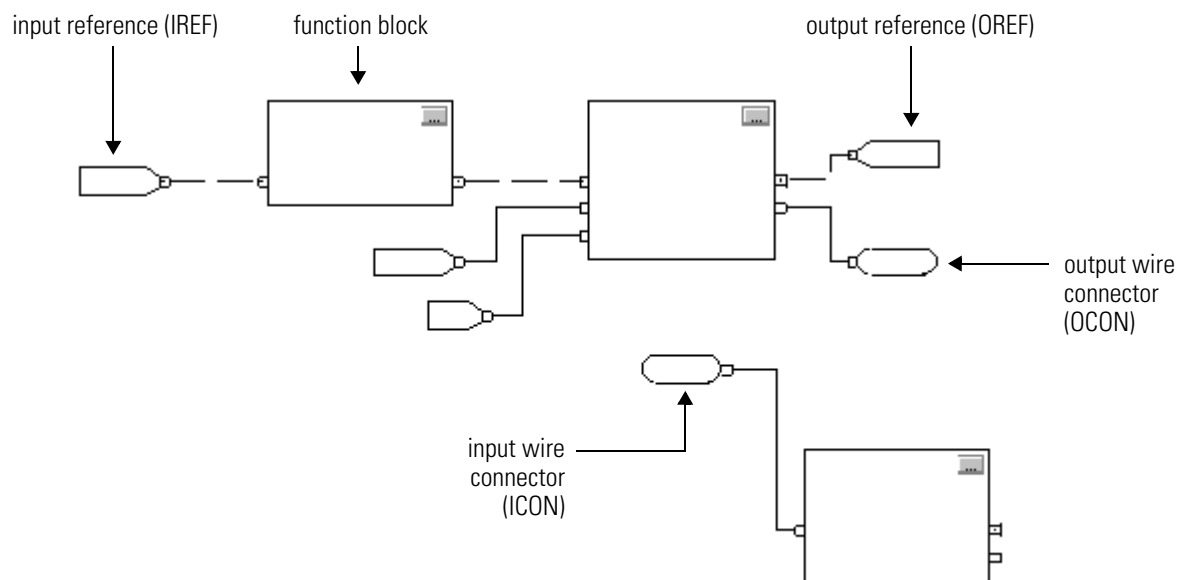


Use the following table to choose your function block elements:z

| 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) |

| If You Want To | Use a |
|---|---|
| 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:<br><br>• far apart on the same sheet<br>• on different sheets within the same routine | output wire connector (OCON) and an input wire connector (ICON) |
| disperse data to several points in the routine | single output wire connector (OCON) and multiple input wire connectors (ICON) |

## Latching Data

If you use an IREF to specify input data for a function block instruction, the data in that IREF is latched for the scan of the function block routine. The IREF latches data from program-scoped and controller-scoped tags. The controller updates all IREF data at the beginning of each scan.
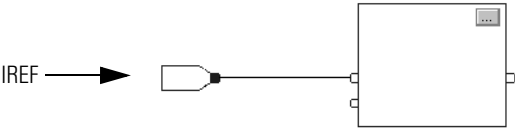


In this example, the value of tagA is stored at the beginning of the routine's execution. The stored value is used when Block_01 executes. The same stored value is also used when Blcock_02 executes. If the value of tagA changes during execution of the routine, the stored value of tagA in the IREF does not change until the next execution of the routine.
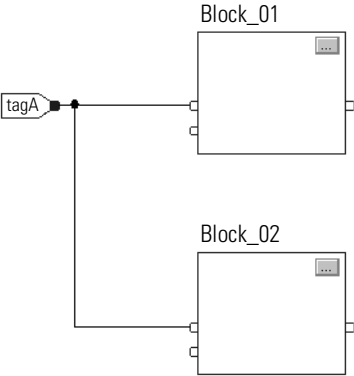
This example is the same as the one above. The value of tagA is stored only once at the beginning of the routine's execution. The routine uses this stored value throughout the routine.

Block_01

tagA

Block_02

tagA
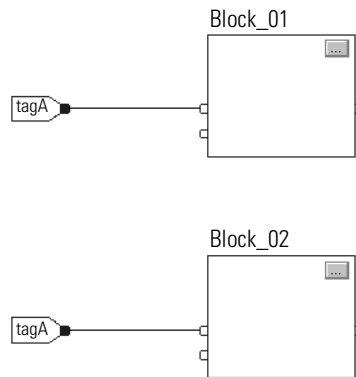
Starting with RSLogix 5000 software, version 11, you can use the same tag in multiple IREFs and an OREF in the same routine. Because the values of tags in IREFs are latched every scan through the routine, all IREFs will use the same value, even if an OREF obtains a different tag value during execution 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.

Block_01

tagA          tagA

Block_02

tagA
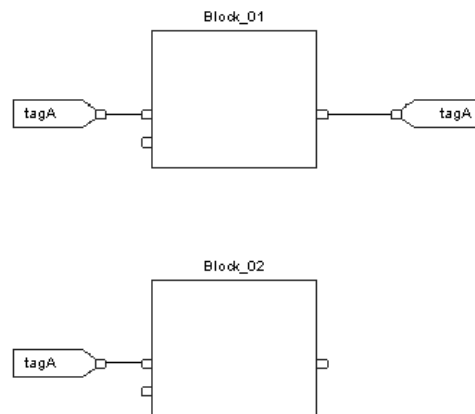
## Order of Execution

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.
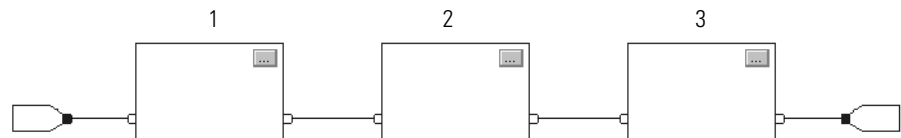


Execution order is only relative to the blocks that are wired together. The following example is fine because the two groups of blocks are not wired together. The blocks within a specific group execute in the appropriate order in relation to the blocks in that group.
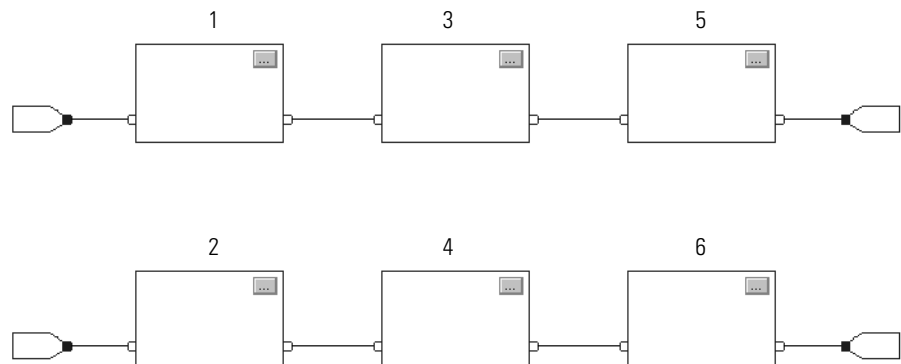
## Resolve a Loop

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.

This input pin uses an output that the block produced on the previous scan.
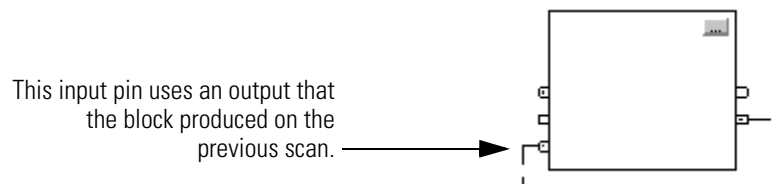
If a group of blocks are in a loop, the controller cannot determine which block to execute first. In other words, it cannot resolve the loop.
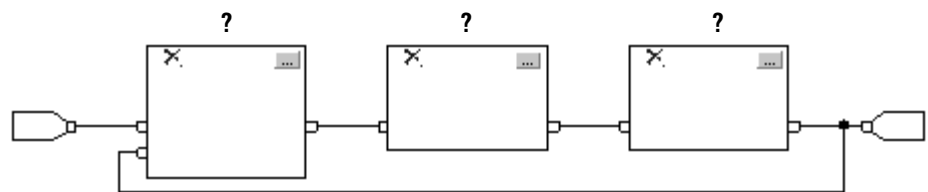
To identify which block to execute first, mark the input wire that creates the loop (the feedback wire) with the *Assume Data Available* indicator. In the following example, block 1 uses the output from block 3 that was produced in the previous execution of the routine.

This input pin uses the output that block 3 produced on the previous scan.

Assume Data Available indicator
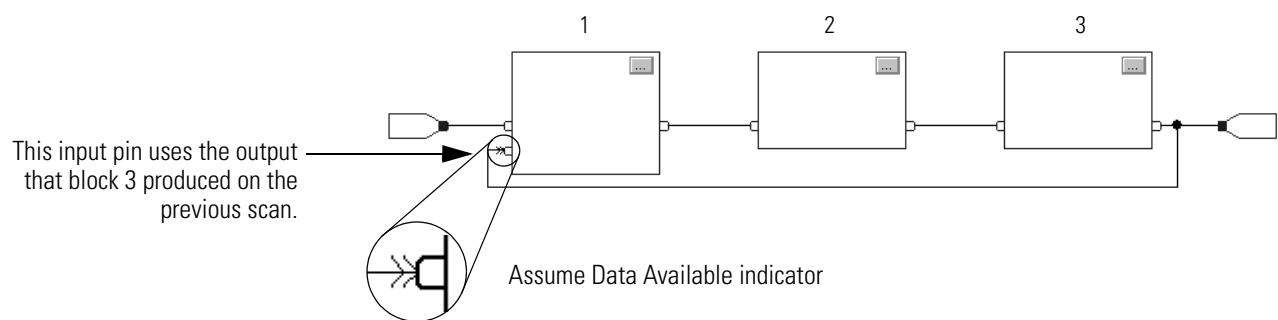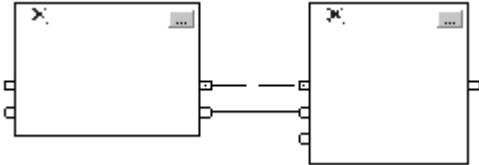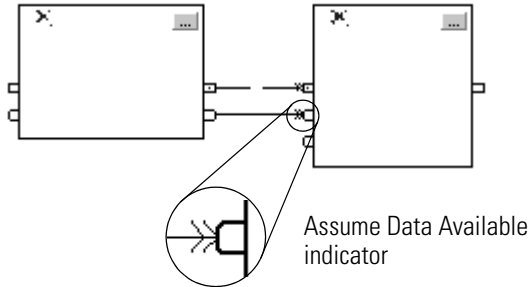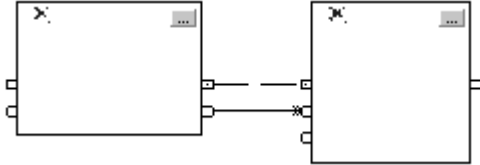
The *Assume Data Available* indicator defines the data flow within the loop. The arrow indicates that the data serves as input to the first block in the loop.

*Do not* mark all the wires of a loop with the *Assume Data Available* indicator.

| This is OK | This is NOT OK |
|---|---|
|  Assume Data Available indicator<br><br>The *Assume Data Available* indicator defines the data flow within the loop. | <br><br>The controller cannot resolve the loop because all the wires use the *Assume Data Available* indicator. |

## Resolve Data Flow Between Two Blocks

If you use two or more wires to connect two blocks, use the same data flow indicators for all of the wires between the two blocks.

| This is OK | This is NOT OK |
|---|---|
|  |  |
| Neither wire uses the *Assume Data Available* indicator. | One wire uses the *Assume Data Available* indicator while the other wire does not. |
|  | |
| Both wires use the *Assume Data Available* indicator. | |

## Create a One Scan Delay

To produce a one scan delay between blocks, use the *Assume Data Available* indicator. In the following example, block 1 executes first. It uses the output from block 2 that was produced in the previous scan of the routine.
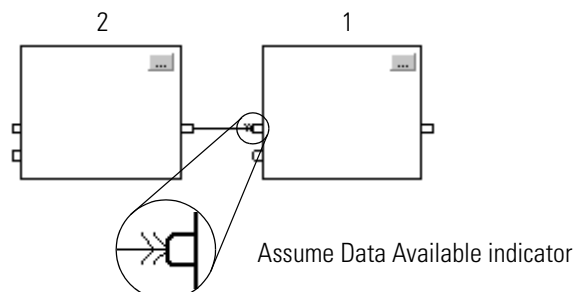
## Summary

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 ±NAN, or ±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 ±NAN or ±INF. If ±NAN or ±INF, the block outputs ±NAN or ±INF. | Blocks with output limiting execute their algorithm and check the result for ±NAN or ±INF. The output limits are defined by the HighLimit and LowLimit input parameters. If ±INF, the block outputs a limited result. If ±NAN, the output limits are not used and the block outputs ±NAN. | The overflow condition does not apply. These instructions typically have a boolean output. |
| ALMNTCH | HLL | BANDOSRI |
| DEDTPMUL | INTG | BNOTRESD |
| DERVPOSP | PI | BORRTOR |
| ESELRLIM | PIDE | BXORSETD |
| FGENRMPS | SCL | CUTDTOFR |
| HPFSCRV | SOC | D2SDTONR |
| LDL2SEL | | D3SD |
| LDLGSNEG | | DFF |
| LPFSRTP | | JKFF |
| MAVESSUM | | OSFI |
| MAXCTOT | | |
| MINCUPDN | | |
| 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 | Periodic mode is the default mode and is suitable for most control applications. We recommend that you place the instructions that use this mode in a routine that executes in a periodic task. The delta time (DeltaT) for the instruction is determined as follows: |
| | **If The Instruction Executes In a**     **Then DeltaT Equals** |
| | periodic task     period of the task |
| | event or continuous task     elapsed time since the previous execution |
| | The controller truncates the elapsed time to whole milliseconds (ms). For example, if the elapsed time = 10.5 ms, the controller sets DeltaT = 10 ms. |
| | The update of the process input needs to be synchronized with the execution of the task or sampled 5-10 times faster than the task executes in order to minimize the sampling error between the input and the instruction. |
| oversample | In oversample mode, the delta time (DeltaT) used by the instruction is the value written into the OversampleDT parameter of the instruction. If the process input has a time stamp value, use the real time sampling mode instead. |
| | Add logic to your program to control when the instruction executes. For example, you can use a timer set to the OversampleDeltaT value to control the execution by using the EnableIn input of the instruction. |
| | The process input needs to be sampled 5-10 times faster than the instruction is executed in order to minimize the sampling error between the input and the instruction. |

| Timing Mode | Description |
|---|---|
| real time sampling | In the real time sampling mode, the delta time (DeltaT) used by the instruction is the difference between two time stamp values that correspond to the updates of the process input. Use this mode when the process input has a time stamp associated with its updates and you need precise coordination. |
| | The time stamp value is read from the tag name entered for the RTSTimeStamp parameter of the instruction. Normally this tag name is a parameter on the input module associated with the process input. |
| | The instruction compares the configured RTSTime value (expected update period) against the calculated DeltaT to determine if every update of the process input is being read by the instruction. If DeltaT is not within 1 millisecond of the configuration time, the instruction sets the RTSMissed status bit to indicate that a problem exists reading updates for the input on the module. |

Time-based instructions require a constant value for DeltaT in order for the control algorithm to properly calculate the process output. If DeltaT varies, a discontinuity occurs in the process output. The severity of the discontinuity depends on the instruction and range over which DeltaT varies. A discontinuity occurs if the:

- instruction is not executed during a scan.
- instruction is executed multiple times during a task.
- task is running and the task scan rate or the sample time of the process input changes.
- user changes the time base mode while the task is running.
- Order parameter is changed on a filter block while the task is running. Changing the Order parameter selects a different control algorithm within the instruction.

# Common Instruction Parameters for Timing Modes

The instructions that support time base modes have these input and output parameters:

*Input parameters*

| Input Parameter | Data Type | Description |
|---|---|---|
| TimingMode | DINT | Selects timing execution mode. |
| | | **Value:** **Description:** |
| | | 0    periodic mode |
| | | 1    oversample mode |
| | | 2    real time sampling mode |
| | | valid = 0 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.<br><br>valid = 1 to 32,767ms<br><br>default = 1 |
| RTSTimeStamp | DINT | Module time stamp value for real time sampling timing. The time stamp value that corresponds to the last update of the input signal. This value is used to calculate DeltaT. If invalid, the instruction sets the appropriate bit in Status, disables execution of the control algorithm, and disables RTSMissed checking.<br><br>valid =1 to 32,767ms (wraps from 32767 to 0)<br><br>1 count = 1 millisecond<br><br>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.<br><br>Periodic: DeltaT = task scan rate if task is Periodic task, DeltaT = elapsed time since previous instruction execution if task is Event or Continuous task<br><br>Oversample: DeltaT = OversampleDT<br><br>Real Time Sampling: DeltaT = $(RTSTimeStamp_n - RTSTimeStamp_{n-1})$ |
| Status | DINT | Status of the function block. |
| TimingModeInv (Status.27) | BOOL | Invalid TimingMode value. |
| RTSMissed (Status.28) | BOOL | Only used in real time sampling mode. Set when ABS \| DeltaT - RTSTime \| > 1 (.001 second). |
| RTSTimeInv (Status.29) | BOOL | Invalid RTSTime value. |
| RTSTimeStampInv (Status.30) | BOOL | Invalid RTSTimeStamp value. |
| DeltaTInv (Status.31) | BOOL | Invalid DeltaT value. |

## Overview of Timing Modes

The following diagram shows how an instruction determines the appropriate timing mode.

```
                          ↓
              ┌─────────────────────────┐
              │ Determine time base mode │
              └─────────────────────────┘
```

TimingMode = 0          TimingMode = 1          TimingMode = 2

( Periodic timing )     ( Oversample timing )     ( Real time timing )

DeltaT = OversampleDT

If DeltaT < 0 or DeltaT > 4194.303 secs. the instruction sets DeltaT = 0.0 and sets the appropriate bit in Status.

$DeltaT = RTSTimeStamp_n - RTSTimeStamp_{n-1}$

If DeltaT > 0, the instruction executes.

┌─────────────────────┐
│ Determine task type │
└─────────────────────┘

( Periodic task )       ( Event or Continuous task )

DeltaT = task scan time

DeltaT = elapsed time since last execution

## 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.



Wiring a "1" into ProgOperReq means the user program always wants the TOT to be in Operator control

Because the ProgOperReq input is always set, pressing the "Program" button on the faceplate (which sets the OperProgReq input) has no effect. Normally, setting OperProgReq puts the TOT in Program control.
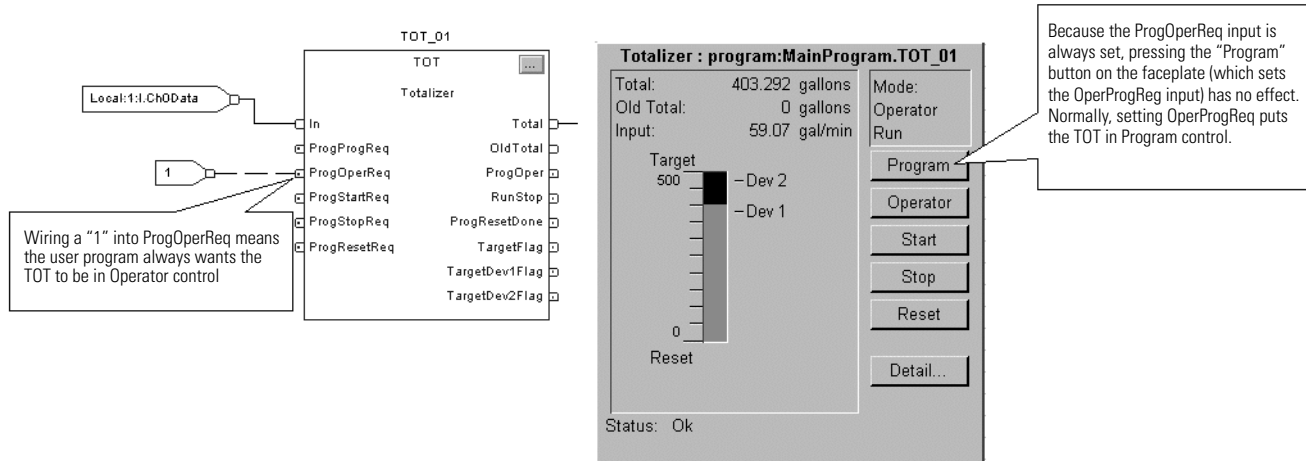
Likewise, constantly setting the ProgProgReq can "lock" the instruction into Program control. This is useful for automatic startup sequences when you want the program to control the action of the instruction without worrying about an operator inadvertently taking control of the instruction. In this example, 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.



When StartupSequenceActive is set, the PIDE instruction is placed in Program control and Manual mode. The StartupCV value is used as the loop output.
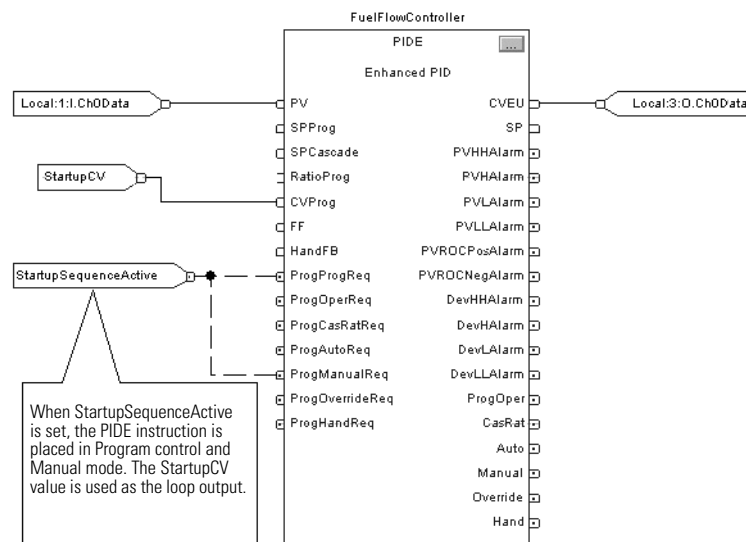
Operator request inputs to an instruction are always cleared by the instruction when it executes. This allows operator interfaces to work with these instructions by merely setting the desired mode request bit. You don't have to program the operator interface to reset the request bits. For example, if 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.

```
        TIC101AutoReqPB    TIC101AutoReqPBOneShot                                    TIC101.ProgAutoReq
           ] [                      [ONS]                                                  <L>
```

**Notes:**

# Structured Text Programming

## Introduction

This appendix describes issues that are unique with structured text programming. Review the information in this appendix to make sure you understand how your structured text programming will execute.

| For Information About | See Page |
|---|---|
| Structured Text Syntax | 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<br><br>(see page 661) | Use an assignment statement to assign values to tags.<br><br>The := operator is the assignment operator.<br><br>Terminate the assignment with a semi colon ";". | *tag := expression;* |

| Term | Definition | | Examples |
|------|-----------|---|----------|
| expression<br><br>(see [page 663](#)) | An expression is part of a complete assignment or construct statement.<br>An expression evaluates to a number (numerical expression) or to a true or false state (BOOL expression). | | |
| | An expression contains: | | |
| | tags | A named area of the memory where data is stored (BOOL, SINT,INT,DINT, REAL, string). | *value1* |
| | immediates | A constant value. | *4* |
| | operators | A symbol or mnemonic that specifies an operation within an expression. | *tag1 + tag2*<br><br>*tag1 >= value1* |
| | functions | When executed, a function yields one value. Use parentheses to contain the operand of a function.<br><br>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. | *function(tag1)* |
| instruction<br><br>(see page 670) | An instruction is a standalone statement. | | *instruction();* |
| | An instruction uses parenthesis to contain its operands. | | |
| | Depending on the instruction, there can be zero, one, or multiple operands. | | *instruction(operand);* |
| | When executed, an instruction yields one or more values that are part of a data structure.<br><br>Terminate the instruction with a semi colon ";". | | *instruction(operand1, operand2,operand3);* |
| | 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. | | |

| Term | Definition | Examples |
|------|-----------|----------|
| construct<br><br>(see page 671) | A conditional statement used to trigger structured text code (i.e, other statements).<br><br>Terminate the construct with a semi colon ";". | `IF...THEN`<br><br>`CASE`<br><br>`FOR...DO`<br><br>`WHILE...DO`<br><br>`REPEAT...UNTIL`<br><br>`EXIT` |
| comment<br><br>(see page 687) | Text that explains or clarifies what a section of structured text does.<br><br>• Use comments to make it easier to interpret the structured text.<br>• Comments do not affect the execution of the structured text.<br>• Comments can appear anywhere in structured text. | `//comment`<br><br>`(*start of comment . . . end of comment*)`<br><br>`/*start of comment . . . end of comment*/` |

## Assignments

Use an assignment to change the value stored within a tag. An assignment has this syntax:

*tag := expression;*

where:

| Component | Description | | |
|-----------|-------------|---|---|
| *tag* | represents the tag that is getting the new value | | |
| | the tag must be a BOOL, SINT, INT, DINT, or REAL | | |
| := | is the assignment symbol | | |
| *expression* | represents the new value to assign to the tag | | |
| | **If *tag* is this data type:** | **Use this type of expression:** | |
| | BOOL | BOOL expression | |
| | SINT | 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 | |
|---|---|---|
| *tag* | represents the tag that is getting the new value | |
| | the tag must be a BOOL, SINT, INT, DINT, or REAL | |
| [:=] | is the non-retentive assignment symbol | |
| *expression* | represents the new value to assign to the tag | |
| | **If `tag` is this data type:** | **Use this type of expression:** |
| | BOOL | BOOL expression |
| | SINT | 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 *not* OK. |
| --- | --- |
| `string1.DATA[0]:= 65;` | `string1.DATA[0] := A;` |
| `string1.DATA[0]:= string2.DATA[0];` | `string1 := string2;` |

To add or insert a string of characters to a string tag, use either of these ASCII string instructions:

| To | Use This Instruction |
| --- | --- |
| add characters to the end of a string | CONCAT |
| insert characters into a string | INSERT |

**Expressions**

An expression is a tag name, equation, or comparison. To write an expression, use any of the following:

- tag name that stores the value (variable)
- number that you enter directly into the expression (immediate value)
- functions, such as: ABS, TRUNC
- operators, such as: +, -, <, >, And, Or

As you write expressions, follow these general rules:

- Use any combination of upper-case and lower-case letter. For example, these three variations of "AND" are acceptable: AND, And, and.
- For more complex requirements, use parentheses to group expressions within expressions. This makes the whole expression easier to read and ensures that the expression executes in the desired sequence. See "<span>Determine the order of execution</span>"on <span>page 669</span>.

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 (*numeric_expression*) | DINT, REAL |
| arc cosine | ACOS (*numeric_expression*) | REAL |
| arc sine | ASIN (*numeric_expression*) | REAL |
| arc tangent | ATAN (*numeric_expression*) | REAL |
| cosine | COS (*numeric_expression*) | REAL |
| radians to degrees | DEG (*numeric_expression*) | DINT, REAL |
| natural log | LN (*numeric_expression*) | REAL |
| log base 10 | LOG (*numeric_expression*) | REAL |
| degrees to radians | RAD (*numeric_expression*) | DINT, REAL |
| sine | SIN (*numeric_expression*) | REAL |
| square root | SQRT (*numeric_expression*) | DINT, REAL |
| tangent | TAN (*numeric_expression*) | REAL |
| truncate | TRUNC (*numeric_expression*) | DINT, REAL |

For example:

| Use This Format | Example | |
|---|---|---|
| | **For This Situation** | **You'd Write** |
| `value1 operator value2` | If *gain_4* and *gain_4_adj* are DINT tags and your specification says: "Add 15 to *gain_4* and store the result in *gain_4_adj*." | `gain_4_adj := gain_4+15;` |
| `operator value1` | If alarm and *high_alarm* are DINT tags and your specification says: "Negate *high_alarm* and store the result in *alarm*." | `alarm:= -high_alarm;` |
| `function(numeric_expression)` | If *overtravel* and *overtravel_POS* are DINT tags and your specification says: "Calculate the absolute value of *overtravel* and store the result in *overtravel_POS*." | `overtravel_POS := ABS(overtravel);` |
| `value1 operator (function((value2+value3)/2)` | If *adjustment* and *position* are DINT tags and *sensor1* and *sensor2* are REAL tags and your specification says: "Find the absolute value of the average of *sensor1* and *sensor2*, add the *adjustment*, and store the result in *position*." | `position := adjustment + ABS((sensor1 + sensor2)/2);` |

## Use relational operators

Relational operators compare two values or strings to provide a true or false result. The result of a relational operation is a BOOL value:

| If The Comparison Is | The Result Is |
|---|---|
| true | 1 |
| false | 0 |

Use 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** |
| `value1 operator value2` | If *temp* is a DINT tag and your specification says: "If *temp* is less than 100° then…" | `IF temp<100 THEN...` |
| `stringtag1 operator stringtag2` | If *bar_code* and *dest* are string tags and your specification says: "If *bar_code* equals *dest* then…" | `IF bar_code=dest THEN...` |
| `char1 operator char2`<br><br>To enter an ASCII character directly into the expression, enter the decimal value of the character. | If *bar_code* is a string tag and your specification says: "If *bar_code.DATA[0]* equals 'A' then…" | `IF bar_code.DATA[0]=65 THEN...` |
| `bool_tag := bool_expressions` | If *count* and *length* are DINT tags, *done* is a BOOL tag, and your specification says "If *count* is greater than or equal to *length*, you are done counting." | `done := (count >= length);` |

### How Strings Are Evaluated

The hexadecimal values of the ASCII characters determine if one string is less than or greater than another string.

- When the two strings are sorted as in a telephone directory, the order of the strings determines which one is greater.



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

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** |
| *BOOLtag* | If *photoeye* is a BOOL tag and your specification says: "If *photoeye* is on then…" | `IF photoeye THEN...` |
| NOT *BOOLtag* | If *photoeye* is a BOOL tag and your specification says: "If *photoeye* is off then…" | `IF NOT photoeye THEN...` |
| *expression1* & *expression2* | If *photoeye* is a BOOL tag, *temp* is a DINT tag, and your specification says: "If *photoeye* is on and *temp* is less than 100° then…". | `IF photoeye & (temp<100) THEN...` |
| *expression1* OR *expression2* | If *photoeye* is a BOOL tag, *temp* is a DINT tag, and your specification says: "If *photoeye* is on or *temp* is less than 100° then…". | `IF photoeye OR (temp<100) THEN...` |
| *expression1* XOR *expression2* | If *photoeye1* and *photoeye2* are BOOL tags and your specification says: "If: <br><br>• *photoeye1* is on while *photoeye2* is off or <br>• *photoeye1* is off while *photoeye2* is on <br><br>then…" | `IF photoeye1 XOR photoeye2 THEN...` |
| *BOOLtag := expression1 & expression2* | If *photoeye1* and *photoeye2* are BOOL tags, *open* is a BOOL tag, and your specification says: "If *photoeye*1 and *photoeye2* are both on, set *open* to true". | `open := photoeye1 & photoeye2;` |

## Use bitwise operators

Bitwise operators manipulate the bits within a value based on two values.

| For | Use This Operator | Optimal Data Type |
|-----|-------------------|-------------------|
| bitwise AND | &, AND | DINT |
| bitwise OR | OR | DINT |
| bitwise exclusive OR | XOR | DINT |
| bitwise complement | NOT | DINT |

For example:

| Use This Format | Example | |
|-----------------|---------|---|
| | **For This Situation** | **You'd Write** |
| `value1 operator value2` | If *input1*, *input2*, and *result1* are DINT tags and your specification says: "Calculate the bitwise result of *input1* and *input2*. Store the result in *result1*." | `result1 := input1 AND input2;` |

## Determine the order of execution

The operations you write into an expression are performed in a prescribed order, not necessarily from left to right.

- Operations of equal order are performed from left to right.
- If an expression contains multiple operators or functions, group the conditions in parenthesis "( )" . This ensures the correct order of execution and makes it easier to read the expression.

| Order | Operation |
|-------|-----------|
| 1. | ( ) |
| 2. | function (…) |
| 3. | ** |
| 4. | −(negate) |
| 5. | NOT |
| 6. | *, /, MOD |
| 7. | +, - (subtract) |
| 8. | <, <=, >, >= |
| 9. | =,  <> |
| 10. | &, AND |
| 11. | XOR |
| 12. | OR |

## Instructions

Structured text statements can also be instructions. See the Locator Table 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 | BOOL tag or expression that evaluates to a BOOL value (BOOL expression) |
| | | expression | |

**Description:** The syntax is:

```
IF bool_expression1 THEN
        <statement >;                    statements to execute when
                                         bool_expression1 is true
            .
            .
            .
ELSIF bool_expression2 THEN
        <statement>;                     statements to execute when
                                         bool_expression2 is true
            .
            .
            .
ELSE
        <statement>;                     statements to execute when
                                         both expressions are false
            .
            .
            .
END_IF;
```

optional → ELSIF ... block

optional → ELSE ... block

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<br><br>    conveyor = off (0)<br><br>    alarm = on (1) | `IF rejects > 3 THEN`<br><br>`    conveyor := 0;`<br><br>`    alarm := 1;`<br><br>`END_IF;` |

**Example 2:**   **IF…THEN…ELSE**

| If You Want This | Enter This Structured Text |
|---|---|
| If conveyor direction contact = forward (1) then<br><br>    light = off<br><br>Otherwise light = on | `IF conveyor_direction THEN`<br><br>`    light := 0;`<br><br>`ELSE`<br><br>`    light [:=] 1;`<br><br>`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 | `IF Sugar.Low & Sugar.High THEN` |
| inlet valve = open (on) | `Sugar.Inlet [:=] 1;` |
| Until sugar high limit switch = high (off) | `ELSIF NOT(Sugar.High) THEN` |
| | `Sugar.Inlet := 0;` |
| | `END_IF;` |

The [:=] tells the controller to clear *Sugar.Inlet* whenever the controller:

- enters the RUN mode
- leaves the step of an SFC if you configure the SFC for *Automatic reset* (This applies only if you embed the assignment in the action of the step or use the action to call a structured text routine via a JSR instruction.)

**Example 4:** **IF…THEN…ELSIF…ELSE**

| If You Want This | Enter This Structured Text |
|---|---|
| If tank temperature > 100 | `IF tank.temp > 200 THEN` |
| then pump = slow | `pump.fast :=1; pump.slow :=0; pump.off :=0;` |
| If tank temperature > 200 | `ELSIF tank.temp > 100 THEN` |
| then pump = fast | `pump.fast :=0; pump.slow :=1; pump.off :=0;` |
| otherwise pump = off | `ELSE` |
| | `pump.fast :=0; pump.slow :=0; pump.off :=1;` |
| | `END_IF;` |

# CASE...OF

Use CASE to select what to do based on a numerical value.

## Operands:

CASE *numeric_expression* OF

    *selector1*: *statement*;

    *selectorN*: *statement*;

ELSE

    *statement*;

END_CASE;

### Structured Text

| Operand | Type | Format | Enter |
|---------|------|--------|-------|
| *numeric_* | SINT | tag | tag or expression that evaluates to a number (numeric expression) |
| *expression* | INT | expression | |
| | DINT | | |
| | REAL | | |
| *selector* | SINT | immediate | same type as *numeric_expression* |
| | 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

```
ELSE

                        <statement>;
                                .
                                .
                                .
```

statements to execute when
*numeric_expression* ≠ any
selector

optional

```
END_CASE;
```
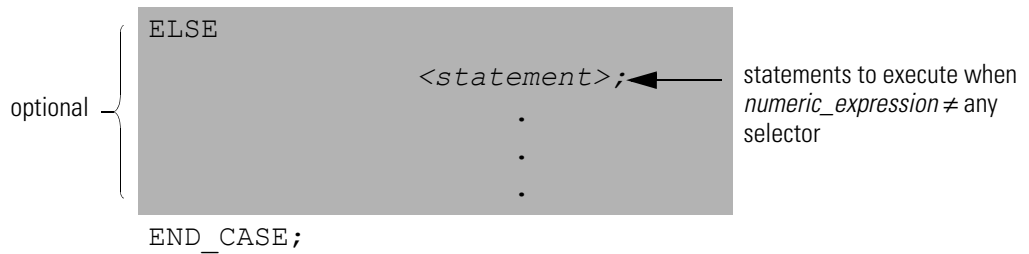
See the table on the next page for valid selector values.

The syntax for entering the selector values is:

| When Selector Is | Enter |
| --- | --- |
| one value | *value*: *statement* |
| multiple, distinct values | *value1, value2, valueN* : *<statement>*<br><br><br>Use a comma (,) to separate each value. |
| a range of values | *value1..valueN* : *<statement>*<br><br><br>Use two periods (..) to identify the range. |
| distinct values plus a range of values | *valuea, valueb, value1..valueN* : *<statement>* |

The CASE construct is similar to a switch statement in the C or C++
programming languages. However, with the CASE construct the controller
executes *only* the statements that are associated with the *first matching* selector
value. Execution *always breaks after the statements of that selector* and goes to the
END_CASE statement.

**Arithmetic Status Flags:**     not affected

**Fault Conditions:**     none

**Example**

| If You Want This | Enter This Structured Text |
|---|---|
| If recipe number = 1 then | `CASE recipe_number OF` |
| Ingredient A outlet 1 = open (1) | `1:        Ingredient_A.Outlet_1 :=1;` |
| Ingredient B outlet 4 = open (1) | `          Ingredient_B.Outlet_4 :=1;` |
| If recipe number = 2 or 3 then | `2,3:      Ingredient_A.Outlet_4 :=1;` |
| Ingredient A outlet 4 = open (1) | `          Ingredient_B.Outlet_2 :=1;` |
| Ingredient B outlet 2 = open (1) | |
| If recipe number = 4, 5, 6, or 7 then | `4..7:     Ingredient_A.Outlet_4 :=1;` |
| Ingredient A outlet 4 = open (1) | `          Ingredient_B.Outlet_2 :=1;` |
| Ingredient B outlet 2 = open (1) | |
| If recipe number = 8, 11, 12, or 13 then | `8,11..13  Ingredient_A.Outlet_1 :=1;` |
| Ingredient A outlet 1 = open (1) | `          Ingredient_B.Outlet_4 :=1;` |
| Ingredient B outlet 4 = open (1) | |
| Otherwise all outlets = closed (0) | `ELSE` |
| | `    Ingredient_A.Outlet_1 [:=]0;` |
| | `    Ingredient_A.Outlet_4 [:=]0;` |
| | `    Ingredient_B.Outlet_2 [:=]0;` |
| | `    Ingredient_B.Outlet_4 [:=]0;` |
| | `END_CASE;` |

The [:=] tells the controller to also clear the outlet tags whenever the controller:

- enters the RUN mode
- leaves the step of an SFC if you configure the SFC for *Automatic reset* (This applies only if you embed the assignment in the action of the step or use the action to call a structured text routine via a JSR instruction.)

## FOR…DO

Use the FOR…DO loop to do something a specific number of times before doing anything else.

### Operands:



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

    <statement>;

END_FOR;
```

### Structured Text

| Operand | Type | Format | Description |
|---------|------|--------|-------------|
| count | SINT | tag | tag to store count position as the FOR…DO executes |
|  | INT |  |  |
|  | DINT |  |  |
| initial_value | SINT | tag | must evaluate to a number |
|  | INT | expression | specifies initial value for count |
|  | DINT | immediate |  |
| final_value | SINT | tag | specifies final value for count, which determines when to exit the loop |
|  | INT | expression |  |
|  | DINT | immediate |  |
| increment | SINT | tag | (*optional*) 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`                    If you don't specify an increment, the loop
                                               increments by 1.

```
    DO
    <statement>;
```

optional {   ```
             IF bool_expression THEN
                 EXIT;
             END_IF;
             ```                               If there are conditions when you want to
                                               exit the loop early, use other statements,
                                               such as an IF...THEN construct, to
                                               condition an EXIT statement.

```
END_FOR;
```

The following diagrams show how a FOR...DO loop executes and how an EXIT statement leaves the loop early.



**The FOR…DO loop executes a specific number of times.**



**To stop the loop before the count reaches the last value, use an EXIT statement.**

**Arithmetic Status Flags:**  not affected

**Fault Conditions:**

| A Major Fault Will Occur If | Fault Type | Fault Code |
|---|---|---|
| the construct loops too long | 6 | 1 |

**Example 1:**

| If You Want This | Enter This Structured Text |
|---|---|
| Clear bits 0 - 31 in an array of BOOLs:<br><br>1. Initialize the *subscript* tag to 0.<br><br>2. Clear *array[ subscript ]*. For example, when *subscript* = 5, clear *array[5].*<br><br>3. Add 1 to *subscript*.<br><br>4. If *subscript* is ≤ to 31, repeat 2 and 3.<br><br>Otherwise, stop. | ```For subscript:=0 to 31 by 1 do`<br>    `array[subscript] := 0;`<br>`End_for;``` |

**Example 2:**

| If You Want This | Enter This Structured Text |
|---|---|
| A user-defined data type (structure) stores the following information about an item in your inventory:<br><br>• Barcode ID of the item (string data type)<br>• Quantity in stock of the item (DINT data type)<br><br>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.<br><br>1. Get the size (number of items) of the *Inventory* array and store the result in *Inventory_Items* (DINT tag).<br><br>2. Initialize the *position* tag to 0.<br><br>3. If *Barcode* matches the ID of an item in the array, then:<br><br>  a. Set the *Quantity* tag = *Inventory[position].Qty.* This produces the quantity in stock of the item.<br><br>  b. Stop.<br><br>*Barcode* is a string tag that stores the bar code of the item for which you are searching. For example, when *position* = 5, compare *Barcode* to *Inventory[5].ID.*<br><br>4. Add 1 to *position.*<br><br>5. If *position* is ≤ to (*Inventory_Items -1)*, repeat 3 and 4. Since element numbers start at 0, the last element is 1 less than the number of elements in the array.<br><br>Otherwise, stop. | ```SIZE(Inventory,0,Inventory_Items);`<br>`For position:=0 to Inventory_Items - 1 do`<br>    `If Barcode = Inventory[position].ID then`<br>        `Quantity := Inventory[position].Qty;`<br>        `Exit;`<br>    `End_if;`<br>`End_for;``` |

## WHILE…DO

Use the WHILE…DO loop to keep doing something as long as certain conditions are true.

### Operands:

⊞

```
WHILE bool_expression DO
     <statement>;
END_WHILE;
```

### Structured Text

| Operand | Type | Format | Enter |
|---------|------|--------|-------|
| bool_ expression | BOOL | tag | BOOL tag or expression that evaluates to a BOOL value |
| | | 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:

```
WHILE bool_expression1 DO
     <statement>;          ◄─────   statements to execute while
                                    bool_expression1 is true

     IF bool_expression2 THEN
        EXIT;              ◄─────   If there are conditions when you want to
optional                           exit the loop early, use other statements,
     END_IF;                       such as an IF...THEN construct, to
                                    condition an EXIT statement.
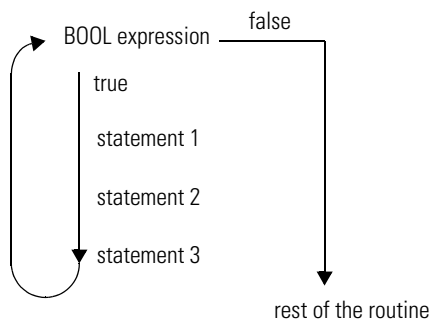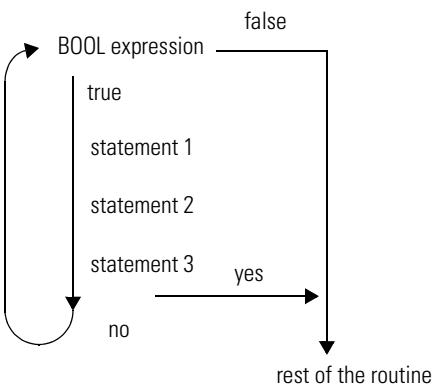
END_WHILE;
```

The following diagrams show how a WHILE...DO loop executes and how an EXIT statement leaves the loop early.

**While the *bool_expression* is true, the controller executes only the statements within the WHILE…DO loop.**

**To stop the loop before the conditions are true, use an EXIT statement.**

**Arithmetic Status Flags:**  not affected

**Fault Conditions:**

| A Major Fault Will Occur If | Fault Type | Fault Code |
|---|---|---|
| the construct loops too long | 6 | 1 |

**Example 1:**

| If You Want This | Enter This Structured Text |
|---|---|
| The WHILE...DO loop evaluates its conditions first. If the conditions are true, the controller then executes the statements within the loop.<br><br>This differs from the REPEAT...UNTIL loop because the REPEAT...UNTIL loop executes the statements in the construct and then determines if the conditions are true before executing the statements again. The statements in a REPEAT...UNTIL loop are always executed at least once. The statements in a WHILE...DO loop might never be executed. | ```pos := 0;``` <br><br>```While ((pos <= 100) & structarray[pos].value <> targetvalue)) do```<br><br>```    pos := pos + 2;```<br><br>```    String_tag.DATA[pos] := SINT_array[pos];```<br><br>```end_while;``` |

**Example 2:**

| If You Want This | Enter This Structured Text |
|---|---|
| Move ASCII characters from a SINT array  into a string tag. (In a SINT array, each element holds one character.) Stop when you reach the carriage return. | ```<br>element_number := 0;<br>SIZE(SINT_array, 0, SINT_array_size);<br>While SINT_array[element_number] <> 13 do<br>    String_tag.DATA[element_number] :=<br>    SINT_array[element_number];<br>    element_number := element_number + 1;<br>    String_tag.LEN := element_number;<br>    If element_number = SINT_array_size then<br>        exit;<br>    end_if;<br>end_while;<br>``` |
| 1. Initialize *Element_number* to 0. | |
| 2. Count the number of elements in *SINT_array* (array that contains the ASCII characters) and store the result in *SINT_array_size* (DINT tag). | |
| 3. If the character at *SINT_array[element_number]* = 13 (decimal value of the carriage return), then stop. | |
| 4. Set *String_tag[element_number]* = the character at *SINT_array[element_number].* | |
| 5. Add 1 to *element_number*. This lets the controller check the next character in *SINT_array*. | |
| 6. Set the Length member of *String_tag* = *element_number*. (This records the number of characters in *String_tag* so far.) | |
| 7. If *element_number* = *SINT_array_size,* then stop. (You are at the end of the array and it does not contain a carriage return.) | |
| 8. Go to 3. | |

## 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<br><br>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>;                    ←——  statements to execute while
                                         bool_expression1 is false
    IF bool_expression2 THEN
        EXIT;                       ←——  If there are conditions when you want to
    END_IF;                              exit the loop early, use other statements,
                                         such as an IF…THEN construct, to
optional                                 condition an EXIT statement.
    UNTIL bool_expression1
END_REPEAT;
```

The following diagrams show how a REPEAT...UNTIL loop executes and how an EXIT statement leaves the loop early.

statement 1

statement 2

statement 3
BOOL expression — true

false

rest of the routine

statement 1

statement 2

statement 3    yes

no

BOOL expression — true

false

rest of the routine

**While the `bool_expression` is false, the controller executes only the statements within the REPEAT…UNTIL loop.**

**To stop the loop before the conditions are false, use an EXIT statement.**

**Arithmetic Status Flags:**    not affected

**Fault Conditions:**

| A Major Fault Will Occur If | Fault Type | Fault Code |
|---|---|---|
| the construct loops too long | 6 | 1 |

**Example 1:**

| If You Want This | Enter This Structured Text |
|---|---|
| The REPEAT...UNTIL loop executes the statements in the construct and then determines if the conditions are true before executing the statements again.<br><br>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. | ```<br>pos := -1;<br><br>REPEAT<br><br>    pos := pos + 2;<br><br>UNTIL ((pos = 101) OR<br>(structarray[pos].value = targetvalue))<br><br>end_repeat;<br>``` |

**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.<br><br>1. Initialize *Element_number* to 0.<br><br>2. Count the number of elements in *SINT_array* (array that contains the ASCII characters) and store the result in *SINT_array_size* (DINT tag).<br><br>3. Set *String_tag[element_number]* = the character at *SINT_array[element_number]*.<br><br>4. Add 1 to *element_number*. This lets the controller check the next character in *SINT_array*.<br><br>5. Set the Length member of *String_tag* = *element_number*. (This records the number of characters in *String_tag* so far.)<br><br>6. If *element_number* = *SINT_array_size,* then stop. (You are at the end of the array and it does not contain a carriage return.)<br><br>7. If the character at *SINT_array[element_number]* = 13 (decimal value of the carriage return), then stop.<br><br>Otherwise, go to 3. | ```element_number := 0;``` <br><br>```SIZE(SINT_array, 0, SINT_array_size);``` <br><br>```Repeat``` <br><br>```    String_tag.DATA[element_number] := SINT_array[element_number];``` <br><br>```    element_number := element_number + 1;``` <br><br>```    String_tag.LEN := element_number;``` <br><br>```    If element_number = SINT_array_size then``` <br><br>```        exit;``` <br><br>```    end_if;``` <br><br>```Until SINT_array[element_number] = 13``` <br><br>```end_repeat;``` |

## Comments

To make your structured text easier to interpret, add comments to it.

- Comments let you use plain language to describe how your structured text works.

- Comments do not affect the execution of the structured text.

To add comments to your structured text:

| To Add A Comment | Use One Of These Formats |
| --- | --- |
| on a single line | `//comment` |
| at the end of a line of structured text | `(*comment*)`<br><br><br>`/*comment*/` |
| within a line of structured text | `(*comment*)`<br><br><br>`/*comment*/` |
| that spans more than one line | `(*start of comment . . . end of comment*)`<br><br><br>`/*start of comment . . . end of comment*/` |

For example:

| Format | Example |
|---|---|
| *//comment* | **At the beginning of a line**<br><br>//Check conveyor belt direction<br><br>IF conveyor_direction THEN...<br><br><br>**At the end of a line**<br><br>ELSE //If conveyor isn't moving, set alarm light<br><br>light := 1;<br><br>END_IF; |
| (*comment*) | Sugar.Inlet[:=]1;(*open the inlet*)<br><br><br>IF Sugar.Low (*low level LS*)& Sugar.High (*high level LS*)THEN...<br><br><br>(*Controls the speed of the recirculation pump. The speed depends on the temperature in the tank.*)<br><br>IF tank.temp > 200 THEN... |
| /*comment*/ | Sugar.Inlet:=0;/*close the inlet*/<br><br><br>IF bar_code=65 /*A*/ THEN...<br><br><br>/*Gets the number of elements in the Inventory array and stores the value in the Inventory_Items tag*/<br><br>SIZE(Inventory,0,Inventory_Items); |

# 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 (or fax) it back to us or email us at RADocumentComments@ra.rockwell.com.

Pub. Title/Type    Logix5000 Controllers General Instructions Reference Manual

Cat. No.    *1756-L6x, 1768-L4x, 1769-L3x, 1789-L6x, 1794-L3x, 20D*    Pub. No.    1756-RM003K-EN-P    Pub. Date    July 2008    Part No.

Please complete the sections below. Where applicable, rank the feature (1=needs improvement, 2=satisfactory, and 3=outstanding).

| **Overall Usefulness** | 1 | 2 | 3 | How can we make this publication more useful for you? |
|---|---|---|---|---|

| **Completeness** (all necessary information is provided) | 1 | 2 | 3 | Can we add more information to help you? |
|---|---|---|---|---|

procedure/step    illustration    feature

example    guideline    other

explanation    definition

| **Technical Accuracy** (all provided information is correct) | 1 | 2 | 3 | Can we be more accurate? |
|---|---|---|---|---|

text    illustration

| **Clarity** (all provided information is easy to understand) | 1 | 2 | 3 | How can we make things clearer? |
|---|---|---|---|---|

| **Other Comments** | | | | You can add additional comments on the back of this form. |
|---|---|---|---|---|

Your Name

Your Title/Function

Location/Phone

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:    Rockwell Automation Technical Communications, 1 Allen-Bradley Dr., Mayfield Hts., OH 44124-9705

Fax: 440-646-3525    Email: RADocumentComments@ra.rockwell.com

Other Comments

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

PLEASE FOLD HERE

- - - - - - - - - - - - - - - - - - - - - - - - - - - -

PLEASE REMOVE

**NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES**

# BUSINESS REPLY MAIL
**FIRST-CLASS MAIL PERMIT NO. 18235 CLEVELAND OH**

**POSTAGE WILL BE PAID BY THE ADDRESSEE**

**Rockwell
Automation**

**1 ALLEN-BRADLEY DR
MAYFIELD HEIGHTS OH 44124-9705**

# ASCII Character Codes

| Character | Dec | Hex | Character | Dec | Hex | Character | Dec | Hex | Character | Dec | Hex |
|-----------|-----|-----|-----------|-----|-----|-----------|-----|-----|-----------|-----|-----|
| [ctrl-@] NUL | 0 | $00 | SPACE | 32 | $20 | @ | 64 | $40 | ' | 96 | $60 |
| [ctrl-A] SOH | 1 | $01 | ! | 33 | $21 | A | 65 | $41 | a | 97 | $61 |
| [ctrl-B] STX | 2 | $02 | " | 34 | $22 | B | 66 | $42 | b | 98 | $62 |
| [ctrl-C] ETX | 3 | $03 | # | 35 | $23 | C | 67 | $43 | c | 99 | $63 |
| [ctrl-D] EOT | 4 | $04 | $ | 36 | $24 | D | 68 | $44 | d | 100 | $64 |
| [ctrl-E] ENQ | 5 | $05 | % | 37 | $25 | E | 69 | $45 | e | 101 | $65 |
| [ctrl-F] ACK | 6 | $06 | & | 38 | $26 | F | 70 | $46 | f | 102 | $66 |
| [ctrl-G] BEL | 7 | $07 | ' | 39 | $27 | G | 71 | $47 | g | 103 | $67 |
| [ctrl-H] BS | 8 | $08 | ( | 40 | $28 | H | 72 | $48 | h | 104 | $68 |
| [ctrl-I] HT | 9 | $09 | ) | 41 | $29 | I | 73 | $49 | i | 105 | $69 |
| [ctrl-J] LF | 10 | $l ($0A) | * | 42 | $2A | J | 74 | $4A | j | 106 | $6A |
| [ctrl-K] VT | 11 | $0B | + | 43 | $2B | K | 75 | $4B | k | 107 | $6B |
| [ctrl-L] FF | 12 | $0C | , | 44 | $2C | L | 76 | $4C | l | 108 | $6C |
| [ctrl-M] CR | 13 | $r ($0D) | - | 45 | $2D | M | 77 | $4D | m | 109 | $6D |
| [ctrl-N] SO | 14 | $0E | . | 46 | $2E | N | 78 | $4E | n | 110 | $6E |
| [ctrl-O] SI | 15 | $0F | / | 47 | $2F | O | 79 | $4F | o | 111 | $6F |
| [ctrl-P] DLE | 16 | $10 | 0 | 48 | $30 | P | 80 | $50 | p | 112 | $70 |
| [ctrl-Q] DC1 | 17 | $11 | 1 | 49 | $31 | Q | 81 | $51 | q | 113 | $71 |
| [ctrl-R] DC2 | 18 | $12 | 2 | 50 | $32 | R | 82 | $52 | r | 114 | $72 |
| [ctrl-S] DC3 | 19 | $13 | 3 | 51 | $33 | S | 83 | $53 | s | 115 | $73 |
| [ctrl-T] DC4 | 20 | $14 | 4 | 52 | $34 | T | 84 | $54 | t | 116 | $74 |
| [ctrl-U] NAK | 21 | $15 | 5 | 53 | $35 | U | 85 | $55 | u | 117 | $75 |
| [ctrl-V] SYN | 22 | $16 | 6 | 54 | $36 | V | 86 | $56 | v | 118 | $76 |
| [ctrl-W] ETB | 23 | $17 | 7 | 55 | $37 | W | 87 | $57 | w | 119 | $77 |
| [ctrl-X] CAN | 24 | $18 | 8 | 56 | $38 | X | 88 | $58 | x | 120 | $78 |
| [ctrl-Y] EM | 25 | $19 | 9 | 57 | $39 | Y | 89 | $59 | y | 121 | $79 |
| [ctrl-Z] SUB | 26 | $1A | : | 58 | $3A | Z | 90 | $5A | z | 122 | $7A |
| ctrl-[ ESC | 27 | $1B | ; | 59 | $3B | [ | 91 | $5B | { | 123 | $7B |
| [ctrl-\] FS | 28 | $1C | < | 60 | $3C | \ | 92 | $5C | | | 124 | $7C |
| ctrl-] GS | 29 | $1D | = | 61 | $3D | ] | 93 | $5D | } | 125 | $7D |
| [ctrl-^] RS | 30 | $1E | > | 62 | $3E | ^ | 94 | $5E | ~ | 126 | $7E |
| [ctrl-_] US | 31 | $1F | ? | 63 | $3F | _ | 95 | $5F | DEL | 127 | $7F |

## Rockwell Automation Support

Rockwell Automation provides technical information on the web to assist you in using 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<br>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**